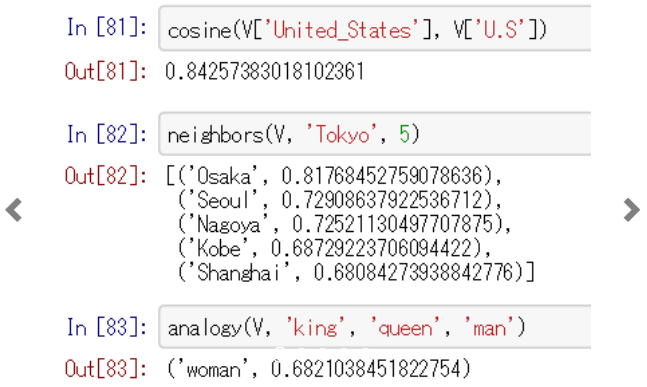

言語処理100本ノック 2015

言語処理100本ノックは，実践的な課題に取り組みながら，プログラミング，データ分析，研究のスキルを楽しく習得することを目指した問題集です

- 実用的でワクワクするような題材を厳選しました
- 言語処理に加えて，統計や機械学習などの周辺分野にも親しめます
- 研究やデータ分析の進め方，作法，スキルを修得できます
- 問題を解くのに必要なデータ・コーパスを配布しています
- 言語はPythonを想定していますが，他の言語にも対応しています

ツイート いいね！ 652 シェア

第1章: 準備運動

テキストや文字列を扱う題材に取り組みながら，プログラミング言語のやや高度なトピックを復習します．

文字列, ユニコード, リスト型, 辞書型, 集合型, イテレータ, スライス, 乱数

第2章: UNIXコマンドの基礎

研究やデータ分析において便利なUNIXツールを体験します．これらの再実装を通じて，プログラミング能力を高めつつ，既存のツールのエコシステムを体感します．

head, tail, cut, paste, split, sort, uniq, sed, tr, expand

第3章: 正規表現

Wikipediaのページのマークアップ記述に正規表現
適用することで，様々な情報・知識を取り出しま
す．

正規表現, JSON, Wikipedia, InfoBox, ウェブサー
ス

第4章: 形態素解析

夏目漱石の小説『吾輩は猫である』に形態素解析器MeCabを適用し，小説中の単語の統計を求めます．

形態素解析, MeCab (http://mecab.googlecode.com/svn/trunk/mecab/doc/index.html), 品詞, 出現頻度, Zipfの法則, matplotlib (http://matplotlib.org/), Gnuplot (http://www.gnuplot.info/)

第5章: 係り受け解析

『吾輩は猫である』に係り受け解析器CaboChaを適用し，係り受け木の操作と統語的な分析を体験します．

クラス, 係り受け解析, CaboCha (https://code.google.com/p/cabocha/), 文節, 係り受け, 格, 機能動詞構文, 係り受けパス, Graphviz (http://www.graphviz.org/)

第6章: 英語テキストの処理

Stanford Core NLPを用いた英語のテキスト処理を
通じて，自然言語処理の様々な基盤技術を概観し
す．

Stanford Core NLP (http://nlp.stanford.edu/software/corenlp.shtml),
テミング, 品詞タグ付け, 固有表現抽出, 共参照解
係り受け解析, 句構造解析, S式

第7章: データベース

Key Value Store (KVS) やNoSQLによるデータベースの構築・検索を修得します．また，CGIを用いたデモ・システムを開発します．

LevelDB (http://leveldb.org/), MongoDB (http://www.mongodb.org/), JSON, インデックス, 整列, CGI, テンプレートエンジン

第8章: 機械学習

評判分析器（ポジネガ分析器）を機械学習で構築します．さらに，手法の評価方法を学びます．

評判分析, ストップワード, 機械学習, 素性, ロジスティック回帰, 交差検定, 適合率, 再現率, scikit-learn (http://scikit-learn.org/), Classias (http://www.chokkan.org/software/classias)

第9章: ベクトル空間法 (I)

大規模なコーパスから単語文脈共起行列を求め，
語の意味を表すベクトルを学習します．その単語
クトルを用い，単語の類似度やアナロジーを求め
す．

ベクトル空間法, 分布仮説, 主成分分析, コサイン
似度, 加法構成性, 複合語, NumPy (http://www.numpy.org/), redsvd (https://code.google.com/p/redsvd/)

第10章: ベクトル空間法 (II)

word2vecを用いて単語の意味を表すベクトルを学習し，正解データを用いて評価します．さらに，クラスタリングやベクトルの可視化を体験します．

word2vec (https://code.google.com/p/word2vec/), scikit-learn (http://scikit-learn.org/), k-meansクラスタリング, 階層型クラスタリング, t-SNE (http://lvdmaaten.github.io/tsne/)

2015年版のねらい

2012年の公開以降，様々な研究室・企業で言語処理100本ノックを採用して頂き，様々なご意見を頂戴しました．2015年版では，再配布可能なデータが題材となるように全面的な改訂を行いました．より実践的な題材への置き換えとともに，単語の分散表現に関するテーマを追加しました．

お知らせ

言語処理100本ノックに関するお問い合わせ・ご
問は，岡崎直観 (http://www.chokkan.org/) (okaza
at-ecei.tohoku.ac.jp, @chokkanorg (https://twitter.com/chokkanorg)) までお願いします．言語処理100本ノックを解くために必要なデータ・コーパスはこちらからダウンロード (data/)て
ます．以前の問題は，こちらのページ (http://www.cl.ecei.tohoku.ac.jp/index.php?NLP%20100%20Drill%20Exercises)から参照でき

す.

第1章: 準備運動

00. 文字列の逆順

文字列"stressed"の文字を逆に（末尾から先頭に向かって）並べた文字列を得よ.

01. 「パタトクカシーー」

「パタトクカシーー」という文字列の1,3,5,7文字目を取り出して連結した文字列を得よ.

02. 「パトカー」＋「タクシー」＝「パタトクカシーー」

「パトカー」＋「タクシー」の文字を先頭から交互に連結して文字列「パタトクカシーー」を得よ.

03. 円周率

"Now I need a drink, alcoholic of course, after the heavy lectures involving quantum mechanics."という文を単語に分解し，各単語の（アルファベットの）文字数を先頭から出現
並べたリストを作成せよ.

04. 元素記号

"Hi He Lied Because Boron Could Not Oxidize Fluorine. New Nations Might Also Sign Peace Security Clause. Arthur King Can."という文を単語に分解し，1, 5, 6, 7, 8, 9, 15, 16,
目の単語は先頭の1文字，それ以外の単語は先頭に2文字を取り出し，取り出した文字列から単語の位置（先頭から何番目の単語か）への連想配列（辞書型もしくはマップ型）を
せよ.

05. n-gram

与えられたシーケンス（文字列やリストなど）からn-gramを作る関数を作成せよ．この関数を用い，"I am an NLPer"という文から単語bi-gram，文字bi-gramを得よ.

06. 集合

"paraparaparadise"と"paragraph"に含まれる文字bi-gramの集合を，それぞれ, XとYとして求め，XとYの和集合，積集合，差集合を求めよ．さらに，'se'というbi-gramがXおよ
含まれるかどうかを調べよ.

07. テンプレートによる文生成

引数x, y, zを受け取り「x時のyはz」という文字列を返す関数を実装せよ．さらに，x=12, y="気温", z=22.4として，実行結果を確認せよ.

08. 暗号文

与えられた文字列の各文字を，以下の仕様で変換する関数cipherを実装せよ.

- 英小文字ならば(219 - 文字コード)の文字に置換
- その他の文字はそのまま出力

この関数を用い，英語のメッセージを暗号化・復号化せよ.

09. Typoglycemia

スペースで区切られた単語列に対して，各単語の先頭と末尾の文字は残し，それ以外の文字の順序をランダムに並び替えるプログラムを作成せよ．ただし，長さが４以下の単語
び替えないこととする．適当な英語の文（例えば"I couldn't believe that I could actually understand what I was reading : the phenomenal power of the human mind ."）を与え，
実行結果を確認せよ.

第2章: UNIXコマンドの基礎

hightemp.txt (data/hightemp.txt)は，日本の最高気温の記録を「都道府県」「地点」「°C」「日」のタブ区切り形式で格納したファイルである．以下の処理を行うプログラムを
し，hightemp.txt (data/hightemp.txt)を入力ファイルとして実行せよ．さらに，同様の処理をUNIXコマンドでも実行し，プログラムの実行結果を確認せよ.

10. 行数のカウント

行数をカウントせよ．確認にはwcコマンドを用いよ.

11. タブをスペースに置換

タブ1文字につきスペース1文字に置換せよ．確認にはsedコマンド，trコマンド，もしくはexpandコマンドを用いよ.

12. 1列目をcol1.txtに，2列目をcol2.txtに保存

各行の1列目だけを抜き出したものをcol1.txtに，2列目だけを抜き出したものをcol2.txtとしてファイルに保存せよ．確認にはcutコマンドを用いよ．

13. col1.txtとcol2.txtをマージ

12で作ったcol1.txtとcol2.txtを結合し，元のファイルの1列目と2列目をタブ区切りで並べたテキストファイルを作成せよ．確認にはpasteコマンドを用いよ．

14. 先頭からN行を出力

自然数Nをコマンドライン引数などの手段で受け取り，入力のうち先頭のN行だけを表示せよ．確認にはheadコマンドを用いよ．

15. 末尾のN行を出力

自然数Nをコマンドライン引数などの手段で受け取り，入力のうち末尾のN行だけを表示せよ．確認にはtailコマンドを用いよ．

16. ファイルをN分割する

自然数Nをコマンドライン引数などの手段で受け取り，入力のファイルを行単位でN分割せよ．同様の処理をsplitコマンドで実現せよ．

17. １列目の文字列の異なり

1列目の文字列の種類（異なる文字列の集合）を求めよ．確認にはsort, uniqコマンドを用いよ．

18. 各行を3コラム目の数値の降順にソート

各行を3コラム目の数値の逆順で整列せよ（注意: 各行の内容は変更せずに並び替えよ）．確認にはsortコマンドを用いよ（この問題はコマンドで実行した時の結果と合わなくてい）．

19. 各行の1コラム目の文字列の出現頻度を求め，出現頻度の高い順に並べる

各行の1列目の文字列の出現頻度を求め，その高い順に並べて表示せよ．確認にはcut, uniq, sortコマンドを用いよ．

第3章: 正規表現

Wikipediaの記事を以下のフォーマットで書き出したファイルjawiki-country.json.gz (data/jawiki-country.json.gz)がある．

- 1行に1記事の情報がJSON形式で格納される
- 各行には記事名が"title"キーに，記事本文が"text"キーの辞書オブジェクトに格納され，そのオブジェクトがJSON形式で書き出される
- ファイル全体はgzipで圧縮される

以下の処理を行うプログラムを作成せよ．

20. JSONデータの読み込み

Wikipedia記事のJSONファイルを読み込み，「イギリス」に関する記事本文を表示せよ．問題21-29では，ここで抽出した記事本文に対して実行せよ．

21. カテゴリ名を含む行を抽出

記事中でカテゴリ名を宣言している行を抽出せよ．

22. カテゴリ名の抽出

記事のカテゴリ名を（行単位ではなく名前で）抽出せよ．

23. セクション構造

記事中に含まれるセクション名とそのレベル（例えば"== セクション名 =="なら1）を表示せよ．

24. ファイル参照の抽出

記事から参照されているメディアファイルをすべて抜き出せ．

25. テンプレートの抽出

記事中に含まれる「基礎情報」テンプレートのフィールド名と値を抽出し，辞書オブジェクトとして格納せよ．

26. 強調マークアップの除去

25の処理時に，テンプレートの値からMediaWikiの強調マークアップ（弱い強調，強調，強い強調のすべて）を除去してテキストに変換せよ（参考: マークアップ早見表 (http://ja.wikipedia.org/wiki/Help:%E6%97%A9%E8%A6%8B%E8%A1%A8)）．

27. 内部リンクの除去

26の処理に加えて，テンプレートの値からMediaWikiの内部リンクマークアップを除去し，テキストに変換せよ（参考: マークアップ早見表 (http://ja.wikipedia.org/wiki/Help:%E6%97%A9%E8%A6%8B%E8%A1%A8)）．

28. MediaWikiマークアップの除去

27の処理に加えて，テンプレートの値からMediaWikiマークアップを可能な限り除去し，国の基本情報を整形せよ．

29. 国旗画像のURLを取得する

テンプレートの内容を利用し，国旗画像のURLを取得せよ．（ヒント: MediaWiki API (http://www.mediawiki.org/wiki/API:Main_page/ja)のimageinfo (http://www.mediawiki.org/wiki/API:Properties/ja#imageinfo_.2F_ii)を呼び出して，ファイル参照をURLに変換すればよい）

第4章: 形態素解析

夏目漱石の小説『吾輩は猫である』の文章（neko.txt (data/neko.txt)）をMeCabを使って形態素解析し，その結果をneko.txt.mecabというファイルに保存せよ．このファイルを用いて，以下の問に対応するプログラムを実装せよ．

なお，問題37, 38, 39はmatplotlib (http://matplotlib.org/)もしくはGnuplot (http://www.gnuplot.info/)を用いるとよい．

30. 形態素解析結果の読み込み

形態素解析結果（neko.txt.mecab）を読み込むプログラムを実装せよ．ただし，各形態素は表層形（surface），基本形（base），品詞（pos），品詞細分類1（pos1）をキーとするマッピング型に格納し，1文を形態素（マッピング型）のリストとして表現せよ．第4章の残りの問題では，ここで作ったプログラムを活用せよ．

31. 動詞

動詞の表層形をすべて抽出せよ．

32. 動詞の原形

動詞の原形をすべて抽出せよ．

33. サ変名詞

サ変接続の名詞をすべて抽出せよ．

34. 「AのB」

2つの名詞が「の」で連結されている名詞句を抽出せよ．

35. 名詞の連接

名詞の連接（連続して出現する名詞）を最長一致で抽出せよ．

36. 単語の出現頻度

文章中に出現する単語とその出現頻度を求め，出現頻度の高い順に並べよ．

37. 頻度上位10語

出現頻度が高い10語とその出現頻度をグラフ（例えば棒グラフなど）で表示せよ．

38. ヒストグラム

単語の出現頻度のヒストグラム（横軸に出現頻度，縦軸に出現頻度をとる単語の種類数を棒グラフで表したもの）を描け．

39. Zipfの法則

単語の出現頻度順位を横軸，その出現頻度を縦軸として，両対数グラフをプロットせよ．

第5章: 係り受け解析

夏目漱石の小説『吾輩は猫である』の文章（neko.txt (data/neko.txt)）をCaboChaを使って係り受け解析し，その結果をneko.txt.cabochaというファイルに保存せよ．このファイルを用いて，以下の問に対応するプログラムを実装せよ．

40. 係り受け解析結果の読み込み（形態素）

形態素を表すクラス `Morph` を実装せよ．このクラスは表層形（ `surface` ），基本形（ `base` ），品詞（ `pos` ），品詞細分類1（ `pos1` ）をメンバ変数に持つこととする．さらに，CaboChaの解析結果（neko.txt.cabocha）を読み込み，各文を `Morph` オブジェクトのリストとして表現し，3文目の形態素列を表示せよ．

41. 係り受け解析結果の読み込み（文節・係り受け）

40に加えて，文節を表すクラス `Chunk` を実装せよ．このクラスは形態素（ `Morph` オブジェクト）のリスト（ `morphs` ），係り先文節インデックス番号（ `dst` ），係り元文節インデックス番号のリスト（ `srcs` ）をメンバ変数に持つこととする．さらに，入力テキストのCaboChaの解析結果を読み込み，1文を `Chunk` オブジェクトのリストとして表現し，8の文節の文字列と係り先を表示せよ．第5章の残りの問題では，ここで作ったプログラムを活用せよ．

42. 係り元と係り先の文節の表示

係り元の文節と係り先の文節のテキストをタブ区切り形式ですべて抽出せよ．ただし，句読点などの記号は出力しないようにせよ．

43. 名詞を含む文節が動詞を含む文節に係るものを抽出

名詞を含む文節が，動詞を含む文節に係るとき，これらをタブ区切り形式で抽出せよ．ただし，句読点などの記号は出力しないようにせよ．

44. 係り受け木の可視化

与えられた文の係り受け木を有向グラフとして可視化せよ．可視化には，係り受け木をDOT言語 (http://ja.wikipedia.org/wiki/DOT%E8%A8%80%E8%AA%9E)に変換し，Graph
(http://www.graphviz.org/)を用いるとよい．また，Pythonから有向グラフを直接的に可視化するには，pydot (https://code.google.com/p/pydot/)を使うとよい．

45. 動詞の格パターンの抽出

今回用いている文章をコーパスと見なし，日本語の述語が取りうる格を調査したい． 動詞を述語，動詞に係っている文節の助詞を格と考え，述語と格をタブ区切り形式で出力せ
ただし，出力は以下の仕様を満たすようにせよ．

- 動詞を含む文節において，最左の動詞の基本形を述語とする
- 述語に係る助詞を格とする
- 述語に係る助詞（文節）が複数あるときは，すべての助詞をスペース区切りで辞書順に並べる

「吾輩はここで始めて人間というものを見た」という例文（neko.txt.cabochaの8文目）を考える． この文は「始める」と「見る」の２つの動詞を含み，「始める」に係る文節
「ここで」，「見る」に係る文節は「吾輩は」と「ものを」と解析された場合は，次のような出力になるはずである．

```
始める　で
見る　　は を
```

このプログラムの出力をファイルに保存し，以下の事項をUNIXコマンドを用いて確認せよ．

- コーパス中で頻出する述語と格パターンの組み合わせ
- 「する」「見る」「与える」という動詞の格パターン（コーパス中で出現頻度の高い順に並べよ）

46. 動詞の格フレーム情報の抽出

45のプログラムを改変し，述語と格パターンに続けて項（述語に係っている文節そのもの）をタブ区切り形式で出力せよ．45の仕様に加えて，以下の仕様を満たすようにせよ．

- 項は述語に係っている文節の単語列とする（末尾の助詞を取り除く必要はない）
- 述語に係る文節が複数あるときは，助詞と同一の基準・順序でスペース区切りで並べる

「吾輩はここで始めて人間というものを見た」という例文（neko.txt.cabochaの8文目）を考える． この文は「始める」と「見る」の２つの動詞を含み，「始める」に係る文節
「ここで」，「見る」に係る文節は「吾輩は」と「ものを」と解析された場合は，次のような出力になるはずである．

```
始める　で　　ここで
見る　　は を　吾輩は ものを
```

47. 機能動詞構文のマイニング

動詞のヲ格にサ変接続名詞が入っている場合のみに着目したい．46のプログラムを以下の仕様を満たすように改変せよ．

- 「サ変接続名詞+を（助詞）」で構成される文節が動詞に係る場合のみを対象とする
- 述語は「サ変接続名詞+を+動詞の基本形」とし，文節中に複数の動詞があるときは，最左の動詞を用いる
- 述語に係る助詞（文節）が複数あるときは，すべての助詞をスペース区切りで辞書順に並べる
- 述語に係る文節が複数ある場合は，すべての項をスペース区切りで並べる（助詞の並び順と揃えよ）

例えば「別段くるにも及ばんさと、主人は手紙に返事をする。」という文から，以下の出力が得られるはずである．

```
返事をする　　と に は　　及ばんさと 手紙に 主人は
```

このプログラムの出力をファイルに保存し，以下の事項をUNIXコマンドを用いて確認せよ．

- コーパス中で頻出する述語（サ変接続名詞+を+動詞）
- コーパス中で頻出する述語と助詞パターン

48. 名詞から根へのパスの抽出

文中のすべての名詞を含む文節に対し，その文節から構文木の根に至るパスを抽出せよ． ただし，構文木上のパスは以下の仕様を満たすものとする．

- 各文節は（表層形の）形態素列で表現する
- パスの開始文節から終了文節に至るまで，各文節の表現を" -> "で連結する

「吾輩はここで始めて人間というものを見た」という文（neko.txt.cabochaの8文目）から，次のような出力が得られるはずである．

```
吾輩は -> 見た
ここで -> 始めて -> 人間という -> ものを -> 見た
人間という -> ものを -> 見た
ものを -> 見た
```

49. 名詞間の係り受けパスの抽出

文中のすべての名詞句のペアを結ぶ最短係り受けパスを抽出せよ．ただし，名詞句ペアの文節番号がiとj（$i < j$）のとき，係り受けパスは以下の仕様を満たすものとする．

- 問題48と同様に，パスは開始文節から終了文節に至るまでの各文節の表現（表層形の形態素列）を" -> "で連結して表現する
- 文節iとjに含まれる名詞句はそれぞれ，XとYに置換する

また，係り受けパスの形状は，以下の2通りが考えられる．

- 文節iから構文木の根に至る経路上に文節jが存在する場合: 文節iから文節jのパスを表示
- 上記以外で，文節iと文節jから構文木の根に至る経路上で共通の文節kで交わる場合: 文節iから文節kに至る直前のパスと文節jから文節kに至る直前までのパス，文節kの内容を" | "で連結して表示

例えば，「吾輩はここで始めて人間というものを見た。」という文（neko.txt.cabochaの8文目）から，次のような出力が得られるはずである．

```
Xは | Yで -> 始めて -> 人間という -> ものを | 見た
Xは | Yという -> ものを | 見た
Xは | Yを | 見た
Xで -> 始めて -> Y
Xで -> 始めて -> 人間という -> Y
Xという -> Y
```

第6章: 英語テキストの処理

英語のテキスト（nlp.txt (data/nlp.txt)）に対して，以下の処理を実行せよ．

50. 文区切り

(. or ; or : or ? or !) → 空白文字 → 英大文字というパターンを文の区切りと見なし，入力された文書を1行1文の形式で出力せよ．

51. 単語の切り出し

空白を単語の区切りとみなし，50の出力を入力として受け取り，1行1単語の形式で出力せよ．ただし，文の終端では空行を出力せよ．

52. ステミング

51の出力を入力として受け取り，Porterのステミングアルゴリズムを適用し，単語と語幹をタブ区切り形式で出力せよ． Pythonでは，Porterのステミングアルゴリズムの実装としてstemming (https://pypi.python.org/pypi/stemming)モジュールを利用するとよい．

53. Tokenization

Stanford Core NLP (http://nlp.stanford.edu/software/corenlp.shtml)を用い，入力テキストの解析結果をXML形式で得よ．また，このXMLファイルを読み込み，入力テキストを1行1語の形式で出力せよ．

54. 品詞タグ付け

Stanford Core NLPの解析結果XMLを読み込み，単語，レンマ，品詞をタブ区切り形式で出力せよ．

55. 固有表現抽出

入力文中の人名をすべて抜き出せ．

56. 共参照解析

Stanford Core NLPの共参照解析の結果に基づき，文中の参照表現（mention）を代表参照表現（representative mention）に置換せよ．ただし，置換するときは，「代表参照表現（参照表現）」のように，元の参照表現が分かるように配慮せよ．

57. 係り受け解析

Stanford Core NLPの係り受け解析の結果（collapsed-dependencies）を有向グラフとして可視化せよ．可視化には，係り受け木をDOT言語 (http://ja.wikipedia.org/wiki/DOT%E8%A8%80%E8%AA%9E)に変換し，Graphviz (http://www.graphviz.org/)を用いるとよい．また，Pythonから有向グラフを直接的に可視化するには，pydot (https://code.google.com/p/pydot/)を使うとよい．

58. タプルの抽出

Stanford Core NLPの係り受け解析の結果（collapsed-dependencies）に基づき，「主語 述語 目的語」の組をタブ区切り形式で出力せよ．ただし，主語，述語，目的語の定義には以下を参考にせよ．

- 述語: nsubj関係とdobj関係の子（dependant）を持つ単語
- 主語: 述語からnsubj関係にある子（dependent）
- 目的語: 述語からdobj関係にある子（dependent）

59. S式の解析

Stanford Core NLPの句構造解析の結果（S式）を読み込み，文中のすべての名詞句（NP）を表示せよ．入れ子になっている名詞句もすべて表示すること．

第7章: データベース

artist.json.gz (data/artist.json.gz)は，オープンな音楽データベース MusicBrainz (https://musicbrainz.org/)の中で，アーティストに関するものをJSON形式に変換し，gzip形式で圧たファイルである．このファイルには，1アーティストに関する情報が1行にJSON形式で格納されている．JSON形式の概要は以下の通りである．

フィールド	型	内容	例
id	ユニーク識別子	整数	20660
gid	グローバル識別子	文字列	"ecf9f3a3-35e9-4c58-acaa-e707fba45060"
name	アーティスト名	文字列	"Oasis"
sort_name	アーティスト名（辞書順整列用）	文字列	"Oasis"
area	活動場所	文字列	"United Kingdom"
aliases	別名	辞書オブジェクトのリスト	
aliases[].name	別名	文字列	"オアシス"
aliases[].sort_name	別名（整列用）	文字列	"オアシス"
begin	活動開始日	辞書	
begin.year	活動開始年	整数	1991
begin.month	活動開始月	整数	
begin.date	活動開始日	整数	
end	活動終了日	辞書	
end.year	活動終了年	整数	2009
end.month	活動終了月	整数	8
end.date	活動終了日	整数	28
tags	タグ	辞書オブジェクトのリスト	
tags[].count	タグ付けされた回数	整数	1
tags[].value	タグ内容	文字列	"rock"
rating	レーティング	辞書オブジェクト	
rating.count	レーティングの投票数	整数	13
rating.value	レーティングの値（平均値）	整数	86

artist.json.gzのデータをKey-Value-Store (KVS) およびドキュメント志向型データベースに格納・検索することを考える．KVSとしては， LevelDB (http://leveldb.org/)， Redis (http://redis.io/)， KyotoCabinet (http://fallabs.com/kyotocabinet/)等を用いよ．ドキュメント志向型データベースとして， MongoDB (http://www.mongodb.org/)を採用したが，CouchDB (http://couchdb.apache.org/)やRethinkDB (http://rethinkdb.com/)等を用いてもよい．

60. KVSの構築

Key-Value-Store (KVS) を用い，アーティスト名（name）から活動場所（area）を検索するためのデータベースを構築せよ．

61. KVSの検索

60で構築したデータベースを用い，特定の（指定された）アーティストの活動場所を取得せよ．

62. KVS内の反復処理

60で構築したデータベースを用い，活動場所が「Japan」となっているアーティスト数を求めよ．

63. オブジェクトを値に格納したKVS

KVSを用い，アーティスト名（name）からタグと被タグ数（タグ付けされた回数）のリストを検索するためのデータベースを構築せよ．さらに，ここで構築したデータベースをい，アーティスト名からタグと被タグ数を検索せよ．

64. MongoDBの構築

アーティスト情報（artist.json.gz）をデータベースに登録せよ．さらに，次のフィールドでインデックスを作成せよ: name, aliases.name, tags.value, rating.value

65. MongoDBの検索

MongoDBのインタラクティブシェルを用いて，"Queen"というアーティストに関する情報を取得せよ．さらに，これと同様の処理を行うプログラムを実装せよ．

66. 検索件数の取得

MongoDBのインタラクティブシェルを用いて，活動場所が「Japan」となっているアーティスト数を求めよ．

67. 複数のドキュメントの取得

特定の（指定した）別名を持つアーティストを検索せよ．

68. ソート

"dance"というタグを付与されたアーティストの中でレーティングの投票数が多いアーティスト・トップ10を求めよ．

69. Webアプリケーションの作成

ユーザから入力された検索条件に合致するアーティストの情報を表示するWebアプリケーションを作成せよ．アーティスト名，アーティストの別名，タグ等で検索条件を指定し，ーティスト情報のリストをレーティングの高い順などで整列して表示せよ．

第8章: 機械学習

本章では，Bo Pang氏とLillian Lee氏が公開しているMovie Review Data (http://www.cs.cornell.edu/people/pabo/movie-review-data/)のsentence polarity dataset v1.0 (http://www.cs.cornell.edu/people/pabo/movie-review-data/rt-polaritydata.README.1.0.txt)を用い，文を肯定的（ポジティブ）もしくは否定的（ネガティブ）に分類するタスク性分析）に取り組む．

70. データの入手・整形

文に関する極性分析の正解データ (http://www.cs.cornell.edu/people/pabo/movie-review-data/rt-polaritydata.tar.gz)を用い，以下の要領で正解データ（sentiment.txt）を作成せよ

1. rt-polarity.posの各行の先頭に"+1 "という文字列を追加する（極性ラベル"+1"とスペースに続けて肯定的な文の内容が続く）
2. rt-polarity.negの各行の先頭に"-1 "という文字列を追加する（極性ラベル"-1"とスペースに続けて否定的な文の内容が続く）
3. 上述1と2の内容を結合（concatenate）し，行をランダムに並び替える

sentiment.txtを作成したら，正例（肯定的な文）の数と負例（否定的な文）の数を確認せよ．

71. ストップワード

英語のストップワードのリスト（ストップリスト）を適当に作成せよ．さらに，引数に与えられた単語（文字列）がストップリストに含まれている場合は真，それ以外は偽を返数を実装せよ．さらに，その関数に対するテストを記述せよ．

72. 素性抽出

極性分析に有用そうな素性を各自で設計し，学習データから素性を抽出せよ．素性としては，レビューからストップワードを除去し，各単語をステミング処理したものが最低限ースラインとなるであろう．

73. 学習

72で抽出した素性を用いて，ロジスティック回帰モデルを学習せよ．

74. 予測

73で学習したロジスティック回帰モデルを用い，与えられた文の極性ラベル（正例なら"+1"，負例なら"-1"）と，その予測確率を計算するプログラムを実装せよ．

75. 素性の重み

73で学習したロジスティック回帰モデルの中で，重みの高い素性トップ10と，重みの低い素性トップ10を確認せよ．

76. ラベル付け

学習データに対してロジスティック回帰モデルを適用し，正解のラベル，予測されたラベル，予測確率をタブ区切り形式で出力せよ．

77. 正解率の計測

76の出力を受け取り，予測の正解率，正例に関する適合率，再現率，F1スコアを求めるプログラムを作成せよ．

78. 5分割交差検定

76-77の実験では，学習に用いた事例を評価にも用いたため，正当な評価とは言えない．すなわち，分類器が訓練事例を丸暗記する際の性能を評価しており，モデルの汎化性能定していない．そこで，5分割交差検定により，極性分類の正解率，適合率，再現率，F1スコアを求めよ．

79. 適合率-再現率グラフの描画

ロジスティック回帰モデルの分類の閾値を変化させることで，適合率-再現率グラフを描画せよ．

第9章: ベクトル空間法 (I)

enwiki-20150112-400-r10-105752.txt.bz2 (data/enwiki-20150112-400-r10-105752.txt.bz2)は，2015年1月12日時点の英語のWikipedia記事のうち，約400語以上で構成される記事から，ランダムに1/10サンプリングした105,752記事のテキストをbzip2形式で圧縮したものである．このテキストをコーパスとして，単語の意味を表すベクトル（分散表現）をしたい．第9章の前半では，コーパスから作成した単語文脈共起行列に主成分分析を適用し，単語ベクトルを学習する過程を，いくつかの処理に分けて実装する．第9章の後半で学習で得られた単語ベクトル（300次元）を用い，単語の類似度計算やアナロジー（類推）を行う．

なお，問題83を素直に実装すると，大量（約7GB）の主記憶が必要になる． メモリが不足する場合は，処理を工夫するか，1/100サンプリングのコーパスenwiki-20150112-400-r100-10576.txt.bz2 (data/enwiki-20150112-400-r100-10576.txt.bz2)を用いよ．

80. コーパスの整形

文を単語列に変換する最も単純な方法は，空白文字で単語に区切ることである． ただ，この方法では文末のピリオドや括弧などの記号が単語に含まれてしまう． そこで，コーパの各行のテキストを空白文字でトークンのリストに分割した後，各トークンに以下の処理を施し，単語から記号を除去せよ．

- トークンの先頭と末尾に出現する次の文字を削除: `.,!?;:()[]'"`
- 空文字列となったトークンは削除

以上の処理を適用した後，トークンをスペースで連結してファイルに保存せよ．

81. 複合語からなる国名への対処

英語では，複数の語の連接が意味を成すことがある．例えば，アメリカ合衆国は"United States"，イギリスは"United Kingdom"と表現されるが，"United"や"States"，"Kingdom"う単語だけでは，指し示している概念・実体が曖昧である．そこで，コーパス中に含まれる複合語を認識し，複合語を1語として扱うことで，複合語の意味を推定したい．しかしながら，複合語を正確に認定するのは大変むずかしいので，ここでは複合語からなる国名を認定したい．

インターネット上から国名リストを各自で入手し，80のコーパス中に出現する複合語の国名に関して，スペースをアンダーバーに置換せよ．例えば，"United States"は"United_States"，"Isle of Man"は"Isle_of_Man"になるはずである．

82. 文脈の抽出

81で作成したコーパス中に出現するすべての単語tに関して，単語tと文脈語cのペアをタブ区切り形式ですべて書き出せ．ただし，文脈語の定義は次の通りとする．

- ある単語tの前後d単語を文脈語cとして抽出する（ただし，文脈語に単語tそのものは含まない）
- 単語tを選ぶ度に，文脈幅dは$\{1, 2, 3, 4, 5\}$の範囲でランダムに決める．

83. 単語／文脈の頻度の計測

82の出力を利用し，以下の出現分布，および定数を求めよ．

- $f(t, c)$: 単語tと文脈語cの共起回数
- $f(t, *)$: 単語tの出現回数
- $f(*, c)$: 文脈語cの出現回数
- N: 単語と文脈語のペアの総出現回数

84. 単語文脈行列の作成

83の出力を利用し，単語文脈行列Xを作成せよ．ただし，行列Xの各要素X_{tc}は次のように定義する．

- $f(t, c) \geq 10$ならば，$X_{tc} = \mathrm{PPMI}(t, c) = \max\{\log \frac{N \times f(t,c)}{f(t,*) \times f(*,c)}, 0\}$
- $f(t, c) < 10$ならば，$X_{tc} = 0$

ここで，$\mathrm{PPMI}(t, c)$はPositive Pointwise Mutual Information（正の相互情報量）と呼ばれる統計量である．なお，行列Xの行数・列数は数百万オーダとなり，行列のすべての要主記憶上に載せることは無理なので注意すること．幸い，行列Xのほとんどの要素は0になるので，非0の要素だけを書き出せばよい．

85. 主成分分析による次元圧縮

84で得られた単語文脈行列に対して，主成分分析を適用し，単語の意味ベクトルを300次元に圧縮せよ．

86. 単語ベクトルの表示

85で得た単語の意味ベクトルを読み込み，"United States"のベクトルを表示せよ．ただし，"United States"は内部的には"United_States"と表現されていることに注意せよ．

87. 単語の類似度

85で得た単語の意味ベクトルを読み込み，"United States"と"U.S."のコサイン類似度を計算せよ．ただし，"U.S."は内部的に"U.S"と表現されていることに注意せよ．

88. 類似度の高い単語10件

85で得た単語の意味ベクトルを読み込み，"England"とコサイン類似度が高い10語と，その類似度を出力せよ．

89. 加法構成性によるアナロジー

85で得た単語の意味ベクトルを読み込み，vec("Spain") - vec("Madrid") + vec("Athens")を計算し，そのベクトルと類似度の高い10語とその類似度を出力せよ．

第10章: ベクトル空間法 (II)

第10章では，前章に引き続き単語ベクトルの学習に取り組む．

90. word2vecによる学習

81で作成したコーパスに対してword2vec (https://code.google.com/p/word2vec/)を適用し，単語ベクトルを学習せよ．さらに，学習した単語ベクトルの形式を変換し，86-89のグラムを動かせ．

91. アナロジーデータの準備

単語アナロジーの評価データ (https://word2vec.googlecode.com/svn/trunk/questions-words.txt)をダウンロードせよ．このデータ中で": "で始まる行はセクション名を表す．例えば，": capital-common-countries"という行は，"capital-common-countries"というセクションの開始を表している．ダウンロードした評価データの中で，"family"というセクション含まれる評価事例を抜き出してファイルに保存せよ．

92. アナロジーデータへの適用

91で作成した評価データの各事例に対して，vec(2列目の単語) - vec(1列目の単語) + vec(3列目の単語)を計算し，そのベクトルと類似度が最も高い単語と，その類似度を求めよ．た単語と類似度は，各事例の末尾に追記せよ．このプログラムを85で作成した単語ベクトル，90で作成した単語ベクトルに対して適用せよ．

93. アナロジータスクの正解率の計算

92で作ったデータを用い，各モデルのアナロジータスクの正解率を求めよ．

94. WordSimilarity-353での類似度計算

The WordSimilarity-353 Test Collection (http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/)の評価データを入力とし，1列目と2列目の単語の類似度を計算し，各行尾に類似度の値を追加するプログラムを作成せよ．このプログラムを85で作成した単語ベクトル，90で作成した単語ベクトルに対して適用せよ．

95. WordSimilarity-353での評価

94で作ったデータを用い，各モデルが出力する類似度のランキングと，人間の類似度判定のランキングの間のスピアマン相関係数を計算せよ．

96. 国名に関するベクトルの抽出

word2vecの学習結果から，国名に関するベクトルのみを抜き出せ．

97. k-meansクラスタリング

96の単語ベクトルに対して，k-meansクラスタリングをクラスタ数$k = 5$として実行せよ．

98. Ward法によるクラスタリング

96の単語ベクトルに対して，Ward法による階層型クラスタリングを実行せよ．さらに，クラスタリング結果をデンドログラムとして可視化せよ．

99. t-SNEによる可視化

96の単語ベクトルに対して，ベクトル空間をt-SNEで可視化せよ．

100本ノックで用いるコーパス・データ

- hightemp.txt (data/hightemp.txt): 気象庁 (http://www.jma.go.jp/)が公開している「歴代全国ランキング>観測史上の順位>最高気温の高い方から (http://www.data.jma.go.jp/obd/stats/etrn/view/rankall.php?prec_no=&block_no=&year=&month=&day=&view=)」を基に，手作業でタブ区切り形式に整形したものです．規約等はこちらのページ (http://www.jma.go.jp/jma/kishou/info/coment.html)を参照して下さい．
- jawiki-country.json.gz (data/jawiki-country.json.gz): 2014年10月18日付けの日本語のWikipedia記事のダンプ (http://dumps.wikimedia.org/jawiki/latest/jawiki-latest-pages-articles.xml.bz2)の中から，国家に言及していると思われる記事を抽出し，JSON形式で格納したものです．このファイルは，クリエイティブ・コモンズ 表示-継承 3.0 非移 (http://creativecommons.org/licenses/by-sa/3.0/legalcode)のライセンスで配布されています．
- neko.txt (data/neko.txt): 青空文庫 (http://www.aozora.gr.jp/)で公開されている夏目漱石の長編小説『吾輩は猫である』をテキストファイルに整形したものです．
- nlp.txt (data/nlp.txt): 英語のWikipedia記事"Natural Language Processing" (https://en.wikipedia.org/wiki/Natural_language_processing)を 1 行 1 文形式にまとめたものです．ファイルはクリエイティブ・コモンズ 表示-継承 3.0 非移植 (http://creativecommons.org/licenses/by-sa/3.0/legalcode)のライセンスで配布されています．
- artist.json.gz (data/artist.json.gz): MusicBrainz (https://musicbrainz.org/)が公開しているデータベースのスナップショット (https://musicbrainz.org/doc/MusicBrainz_Database/Download)のうち，アーティストに関するものを抜き出し，JSON形式にまとめたものです．このファイルは クリエイブ・コモンズ 表示 - 非営利 - 継承 3.0 非移植 (http://creativecommons.org/licenses/by-nc-sa/3.0/)のライセンスで配布されています．
- enwiki-20150112-400-r10-105752.txt.bz2 (data/enwiki-20150112-400-r10-105752.txt.bz2): 2015年1月12日時点のWikipedia記事データベースのダンプ（英語） (http://dumps.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2)うち，約400語以上で構成される記事の中から，ランダムに1/10サンプリングした105,752記テキストをbzip2形式で圧縮したものです．このファイルはクリエイティブ・コモンズ 表示-継承 3.0 非移植 (http://creativecommons.org/licenses/by-sa/3.0/legalcode)のンスで配布されています．

Copyright (c) 2012-2015 Naoaki Okazaki (http://www.chokkan.org/), Inui-Okazaki Laboratory (http://www.cl.ecei.tohoku.ac.jp/).
Updated at 2015-03-13 23:01:07 +0900
Named by Jun Sugiura (http://www.cl.ecei.tohoku.ac.jp/~jun-s/).
Inspired by the exercise done in Sato Laboratory (http://sslab.nuee.nagoya-u.ac.jp/).