勉強会資料

自然言語処理

2021/6/30

青山学院大学

（参考資料）山内長承：Pythonによるテキストマイニング入門，オーム社（2017）.

本日の内容

1. テキストデータの取得と前処理
2. 自然言語処理
3. 次元削減
4. 可視化

1.データの取得と前処理
テキストマイニングとは？

- 今まで（データマイニング）
  - 都道府県に関する統計データを取得して可視化
  - それぞれの数値データに対して基本統計量を算出

- テキストマイニングとデータマイニングの違い
  - 文書（テキスト）に対するデータマイニング
  - テキストから特徴を抽出して分析
  - 分析させるためのデータに加工させれば，後は同じ

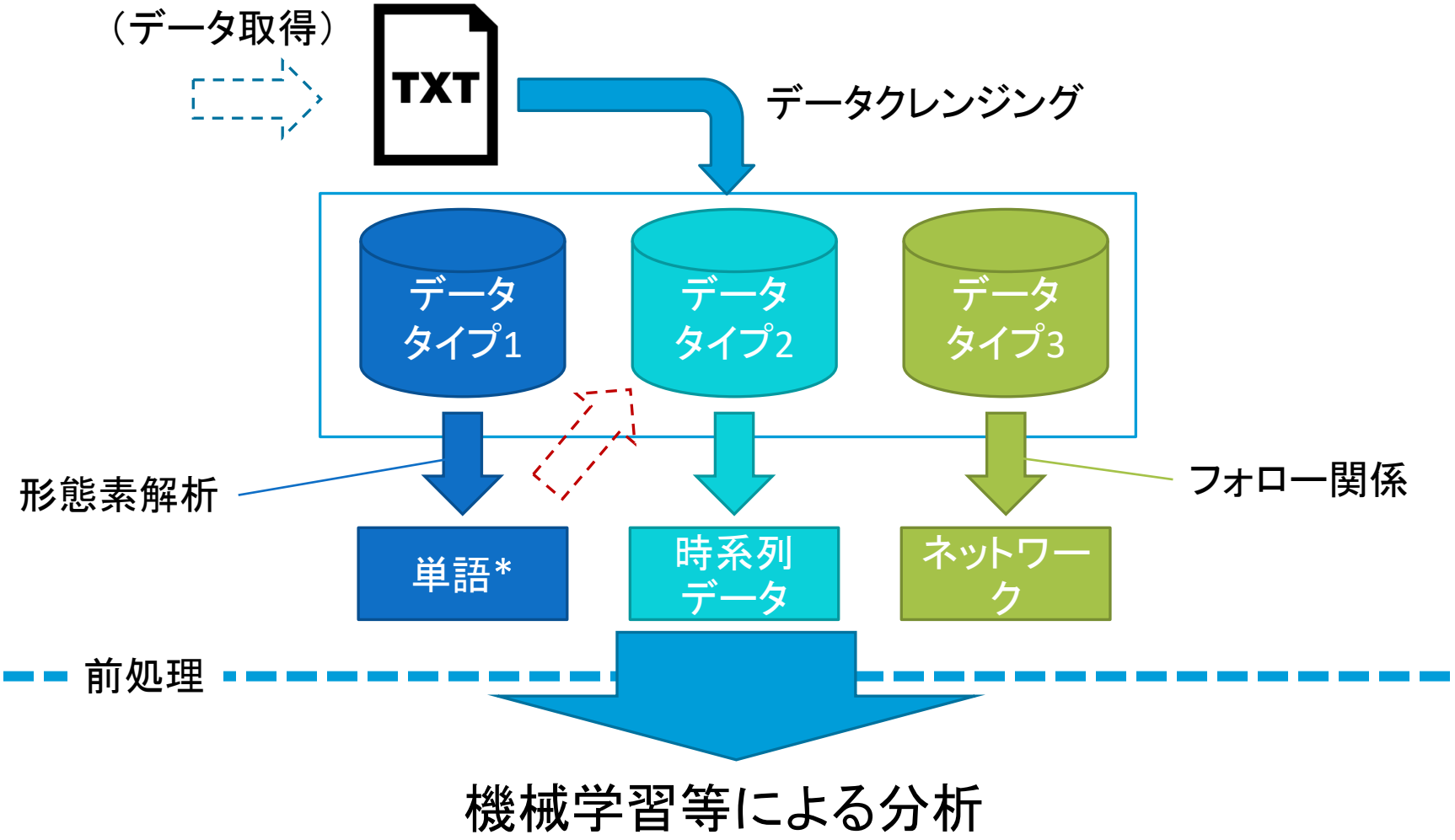
1.データの取得と前処理
データの種類と特徴

- 自由に記述されたテキスト
- 辞書にない未知語，感嘆語，顔文字 etc.

- 短いテキスト，リツイート，リプライ
- リツイート先，リプライの相手の情報 etc.

- 形の整った文章や固有名詞
- 時系列，カテゴリ etc.

1.データの取得と前処理
テキストマイニングの流れ



*厳密には形態素解析されて得られた最小単位の要素を形態素と呼ぶ

1. データの取得と前処理
データの取得

青空文庫のデータ

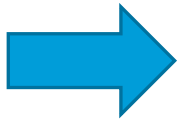
- 著作権の消滅した小説などのデータを公開
- 青空文庫(www.aozora.gr.jp)にアクセスし，
  ~~好きな作品を2つダウンロード（新字新仮名限定）~~
- ~~テキストのzipファイルをダウンロードして解凍~~
- ~~解凍したフォルダにxhtmlファイルを保存~~

⇒ 2020年度は上記のようにして取得した
「吾輩は猫である」の txt と html をコードと共に配布

1. データの取得と前処理
データのクレンジング（1）

- 青空文庫のデータ（txtファイル）
  - ルビが含まれている
  - 前半に注意事項が書かれている

- 青空文庫のデータ（htmlファイル）
  - ルビが含まれている（htmlタグ含む）
  - 様々なhtmlタグが含まれている



不必要なデータを削除する必要

1. データの取得と前処理
データのクレンジング（2）

- テキストデータのクレンジング
  - code200624 の中のaozora.pyを使ってテキスト抽出
  - 使い方：aozora.pyとテキストファイルを置いて以下を実行

```
from aozora import Aozora
aozora = Aozora("ファイル名.txt")
for u in aozora.read():
    print(u)
```

- aozora.pyの解説
  - 青空文庫のテキストファイルの不必要な部分を削除
  - 正規表現を利用

1. データの取得と前処理
正規表現について

正規表現･･･文字列の集合を一つの文字列に統一した表現
⇒ 正規表現の利用によって，コードを簡素化が可能

正規表現の例[1]

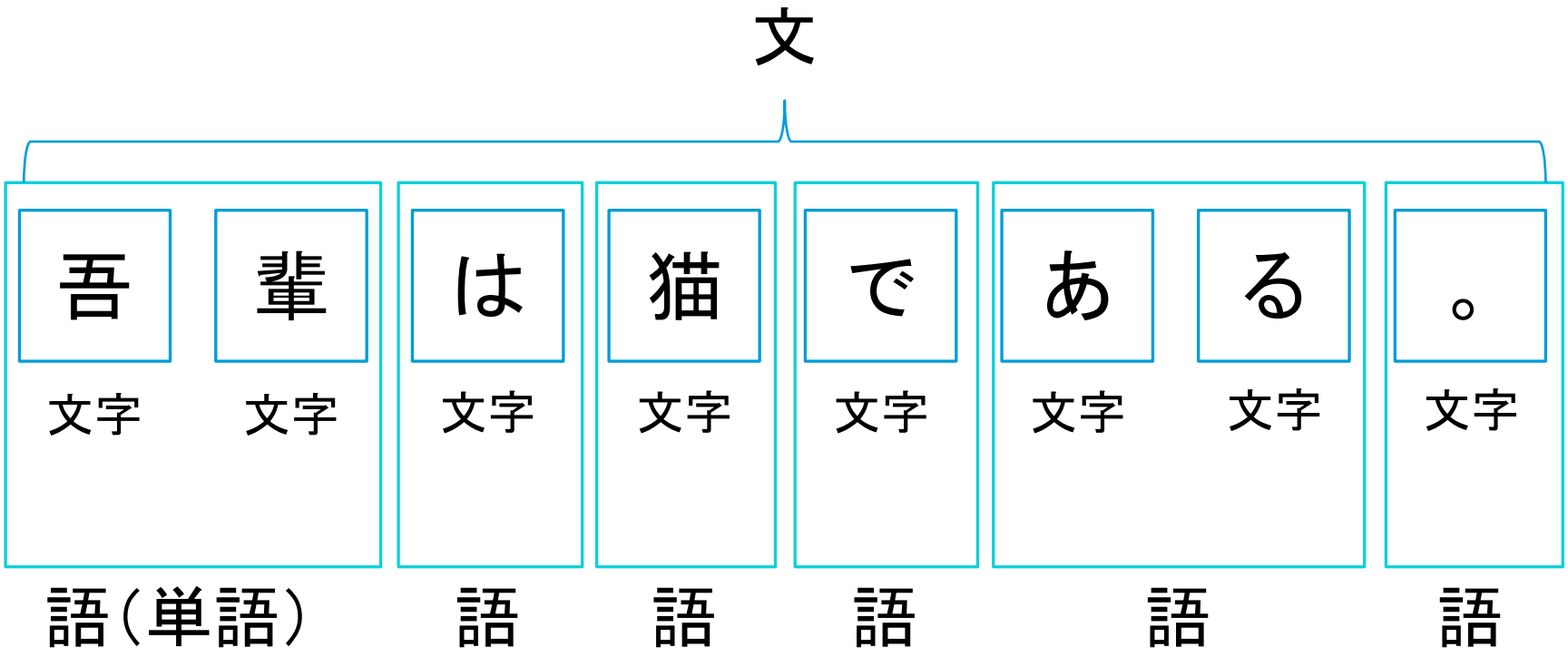
文字	説明	例	match	unmatch
.	任意の一文字	a.c	abc, acc	abbc
^	文字列の先頭	^abc	abcd	dabc
$	文字列の末尾	abc$	dabc	abcd
[]	集合	[a-b]	a, b	c, d
|	和集合	a|b	a, b	c, d

※公式ドキュメントが少し見にくいので，他のまとまった情報を山崎は見ています．上記表も下記サイトから引用

[1] https://qiita.com/luohao0404/items/7135b2b96f9b0b196bf3

2. 自然言語処理
形態素解析(1)

- 形態素解析とは
  - 単語の最小単位である形態素への分割
  - 辞書に基づいて分割される(辞書:品詞等の情報含む)



2. 自然言語処理
形態素解析(2)

- 日本語形態素解析
  - 英語と異なり，分かち書きされていない
  - 単語の切り出しだけでなく品詞等の情報を獲得可能
    ⇒ 英語でも品詞がほしい場合には形態素解析が必要
- 形態素解析器について
  多様な解析器が存在し，環境・用途で使い分ける必要
  - OS によって install のしやすさが異なる
  - 高精度な解析器ほど解析速度が犠牲に
  - テキストにあった辞書選択(NElogd)が必要
- 形態素解析器の例※
  MeCab, JUMAN[2], Sudachi[3], kuromoji, ChaSen, janome

※詳しくは，[2] https://www.slideshare.net/eiennohito/juman-v2-a-practical-and-modern-morphological-analyzer
または [3] https://github.com/WorksApplications/Sudachi

2. 自然言語処理
形態素解析を使わない場合

文の解析に必ずしも形態素解析は必要でない？
e.g.）文字単位，sub word 単位

形態素による分割

90%以上の高い精度を誇るが未知語や低頻度語への対応が困難．語彙数が多くモデルが肥大化

➡ 未知語を補完したり，語彙数を限定した方が良いのでは？

「Hello World」の場合

- 単語単位（あるいは形態素単位）
  Hello, World
- 文字単位
  H, e, l, l, o, W, o, r, l, d
- Sub word 単位（単語が分かれても良い）[4]
  Hello, _Wor, ld

[4] https://github.com/google/sentencepiece

2. 自然言語処理
形態素の取得

今回は janome を使うので，pip install janome を実行

Exercise 1 : 早速試す（ma.examples.py を実行）

```
from janome.tokenizer import Tokenizer
t = Tokenizer()
for token in t.tokenize('すももももももものうち'):
    print(token)
```

※タブ挿入注意！

```
すもも	名詞,一般,*,*,*,*,すもも,スモモ,スモモ
も	助詞,係助詞,*,*,*,*,も,モ,モ
もも	名詞,一般,*,*,*,*,もも,モモ,モモ
も	助詞,係助詞,*,*,*,*,も,モ,モ
もも	名詞,一般,*,*,*,*,もも,モモ,モモ
の	助詞,連体化,*,*,*,*,の,ノ,ノ
うち	名詞,非自立,副詞可能,*,*,*,うち,ウチ,ウチ
```

2. 自然言語処理
janomeの出力内容

すもも　名詞,一般,＊,＊,＊,＊,すもも,スモモ,スモモ

- token.surface：元の単語
- token.part_of_speech：品詞，細分類1，2，3
- token.infl_type：活用形
- token.infl_form：活用型
- token.base_form：原型
- token.reading：読み
- token.phonetic：発音

janomeの品詞体系は，ChaSen（IPA）品詞体系
詳しくは http://www.unixuser.org/~euske/doc/postag/

2. 自然言語処理
名詞の抽出

- Exercise 2：単語リストの出力
  - 品詞体系を確認しながら，品詞ごとに単語リストをファイル出力するプログラムを書いてみましょう
  - token.part_of_speechで取得した文字列に対して，最初のカンマより前の部分（品詞）を抽出
  - 品詞で判定しながらリストにデータを追加して，集合演算で重複を除去
  - それぞれのリストをそれぞれのファイルに出力

```
f = open('text.txt', 'w') # 書き込みモードで開く
f.write(str) # strの文字列をファイルに書き込む
f.close() # ファイルを閉じる
```

※解析結果は，csv や sql などのデータベースへの保存を推奨

2. 自然言語処理
名詞抽出で起こったこと

- 抽出した名詞には意味のない名詞も存在
  - 漢数字（一，二，十 etc.）
  - 代名詞（これ，それ etc.）
  - （恐らくなかったはずだが）顔文字

- データクレンジング（Step 2）
  - 分析する際に邪魔な単語を除去
  - 邪魔な単語をストップワードリストに追加
  - 恣意的な判断での削除ではなく，客観的に見て必要ないと判断できる単語は削除

2. 自然言語処理
単語の正規化（名寄せ）

今までと同様，正解ではなく対処法の一つとして

- カタカナ・ひらがなど読みによる統一
- 原型による統一

表層	原型	活用形
好ん	好む	動詞
好み	好む	動詞
好む	好む	動詞

- 言語資源を用いた統一

  日本語 WordNet などでは同義語が定義されている
  ※同義語の定義の仕方も言語資源によって異なるため
  　定義については必ず確認してから利用すべき

2. 自然言語処理
ストップワードへの対処

- ストップワードリストの読み込み
  ストップワードリストを公開している場合も※
- ストップワードリストの作成
  邪魔な単語をストップワードリストとして保存し，ストップワードが出てきた際に削除

```
# ストップワードリストを読み込んでリストに追加
file = open(r'stopwordlist.txt','r',encoding="utf-8")
stop_word=[]
for data in file.read():
        stop_word.append(data)
```

```
if token.surface not in stop_word:
        # 単語がストップワードにない場合の処理
```

※ストップワードには正解が存在するわけではないことに注意

2. 自然言語処理
N-gram(1)

- 形態素解析で得られた単語リスト
  - テキストを表現しそうな単語を抽出
  - テキストの順序情報は破壊 → Bag of Words(BoW)

- 順序情報を維持した特徴
  - N-gram(N 個の文字・単語の連なり)
  - 文字または単語の連続性を保持可能
  - 文字 N-gram
  - 単語 N-gram

2. 自然言語処理
N-gram(2)

- 文字 N-gram
  - 文字の N 個の連なり
  - 「吾輩は猫である」の文字2-gram(バイグラム)

「吾-輩」「輩-は」「は-猫」「猫-で」「で-あ」「あ-る」

- 単語 N-gram
  - 単語の N 個の連なり
  - 「吾輩は猫である」の単語2-gram

    「吾輩-は」「は-猫」「猫-で」「で-ある」

2. 自然言語処理
N -gram（3）

- Exercise 4
  - 「吾輩は猫である」の単語2−gramの頻度を計算　してみましょう
  - 注意：名詞以外の単語も使ってください
  - 出現頻度の多い上位10個を出力してみましょう
- Exercise 4 (Advanced)
  - 任意のNを入力することで単語 N −gramとその頻度を出力するプログラムを作ってみましょう

結果：
会話の接続が多いように見える（文の分割が正しくないとも）

('」-「', 1780), ('し-て', 1233), ('て-いる', 1102), ('」-と', 1046), ('で-ある', 959), ('ある-。', 946), ('て-、', 940), ('た-。', 834), ('ない-。', 764), ('が-、', 743)

2. 自然言語処理
共起（コロケーション）

- 文や段落を1単位に，同時に生起する単語ペア
  - 「今日は天気が良いので、散歩に行った。」
  - 共起する名詞は「今日」「天気」「散歩」
  - （今日，天気），（今日，散歩），（天気，散歩）

- 共起は何に使うのか？
  →「花」と「咲く」の関係性
  - 「花」と共起する単語は多いが，「咲く」と共起する単語は「花」である確率が高い

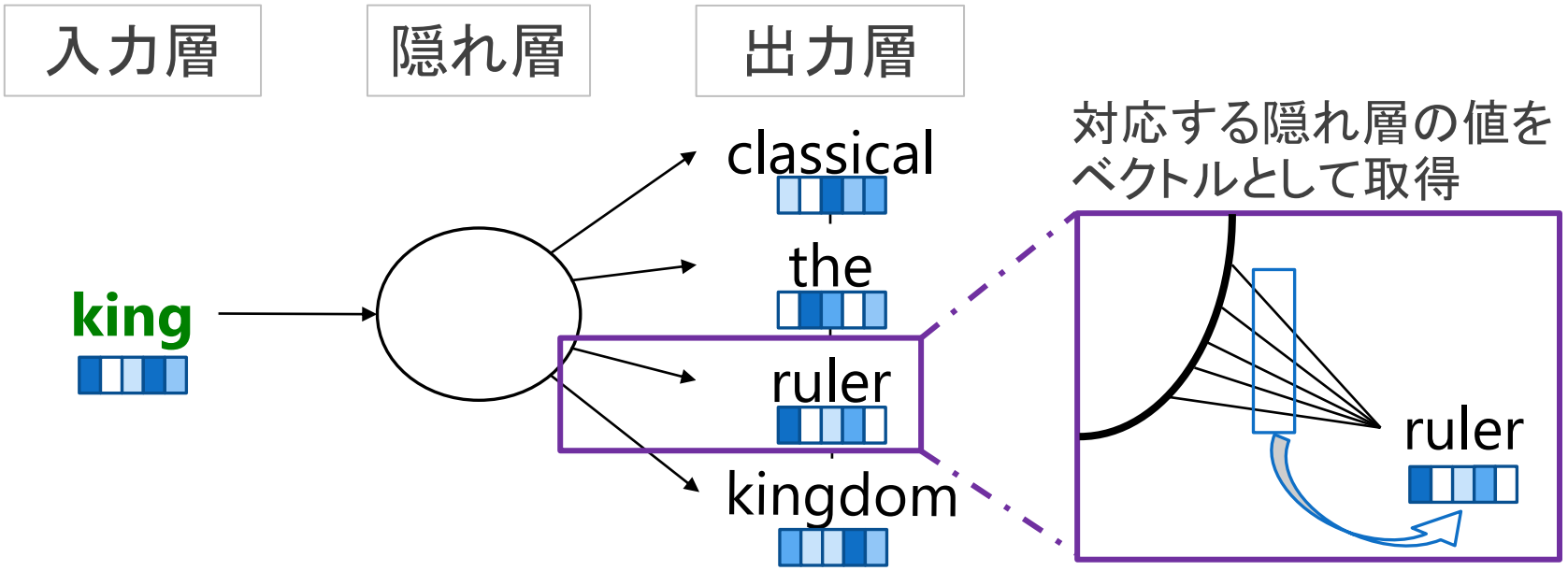
2. 自然言語処理
単語分散表現・単語埋め込み表現

似た文脈をもつ（共起性の高い）単語の類似度（内積）が高くなるようベクトルを学習し，語彙数よりも少ない次元数で単語を表現

学習されたベクトル ＝ 出現するすべての文脈を平均化（近似）したもの

e.g.) word2vec [1] の Skip-Gram における単語 king の学習

入力文 | classical European feudalism, the title of **king** as the ruler of a kingdom



※word2vec の他にも Glove や fasttext などがある

2. 自然言語処理
TF-IDF（1）

TF-IDF・・・文のベクトル表現を獲得する手法

各文章に固有の単語の重要度を高く設定する評価手法

- TF（Term Frequency）：単語の出現頻度
- IDF（Inverse Document Frequency）：逆文書頻度

式の定義（一例*）$tfidf = tf_{i,j} \times idf_i$

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \qquad idf_i = \log \frac{|D|}{|d : t_i \in d|}$$

- $n_{i,j}$：単語 t_i のテキスト d_j における出現回数
- $tf_{i,j}$ の分母：文書 d_j における全単語の出現回数の和
- $|D|$：総テキスト数
- $|\{d : t_i \in d\}|$：単語 t_i を含むテキスト数

＊一例としたのは，文献等によって定義が変わるため．

※2019年度はクラスタ単位で算出する TF-ICF を使っていた人がいた

2. 自然言語処理
TF-IDF（2）

TF-IDFの計算例

A:「白，黒，赤」，B:「白，白，黒」

C:「白，黒，黒，黒」，D:「白」

出現頻度

	白	赤	黒
A	1	1	1
B	2	0	1
C	1	0	3
D	1	0	0

TF-IDF

	白	赤	黒
A	0.1	1.0	0.3
B	0.2	0.0	0.3
C	0.1	0.0	0.7
D	0.1	0.0	0.0

単語頻度のベクトル表現

白の次元　赤の次元　黒の次元

2. 自然言語処理
TF-IDF（3）

- Exercise 5
  - 「吾輩は猫である」の先頭3行のTF-IDFを算出
  - [wagahai-TFIDF.py]を実行してみましょう

	ある	か	が	た	つか	で	とんと	どこ	ぬ
1行目	1.69	0	0	0	0	1.29	0	0	0
2行目	0	0	0	0	0	0	0	0	0
3行目	0	1.69	1.69	1.69	1.69	1.29	1.69	1.69	1.69

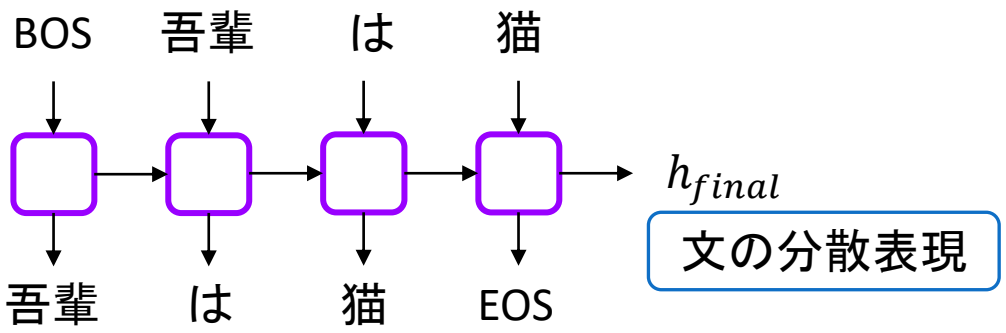
…

は	まだ	名前	吾輩	無い	猫	生まれ	見当
1.29	0	0	1.69	0	1.69	0	0
1.29	1.69	1.69	0	1.69	0	0	0
0	0	0	0	0	0	1.69	1.69

2. 自然言語処理
文の埋め込み表現

単語単位ではなく文の埋め込み表現も存在

- 単語分散表現の平均やTF-IDFによって表現
  ⇒ 文特有の単語を強調できれば文の特徴が表現可能
- 文の埋め込み表現を学習するアルゴリズムを使用
  - Doc2Vec などの word2vec を改良したもの
  - Recurrent Neural Network の最終出力層を利用したもの



□: RNN の各 cell
h_0: 0番目の隠れ層の状態
h_{final}: 2層のbiRNNの結合
$\boldsymbol{h} = [h_0, h_1, h_2, h_3]$
BOS: Beginning of Sentence
EOS: End of Sentence

※図のような，順に単語を予測していくモデルは言語モデルとも呼称

2. 自然言語処理
感情分析(1)

- 各単語の極性値を使った分析
  - 文や単語がネガティブかポジティブかを分析
  - ネガポジ分析，センチメント分析と呼ばれることも
  - 単語極性値の辞書データを利用

- 何に使うのか？
  - 記述された内容がポジティブかネガティブかを判断
  - 卒研ではこれを使った人が結構います

2. 自然言語処理
感情分析(2)

- Exercise 6
  - [sentiment.py]を実行してみましょう
  - 感情極性値は，「単語感情極性対応表」を利用
    http://www.lr.pi.titech.ac.jp/~takamura/pndic_ja.html
  - サンプルは「吾輩は猫である」の3番目～8番目に対して感情極性値を出力
  - 単語の極性値がわかるように出力してみましょう

結果：

- 対象語 ['猫'] 語の感情値 [-0.689341] 感情値の合計 -0.689341 語数 1 平均 -0.689341
- 対象語 ['名前'] 語の感情値 [-0.412125] 感情値の合計 -0.412125 語数 1 平均 -0.412125
- 対象語 ['生れ', 'とんと', '見当'] 語の感情値 [-0.0481131, -0.605903, -0.739512] 感情値の合計 -1.3935281 語数 3 平均 -0.4645093666666665

2. 自然言語処理
カバーできていない範囲

自然言語処理の分野は広く扱っていない要素もかなりある

- 係り受け解析
  主語・述語関係のような係り受け関係の解析
- 言語資源
  日本語 WordNet や感情辞書などの構築された辞書など
  言語資源章一覧（https://www.anlp.jp/award/shigen.html）
- トピックモデル
  文書のトピックを扱うモデル e.g.) Latent Dirichlet Allocation
- Transformer など近年主流の NN 系の手法
  最低限の知識のみフォローしているので，独学必須

基本的なことは自然言語処理100本ノックで身につくはず
https://nlp100.github.io/ja/

1. 機械学習(2)
主成分分析(1)

- 高次元のデータ
  - 可視化が困難(人間は3次元までしか認識できない)
    → 各次元間で確認することが難しい場合もある
  - 「見えない」次元が隠れていることがある

- 主成分分析とは何か
  - 膨大な次元のデータに対する次元圧縮法
  - データの情報を極力失わずに新たな次元を獲得
  - 次元の圧縮 → データを把握しやすくなる
  - 教師なし学習，先々週説明できなかったので今回説明

3. 次元削減
主成分分析(1)

語彙数が多い場合など，計算量の削減のため次元削減が必要にその一つとして主成分分析を紹介

- 主成分分析の流れ
  - 各次元の平均が0になるように変換(重心の発見)
  - 重心からデータの分散が最大になる方向を計算
  - 分散が最大化する方向が第1主成分
  - 第1主成分の射影を取り除いて次の主成分を計算
  - 求める第 k 主成分まで繰り返す $(n \rightarrow k$, ただし $n>k)$
- Exercise 7
  - irisデータで確認してみましょう
  - irisデータの任意の2次元を散布図で可視化し，主成分分析で2次元まで削減したものと比較してみましょう

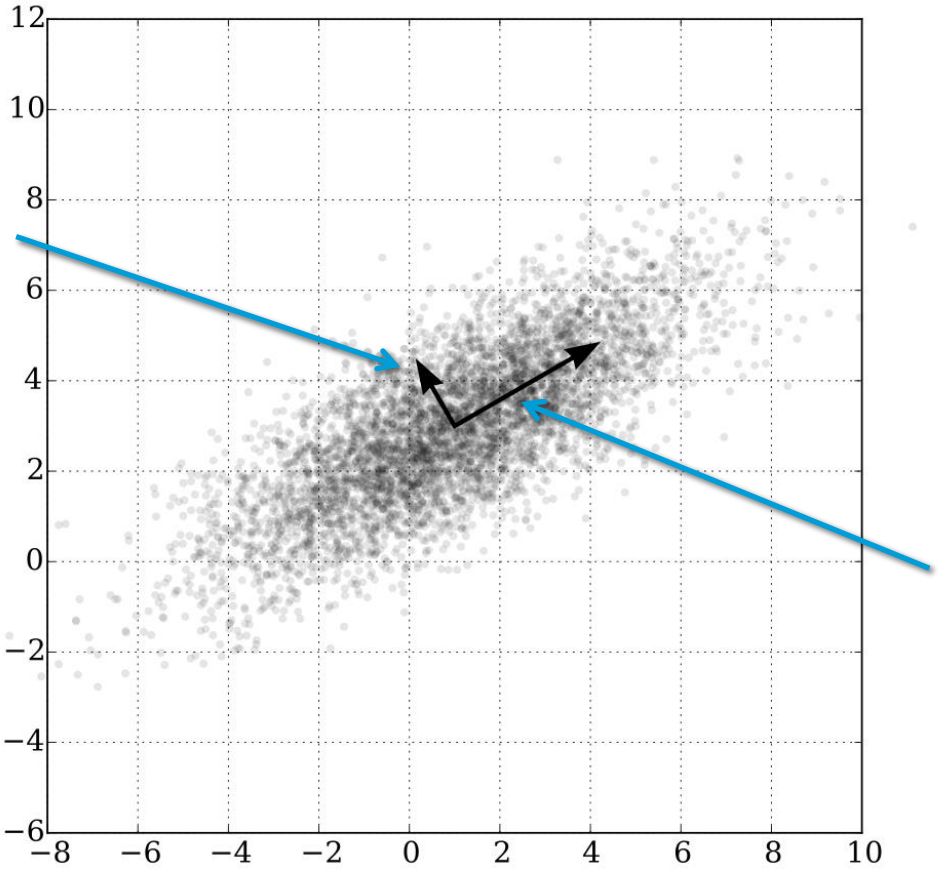
3. 次元削減
主成分分析(2)

```
from sklearn.datasets import load_iris
from sklearn.decomposition import PCA

iris=load_iris() #irisデータの読み込み
pca=PCA(n_components=3)
# 主成分分析の実行
# n_componentsの数値は第n主成分まで次元削減するかを指定

X=iris.data #変数Xにirisのデータ部分を読み込む
pca.fit(X) #主成分分析の適用
Xtran=pca.fit_transform(X)
#適用した主成分に合わせて各データを射影

print(pca.mean_) #主成分の平均
print(pca.components_)  #主成分の中身
print(pca.explained_variance_ratio_)
#各次元の寄与率(第1主成分から加算したものが累積寄与率)
```

上記のソースコードを実行してみましょう

3. 次元削減
主成分分析(3)

第2主成分



第1主成分

図はwikipedia(主成分分析)より拝借

※すいません．可視化は難しくはないんですが元気なく・・・

3. 次元削減
主成分分析(4)

- 主成分分析の結果
  - 第2主成分は第1主成分と直交
  - 共分散行列の固有ベクトル＝主成分ベクトル
  - 得られた主成分はどんな次元？
    → 4つの特徴次元から構成された次元(ラベルなし)
- 主成分分析の他にも様々な手法が存在
  - カテゴリ内外の分散を考慮した直線による分割
    LDA(Linear Discriminant Analysis)
  - 特異値分解ベース
    LSA(Latent Semantic Analysis)
  - 分布ベース
    t-SNE(t-distributed Stochastic Neighbor Embedding)