平成 30 年度 茨城大学工学部情報工学科
卒業研究論文

Word Mover's Distance に基づく類似文検索における クラスタリングによる高速化

2019 年 2 月 4 日

茨城大学 工学部 情報工学科
15T4061L 南濱 篤
指導教員：新納 浩幸 教授

平成 30 年度茨城大学工学部情報工学科卒業研究

Word Mover's Distance に基づく類似文検索における クラスタリングによる高速化

氏名：15T4061L 南濱 篤
指導教員：新納 浩幸 教授

論文要旨

本論文では Word Mover's Distance (以下、WMD) に基づいて文検索を行う際に、検索時間が多大にかかるという問題に対して、検索対象の文集合をクラスタリングしておくことを提案する。

2 つの文の距離（あるいは類似度）を計算する処理は、自然言語処理システムの多くの場面で必要になる。距離を測る手法としては編集距離や N-gram を利用した手法が簡易で実用的である。また Python の difflib も有益なツールである。ただしそれらの手法は文字列の比較が基本となっているために、意味的に類似していても文字列として全く異なる場合、適切な距離が求まらない。例えば「秋田犬を好む」と「ゴールデンレトリバーを愛する」は意味的には似ているが文字列上の一致が少ないために、大きな距離が算出される。

一方、WMD は単語の分散表現を利用して、文同士の距離を計算する手法である。概略、単語間の対応を取って単語間の距離を分散表現から求めるために、類似の意味を持つ単語が文字列上一致していなくても、適切に距離を算出できる。ただし WMD は単語間の対応を取る処理が複雑なため、処理時間の点で問題がある。特に類似文検索に WMD を利用する場合、検索対象が膨大なため検索システムとして実用にならない。

本論文では、上記問題に対して検索対象の文集合を予めクラスタリングしておくことを提案する。クラスタリングにより得られたクラスタ毎に代表点（重心）を選出しておき、クエリの文が入力されたときに、最も距離の近い代表点を探し、その後に対応するクラスタ内の文だけを WMD による検索対象とする。これによって WMD で距離を測る処理が大幅に減らせるために検索時間が短縮される。

実験では 10,000 文を検索対象として 5 つのクエリ文で、本手法を用いた場合と用いない場合の検索時間と検索結果を比較し、提案手法の有効性を示す。

目次

表目次

図目次

第 1 章

はじめに

本論文では Word Mover's Distance (以下、WMD) に基づいて文検索を行う際に、検索時間が多大にかかるという問題に対して、検索対象の文集合をクラスタリングしておくことを提案する [5]。

2 つの文の距離（あるいは類似度）を計算する処理は、自然言語処理システムの多くの場面で必要になる。距離を測る手法としては編集距離や N-gram を利用した手法が簡易で実用的である。また Python の difflib も有益なツールである。ただしそれらの手法は文字列の比較が基本となっているために、意味的に類似していても文字列として全く異なる場合、適切な距離が求まらない。例えば「秋田犬を好む」と「ゴールデンレトリバーを愛する」は意味的には似ているが文字列上の一致が少ないために、大きな距離が算出される。

一方、WMD は単語の分散表現を利用して、文同士の距離を計算する手法である。概略、単語間の対応を取って単語間の距離を分散表現から求めるために、類似の意味を持つ単語が文字列上一致していなくても、適切に距離を算出できる。ただし WMD は単語間の対応を取る処理が複雑なため、処理時間の点で問題がある。特に類似文検索に WMD を利用する場合、検索対象が膨大なため検索システムとして実用にならない。

本論文では、上記問題に対して検索対象の文集合を予めクラスタリングしておくことを提案する。クラスタリングにより得られたクラスタ毎に代表点（重心）を選出しておき、クエリの文が入力されたときに、最も距離の近い代表点を探し、その後に対応するクラスタ内の文だけを WMD による検索対象とする。これによって WMD で距離を測る処理が大幅に減らせるために検索時間が短縮される。

実験では 10,000 文を検索対象として 5 つのクエリ文で、本手法を用いた場合と用いない場合の検索時間と検索結果を比較し、提案手法の有効性を示す。

第 2 章

関連研究

類似文検索を高速化する単純なアプローチとしては、2 文間の類似度を測る処理を高速化することである。その処理が文字列に基づくものであれば、文字列比較のための高速なアルゴリズム [4] を用いれば良い。しかし WMD は文字列を比較しているわけではないので、それらの手法は利用できない。WMD 自体の高速化についてはオリジナルの論文 [1] で議論されている。そこでは単語の対応関係を求める際に全解探索を行わず、上位の数個で探索を止める RWMD が提案されている。

文字列ではなく画像の高速な類似度算出方法としては知覚ハッシュ (Perceptual Hash)[3] が有名である。これはハッシュを生成する元のデータが知覚的に類似しているとハッシュの値が近くなる特徴があるため、ハッシュの値を利用して類似度を測ることができる。

文字列や文あるいは画像などそのデータを問わず、一般の検索自体を高速化する方法として何らかのインデックスを作っておくことは非常に有効である。インデックスを用いて、検索対象のデータを小規模にできれば、検索の大きな効率化になる。本論文ではこのインデックスをクラスタの重心とした手法と言える。

第 3 章

WMD

WMD は単語の分散表現を用いた文同士の距離を計算する手法である。

n 単語の分散表現からなる行列を $\mathbf{X} \in \mathbb{R}^{d \times n}$ とし、単語 i の分散表現を $\mathbf{x}_i \in \mathcal{R}^d$ とする。また、文を正規化された bag-of-words(nBOW) で表現されているベクトル $\mathbf{d} \in \mathcal{R}^n$, 単語 i の文書内での出現回数を c_i とする $d_i = \frac{c_i}{\sum_{j=1}^{n} c_j}$ で表す。

具体的には、以下の 2 つの文 D_0、D_1 が与えられたとする。

$$D_0 \ : \text{The President greets the press in Chicago.}$$
$$D_1 \ : \text{Obama speaks in Illinois.}$$

D_0、D_1 の nBOW ベクトルはそれぞれ以下の d、d' のようになる。

	President	greets	press	Chicago	Obama	speaks	Illinois
$d = ($	$\frac{1}{4}$,	$\frac{1}{4}$,	$\frac{1}{4}$,	$\frac{1}{4}$,	0,	0,	0)
$d' = ($	0,	0,	0,	0,	$\frac{1}{3}$,	$\frac{1}{3}$,	$\frac{1}{3}$)

単語間の距離の算出方法として、分散表現によるユークリッド距離がある。語 i から語 j への距離は $c(i,j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2$ となる。

文同士の距離の算出には、変換行列 $\mathbf{T} \in R^{n \times n}$ を用いる。nBOW で表現された文のベクトル d を d' への変換を考える。d 内の単語 i を d' 内の全てもしくは一部の任意の単語 j 変換するとき、行列 $\mathbf{T}$ の成分 $\mathbf{T}_{ij} \geq 0$ は語 i から語 j への出現頻度の分配量を示す。ゆえに、単語 i に対応する行の成分の総和は変換前の d 内の単語 i の出現頻度 d_i、すなわち、 $\sum_{j=1}^{n} \mathbf{T}_{i,j} = d_i$ となる。また、単語 j に対応する列の総和は変換先の d' 内の単語 j の出現頻度 d'_j、すなわち、$\sum_{i=1}^{n} \mathbf{T}_{i,j} = d'_j$ となる。

具体的に、以下のような d、d' が与えられたとする。

$$d = (\ \tfrac{1}{4},\ \tfrac{1}{4},\ \tfrac{1}{4},\ \tfrac{1}{4},\ 0,\ 0,\ 0\ )$$
$$d' = (\ 0,\ 0,\ 0,\ 0,\ \tfrac{1}{3},\ \tfrac{1}{3},\ \tfrac{1}{3}\ )$$

d から d' への変換を考えたとき、変換行列 $\mathbf{T}$ の一つとして以下のようなものが考えられる。

$$T = \begin{pmatrix} 0 & 0 & 0 & 0 & \frac{1}{12} & \frac{1}{6} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{6} & \frac{1}{12} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{12} & \frac{1}{6} \\ 0 & 0 & 0 & 0 & \frac{1}{12} & 0 & \frac{1}{6} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

このとき、2 番目の行に注目した時

$$\sum_{j=1}^{7} \mathbf{T}_{2,j} = \frac{1}{4} = d_2$$

となる。

また、6 番目の列に注目した時

$$\sum_{i=1}^{7} \mathbf{T}_{i,6} = \frac{1}{3} = d'_6$$

となる。このときの、変換を行うときのイメージは以下のようになる。

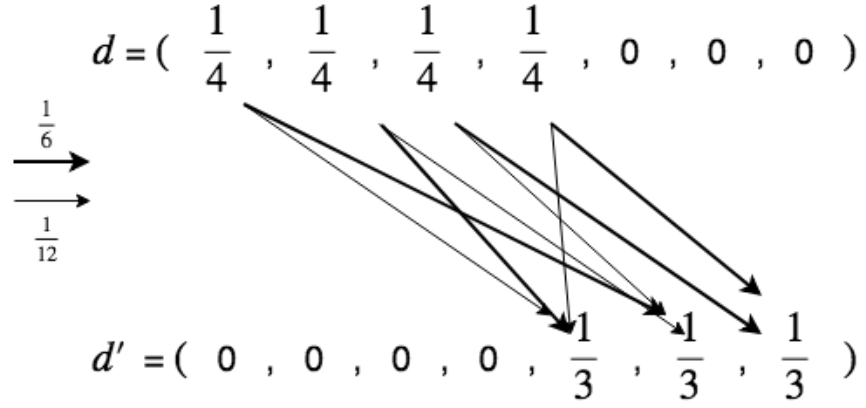

図 3.1　行列 $\mathbf{T}$ を用いた d から d' への変換

変換行列 $\mathbf{T}$ の各成分に対応した語同士の距離をかけたものの総和を求める。このとき、距離の総和が最小となる $\mathbf{T}$ を定めたとき、その距離の総和が文と文の距離となる。

$$\min_{\mathbf{T}\geq 0} \sum_{i,j=1}^{n} \mathbf{T}_{i,j} c(i,j)$$

$$\textbf{subject to} : \sum_{j=1}^{n} \mathbf{T}_{i,j} = d_i \forall i \in \{1,...,n\}$$

$$\sum_{i=1}^{n} \mathbf{T}_{i,j} = d'_j \forall j \in \{1,...,n\}$$

この最適化問題は Earth Mover's Distance (EMD)[2] を用いて求めることが出来る特殊な形式になっている。

第 4 章

WMD の使用方法

WMD は Python のライブラリの一つである gensim 内に実装されており、それを使用する。
まず gensim のインストールは以下のコマンドを使用する。

$pip install –upgrade gensim

gensim 内の WMD の使用手順として、あらかじめ Word2Vec による分散表現を利用しモデルを構築するか、用意されたモデル読み込み使用する。

WMD は gensim.models.Word2Vec 内の関数 wmdistance として実装されており、計算を行う 2 つの文を引数として渡すことで計算を行う。

例として以下のように使用する。

wmdistance(”The President greets the press in Chicago”,”Obama speaks in Illinois”)

第 5 章

提案手法

5.1 KMeans によるクラスタリング

成分に文 j における語 i の TFIDF 値を $d_{i,j}$ として設定する。

$$d_{i,j} = tf(i,j) \cdot \log\left(\frac{N}{df(i)}\right)$$

ここで N は文集合、$tf(i,j)$ は文 j での語 i の出現頻度、$df(i)$ は語 i の出現する文の数である。

TFIDF 値を基に検索対象の全文 N に対し KMeans によるクラスタリングを行う。

$$\sum_{x_j \in N} \min_{i \in K} \|x_j - c_i\|^2$$

x_j は文 j , K クラスタ数, c_i はクラスタ i の中心である。この式を最小化するクラスタの中心を求める。各文に対し最も近いクラスタを割り当て、各クラスタの中心を計算し更新する作業を繰り返し、求まったクラスタの中心を利用しクラスタリングを行う。

$$c_i = \frac{1}{\mid C_i \mid} \sum_{x_j \in C_i} x_j$$

C_i はクラスタ i の文集合、$\mid C_i \mid$ はデータ数である。

5.2 クエリと重心の距離比較

クエリ q_j と各クラスタの中心 c_i との距離の比較を行い、最小の距離となるクラスタをクエリに割り当てる。

$$\min_{i \in K} \|q_j - c_i\|^2$$

5.3 クラスタ内での距離計算

クラスタ内の文 x_i とクエリ q_j の距離計算を WMD により行い、クラスタ内での距離が最小となるものを求める。

第 6 章

実験

WMD を利用するには単語の分散表現が必要である。ここでは Wikipedia のダンプデータを基に 200 次元のものを生成し用いた。Wikipedia のダンプデータはサイト[*1]で公開されている jawiki-latest-pages-articles.xml.bz2 である。また実験で利用する検索対象の文やクエリの文も上記データから取りだした。具体的には上記データを展開し、全角記号のみからなる文を取り出し、そこからランダムに検索対象の 10,000 文とクエリとなる 5 文抽出した。クエリとなる 5 文を表 6.1 に示す。

クエリ 1	なお、この日本武尊像には、「ハトが寄り付かない」という逸話がある。
クエリ 2	金沢城から見て巽（東南）の方向にあること、京都の鷹司家が辰巳殿と呼ばれていたことから当時は巽御殿と呼ばれていた。
クエリ 3	石川県立美術館別館・石川県文化財修復工房は、もともと石川県庁出羽町庁舎内にあった文化財修復工房を移転リニューアルしたもの。
クエリ 4	表具・漆芸品の各修復室と展示スペースなどからなり、展示スペースでは、実際に文化財を修復している現場を見学することが出来る。
クエリ 5	ヒロイン・アクション劇画の第一人者にしてパイオニア的存在で、多くの作品が映像化されているが、アニメ化された作品は一本もない。

表 6.1　各クエリの内容

クラスタリングには scikit-learn の KMeans を用いる。クラスタ数を 100 とし、検索対象のクラスタリングを行う。得られた検索対象のクラスタに対し、クエリが属するクラスタを求め、クラスタ内で WMD による計算を行う。

WMD による計算は gensim を使用する。クエリの 5 文を用い、検索対象に対して類似文検索を行う。このとき、クラスタリングを用いないで全文に対しての検索とクラスタリングを用いてクラスタないでの検索を行い、1 文あたり検索時間と出力結果を求める。表 6.2 には、クエリの 1 文あたりの類似文の検索時間の平均を求めたものである。

[*1] https://dumps.wikimedia.org/jawiki/latest/

	平均検索時間 (秒)
提案手法	90
全体比較	500

表 6.2　検索時間

検索時間は表 6.2 から提案手法の方が短く、検索の高速化が実現出来ている。出力結果の精度について、クラスタリングを用いないで得られた出力結果を正解とし、提案手法から得られた出力結果に対し正解の結果から上位何番目に位置するかを表 6.3 に示した。

	1 位	2 位	3 位	4 位	5 位	平均順位
クエリ 1	1	5	7	10	14	7.4
クエリ 2	2	12	14	27	41	19.2
クエリ 3	5	8	9	13	33	13.6
クエリ 4	3	21	22	42	43	26.2
クエリ 5	1	13	38	57	71	36.0

表 6.3　正解から提案手法の結果に対応する順位付け

平均順位は 3 に近い値ほど良い結果となる。クエリ 1 の出力結果は正解の出力結果に近い値を表示していることがわかる。またどのクエリに対しても第 1 順位で検索できる文は、全体を WMD で検索した場合の上位に位置する文であり、1 つだけを検索したい場合には有用と考えられる。

出力結果の例として、クエリ 5 の文「ヒロイン・アクション劇画の第一人者にしてパイオニア的存在で、多くの作品が映像化されているが、アニメ化された作品は一本もない。」が与えられた時、それに対するクラスタリング用いない結果の 10 文とクラスタリングを用いた結果の 5 文を表 6.4 と表 6.5 に示す。類似文の検索結果としては大きな差はないように感じる。

順位	文
1	「キャストについてはデュナン役は前作から引き続いて小林愛が担当するが、それ以外のキャラクターについては大幅なキャストの変更が加えられている。」
2	「カラー原稿は、少女誌で活動していた頃はカラーインクやエアスプレーを使用していたが、後にコピックやアクリルガッシュに移行。」
3	「ローグライクゲーム形式の「森」や「塔」でアイテムやゴールドを入手する、鍛えたモンスターを市場で売買する、「闘技場」で他のユーザのモンスターと対決させるといったこともできる。」
4	「代表作に『超少女明日香』シリーズ、『忍者飛翔』、『怪盗アマリリス』、『ピグマリオ』、『スケバン刑事』、『少女鮫』など。」
5	「セットは一つの終端の可能なドットの数の中で、およびピースの組み合わせの数によってさまざまである。」
6	「現在ではこのようなチェンバロは、歴史的なチェンバロや、それらに準じて製作されたチェンバロとは区別して、モダン・チェンバロと呼ばれる。」
7	「しかし、ファンの中にはカーペンターズのレコードにエレキギターのソロが入ることに不満を抱く者も少なくなく、嫌がらせの手紙を送りつける者さえいたという。」
8	「原作コミックス版にて「猫実臨海水族館」として描かれている建物は、学校法人桐蔭学園の「鵜川メモリアルホール」と呼ばれる多目的ホールの外観を模している。」
9	「中国版限定のアイテムもあり、日本版に逆輸入される現象もおきている。」
10	「特に日本市場ではユーザーの意思をゲームに反映させる手段にコマンド選択式インターフェイスを採用しているものが多い。」

表 6.4　正解となる文とその順位

順位	文
1	「キャストについてはデュナン役は前作から引き続いて小林愛が担当するが、それ以外のキャラクターについては大幅なキャストの変更が加えられている。」
2	「かつては地上アナログ放送専用のチューナーと呼ばれる単体商品も存在した。」
3	「この映画の放映後数週間はレコード屋からカーペンターズの在庫がなくなったほどである。」
4	「よって、個人向けに、小さなフィルムを使うことでフィルム代や現像代といった感材費をおさえた。」
5	「なお、パーレットの古い分類ではクライミングゲーム・ゴーイングアウトゲームは「シェディングゲーム」、ラミーは「コレクティングゲーム」にまとめていた。」

表 6.5　提案手法による検索結果

第 7 章

考察

検索時間に関して、クラスタリングを行い、クラスタ内でのみ WMD による比較を行うようにしたため計算回数を減らし高速化を実現出来た。しかし、これはクラスタ内の文の数に依存する。クラスタ数を増やせば、一つあたりのクラスタに含まれる文の数は減少しさらに高速化することが可能になるが、出力結果の精度は落ちることが考えられる。類似文検索の出力結果に対して、クラスタリングを用いた場合と用いない場合とで差が生じていることが分かった。これは、検索対象の文に対して bag-of-words(BOW) による TFIDF 値をもとに KMeans によるクラスタリングを行ったが、BOW は個々の単語間の距離を考慮していないため、クラスタリングに影響がでたと考えられる。このことに対し、検索対象内の文同士に対して WMD による計算し、算出された結果を基に KMeans によるクラスタリングを行えば類似文検索の出力結果が向上すると考える。また、クエリがクラスタの中心から離れ、クラスタの境界付近に位置していた場合が考えられる。クラスタの境界付近にクエリが位置していた場合、クエリに近い文が異なるクラスタに属している場合、計算が行われないため、正解との出力結果が大きく異なる。クラスタ数を調整することにより出力結果の向上は見込める。

第 8 章

おわりに

本論文では WMD を用いた類似文検索における高速化の手法を提示した。検索対象の文に対し、TFIDF 値を求め、KMeans によるクラスタリングに利用する。クラスタリング結果を基に、クエリが属するクラスタを求め、クラスタ内の検索対象の文に対して WMD による計算を行う。実験では、クラスタリングを用いた場合と用いない場合とで、検索時間と出力結果から比較を行った。検索時間を短縮することには効果的であるが、出力結果に関しては改善の余地が見られた。今後 KMeans によるクラスタリングを行う際、文同士の比較の際、BOW による　TFIDF 値を基に計算を行うのではなく WMD に基づく計算による比較を行えば出力結果の向上が考えられるため、今後はこの方向で研究を進めていきたい。

謝辞

本研究を進めるにあたり、多くのご指導、ご協力を頂いた指導教員の新納浩幸教授に感謝致します。また、日常の議論を通じて多くの知識や示唆を頂いた新納研究室の皆様に感謝します。

参考文献

[1] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In *International Conference on Machine Learning*, pp. 957–966, 2015.

[2] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, Vol. 40, No. 2, pp. 99–121, 2000.

[3] Christoph Zauner. Implementation and benchmarking of perceptual image hash functions. 2010.

[4] 岡崎直観, 辻井潤一. 集合間類似度に対する簡潔かつ高速な類似文字列検索アルゴリズム. 自然言語処理, Vol. 18, No. 2, pp. 89–117, 2011.

[5] 南濱篤, 新納浩幸, 古宮嘉那子. Word mover's distance に基づく類似文検索におけるクラスタリングによる高速化. 言語処理学会第 25 回年次大会, to appear,2019.

本論文で用いたプログラム

Listing 1　wmd.py

```
import numpy as np
import sys , pickle
import gensim
import time
import multiprocessing

#  Loding
model = gensim.models.KeyedVectors.load_word2vec_format('./jatext8.bin', binary=False)

def save_bin(load_file,save_file):
    sentences = gensim.models.word2vec.Text8Corpus(load_file)
    mod = gensim.models.word2vec.Word2Vec(sentences, size=200)
    mod.save_word2vec_format(save_file)

# Distance text to text
def wmd(t1,t2):
    distance = model.wmdistance(t1,t2)
    return distance

# one milion line
def load_txt(path):
    with open(path) as f:
        lines = [s.strip() for s in f.readlines()]
    return lines

#
def load_clus(path):
    with open(path) as f:
        dic = dict((map(int,(s.strip().split('␣'))) for s in f.readlines()))
    return dic


#
def search_wmd(txt,txt_list):
    count = 0
    dis = {}
    starttime = time.time()
    for li in txt_list:
        dis[count] = wmd(txt,li)
        count += 1
    endtime = time.time()
    intarval = endtime-starttime
    #max_key = max(dis,key=dis.get)
    max_keys = sorted(dis.items(),key = lambda x:x[1])[:100]
    return [intarval,max_keys]


def cluster_wmd(txt,txt_list,clus_list):
    dis = {}
    starttime = time.time()
    for key in clus_list:
        dis[key] = wmd(txt,txt_list[key])
    endtime = time.time()
    intarval = endtime - starttime
    #max_key = max(dis,key=dis.get)
    max_keys = sorted(dis.items(),key = lambda x:x[1])[:10]
    return [intarval,max_keys]
```

```
58
59 def mul_wmd(tuple_data):
60     return tuple_data[0](tuple_data[1],tuple_data[2])
61
62 def mul_cluster(tuple_data):
63     return tuple_data[0](tuple_data[1],tuple_data[2],tuple_data[3])
64
65 milion_lines = load_txt('wiki_00_line.txt')
66 search_txt = load_txt("search_text.txt")
67 clus_data = load_clus('./data/clus_data3.txt')
68 clus_search = load_clus('./data/clus_search3.txt')
69 val = clus_search[0]
70 keys = [int(k) for k, v in clus_data.items() if v == val]
71
72 #start
73 result = []
74 #starttime = time.time()
75 #result = search_wmd(search_txt[0],milion_lines)
76
77 #t_data = [(search_wmd,search_txt[i],milion_lines) for i in range(100)]
78 t_data = [(cluster_wmd,search_txt[i],milion_lines,keys) for i in range(100)]
79 with multiprocessing.Pool(5) as p:
80     multi = p.map(mul_cluster,t_data)
81
82 #end
83 #endtime = time.time()
84 #interval = endtime - starttime
85 #print(str(interval) + "秒")
86
87 #print(result)
88 print('====␣␣␣all␣search␣␣␣====')
89 #for i in multi:
90 #    print(i)
91 print('====␣cluster␣search␣====')
92 for i in multi:
93     print(i)
94 with open('list_clus3.bin','wb') as f:
95     pickle.dump(multi,f)
```

Listing 2　make_txt.py

```
1  import unicodedata
2  import re
3  import os
4
5  def judge(text):
6      count = 0
7      for s in text:
8          if unicodedata.east_asian_width(s) in 'FW':
9              count += 1
10         else:
11             return 0
12     return count
13
14 def load(file_name):
15     with open(file_name) as f:
16         lines = [s.strip() for s in f.readlines()]
17         lines = [re.findall(r'。?.+?(?<=[])',s) for s in lines]
18         lines = [i for j in lines for i in j]
19         lines = [line for line in lines if 30 <= judge(line) <= 100]
20         return lines
21
22 def save(file_name,text):
23     with open(file_name,mode = 'w') as f:
24         f.write('\n'.join(text))
25
26 def file_list(path):
27     dir = os.listdir(path)
28     dir = [path +'/'+ x for x in dir]
29     return sorted(dir)
30
31
32 save_file = 'wiki_00_li.txt'
33 file_path = '../wikipwdia/AA'
```

```
34
35 text_list = []
36 load_file = file_list(file_path)
37 for x in load_file:
38     text_list.extend(load(x))
39     if len(text_list) > 10010:
40         break
41
42 save(save_file,text_list)
```