

工学実験実習IV レポート

自然言語処理

実験日1 2020年6月03日 1～2コマ目
実験日2 2020年6月10日 1～2コマ目
実験日3 2020年6月17日 1～2コマ目
実験日4 2020年6月24日 1～2コマ目
実験日5 2020年7月08日 1～2コマ目
提出日 令和2年7月14日

組番号 408
学籍番号 17406

氏名 金澤雄大

1 目的

自然言語処理における言語処理の基礎的な知識および開発環境を学ぶことを目的とする。また自然言語処理の結果を分析し、文章の特徴を抽出するように改良を行うことを目的とする。

2 理論・概要

本章では次に示す理論、概要について述べる。

1. 形態素解析
2. Term Frequency
3. Inverse Document Frequency
4. Tf-Idf

2.1 形態素解析

自然言語とは人間が普段話す「日本語」や「英語」を代表とする言語である。本実験では「日本語」を処理する。例として「技術者がこの先生きのこるためにすべきことはなにか」という文について考える。人間がこの文の意味を理解するためには、「技術者が、この先、生きのこるために、すべきことはなにか」というように「、」で文を区切って理解するのではないだろうか。しかし逐次的に文章をたどっていくと「技術者が、この先生、きのこるためにすべきことはなにか」というように解釈されることもあり得る。この解釈を一意にするため、文法の規範や辞書をもとにして、文章を形態素(単語ごとの区切り)に分割するツールが形態素解析器である。本実験では形態素解析器として MeCab[1] を用いる。MeCab とは京都大学情報学研究科ー日本電信電話株式会社コミュニケーション科学基礎研究所 共同研究ユニットプロジェクトを通じて開発されたオープンソースの形態素解析エンジンである。例えば「技術者がこの先生きのこるためにすべきことはなにか」という文章を MeCab に入力するとリスト 1 のような出力が得られる。リスト 1 から「技術者が、この先、生きのこるために、すべきことはなにか」というように文章を分割していることが読み取れる。また、リスト 1 の EOS(End Of Statement) は文章の終わりのことである。

リスト 1: MeCab の出力の例

1	技術	名詞, 一般,*,*,*,*, 技術, ギジユツ, ギジユツ
2	者	名詞, 接尾, 一般,*,*,*,*, 者, シャ, シャ
3	が	助詞, 格助詞, 一般,*,*,*,*, が, ガ, ガ
4	この	連体詞,*,*,*,*,*, この, コノ, コノ
5	先	名詞, 一般,*,*,*,*, 先, サキ, サキ
6	生き	動詞, 自立,*,*,*, 一段, 連用形, 生きる, イキ, イキ
7	のこる	動詞, 自立,*,*,*, 五段・ラ行, 基本形, のこる, ノコル, ノコル
8	ため	名詞, 非自立, 副詞可能,*,*,*,*, ため, タメ, タメ
9	に	助詞, 格助詞, 一般,*,*,*,*, に, ニ, ニ
10	す	動詞, 自立,*,*,*, サ変・スル, 文語基本形, する, ス, ス
11	べき	助動詞,*,*,*,*, 文語・ベシ, 体言接続, ベシ, ベキ, ベキ
12	こと	名詞, 非自立, 一般,*,*,*,*, こと, コト, コト
13	は	助詞, 係助詞,*,*,*,*, は, ハ, ハ
14	なにか	副詞, 助詞類接続,*,*,*,*, なにか, ナニカ, ナニカ
15	EOS	

2.2 Term Frequency

形態素解析の結果を用いて、文章の特徴を表すような単語を探したい。文章の特徴を表すような単語がどのようなものであるか考えると、文書に頻繁に出現する語が文章を特徴づける語であると考えられる。この

考えを実装したものが Term Frequency(以下 tf) という重み付けである. ある文章 d に出てくる一つの単語 t について, tf による重み $W_{tf_t}^d$ は式 (1) で表せる.

$$W_{tf_t}^d = \frac{tf(t, d)}{\sum_{s \in d} tf(s, d)} \quad (1)$$

式 (1) において $tf(t, d)$ は単語 t が文書 d に出現する回数, $\sum_{s \in d} tf(s, d)$ は文書 d に登場する全単語数である. 何故全単語数で割るのかについて説明する. 例として, 次の 2 つの文書について考える.

- 全単語数 200 で, ある単語 t が 10 回でてくる文書 A.
- 全単語数 20000 で, ある単語 t が 10 回でてくる文書 B.

どちらもある単語 t が 10 回でてくるため $tf(t, d) = 10$ である. しかし全単語数 200 である A の方が単語 t の登場頻度が高いから, 文書 A, B における単語 t の出現頻度が同じであるという判断は正しくない. このため $tf(t, d)$ を全単語数で割る操作を行う. この操作を正規化という.

2.3 Inverse Document Frequency

Term Frequency は, 文書に頻繁に出現する語が文章を特徴づける語である, という考えから生まれた重みづけの方法であった. しかし文章にまれに登場する語のほうが文章を特徴づける語なのではないか, と考えることもできる. この考えを実装したものが Inverse Document Frequency(以下 idf) である. 文章にまれに登場するかどうかは他の文章と比較して決定する必要があるから, idf は複数文書間の重みづけになる. 例として, 文書 1 万件の中で, 「自動車」が含まれている文書が 1000 件ある場合と, 「スペースシャトル」が含まれている文書が 10 件ある場合では「スペースシャトル」を含む文書のほうが膨大にある文書の中から絞り込みができていいると考えられる.

文書の総数 N , ある単語 t を含んでいる文書の数を $df(t)$ とする. このとき単語 t の idf 値 $idf(t)$ は式 (2) で表される.

$$idf(t) = \log_{10} \frac{N}{df(t)} + 1 \quad (2)$$

単語 t を含んでいる文書の数 $df(t)$ が大きくなると $\frac{N}{df(t)}$ は小さくなる. 図 1 は $N = 1, 10, 100$ のときの $\frac{N}{df(t)}$ のグラフである.

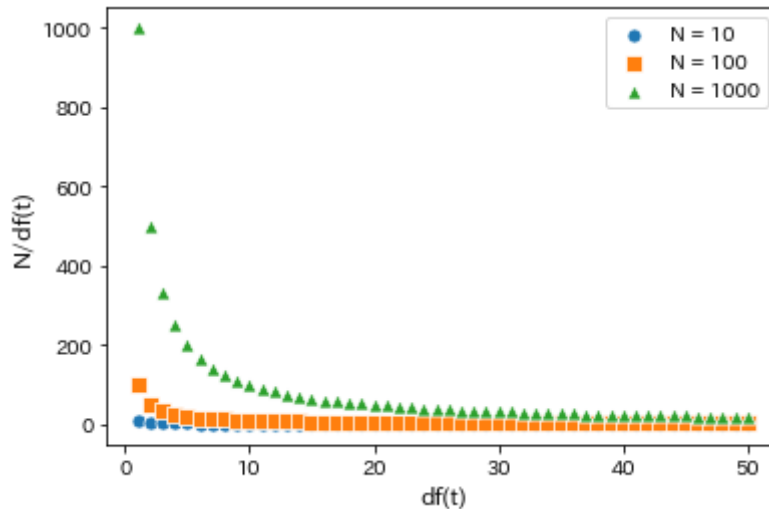


図 1: $\frac{N}{df(t)}$ vs. $df(t)$

図 1 から文書数 1000 のとき, $\frac{N}{df(t)}$ は 0 から 1000 という広い値を取ることがわかる. 例として $\frac{N}{df(t)} = 1000$ の単語 t_1 , $\frac{N}{df(t)} = 10$ の単語 t_2 , $\frac{N}{df(t)} = 1$ の単語 t_3 の 3 単語があったとする. 3 単語を比較したときに単語 t_1

は単語 t_2 よりどれぐらい文書の特徴付ける単語なのか, 単語 t_2 は単語 t_3 よりどれぐらい文書の特徴付ける単語なのかが分かりにくい. このため $\frac{N}{df(t)}$ に対数変換を行って $idf(t)$ が右に裾の長い分布になることを抑える必要がある. 図 2 は $N = 1, 10, 100$ のときの $idf(t)$ のグラフである. 図 2 では対数変換によって重みづけ $idf(t)$ が滑らかになっており, 単語の $idf(t)$ を相対的に比較したときに, どちらがより文書の特徴を表す語であるかがわかりやすい.

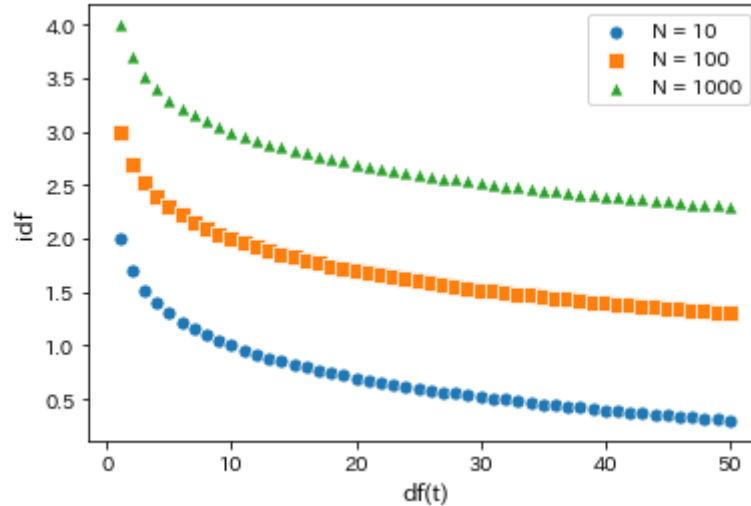


図 2: $idf(t)$ vs. $df(t)$

2.4 Tf-Idf

文書に出現する頻度の高さを特徴として表す重みづけとして Term Frequency を与え, 文書群での出現頻度が低い, 稀である特徴を表す重みづけとして Inverse Document Frequency を与えた. Term Frequency は文書から導出した値で, Inverse Document Frequency は文書群から導出した値であるので, 互いの値を組み合わせて, 文書群に含まれる文書の特徴の重みづけとして Term Frequency-Inverse Document Frequency (Tf-Idf) を与える. これによって文書中の出現品後が少ない単語でも, 他の文書に含まれていない場合は, この文書の特徴づける重みとして, 有効に作用すると予想される.

式 3 は文書 d に登場する単語 t における Tf-Idf の重み $W_{tf-idf_t}^d$ を表している.

$$W_{tf-idf_t}^d = \frac{tf(t, d)}{\sum_{s \in d} tf(s, d)} \times \left(\log_{10} \frac{N}{df(t)} + 1 \right) \quad (3)$$

3 実験方法

本章では開発環境および実験手順について述べる.

3.1 開発環境

開発環境を表 1 に示す. 授業中の資料では文字コードは SJIS を用いることになっているが他環境と競合するため UTF-8 を用いる.

表 1: 開発環境

CPU	Intel Core i7-6500 2.50Ghz
メモリ	16.0GB DDR3
OS	Windows 10 Home
統合開発環境	Eclipse Java EE IDE for Web Developers. Version: 2018-09 (4.9.0)
開発言語	Java 08
文字コード	UTF-8

3.2 実験手順

本実験では, 001.txt から 100.txt までの 100 文書を用いて実験を行う. 実験や解析に用いる文書全体をコーパスと呼ぶ. 本実験ではコーパスを data-utf8 ディレクトリ下に置いている. 本実験で扱う文書の例として, 付録のリスト 25 に 001.txt の全文を示す. リスト 25 から文書ファイルの中身は文章のみで構成されていることがわかる. 他の文書ファイルも同様の形式になっている.

次に示す手順で実験を行う. 各重みを計算し, 計算結果を txt ファイルに書き出すプログラムを実装する. ただし, 本レポートでは MeCab の環境構築, および Eclipse の使用方法については説明しない.

1. 形態素解析と Term Frequency の実装
2. Inverse Document Frequency の実装
3. Term Frequency-Inverse Document Frequency の実装
4. 実行結果の分析と改良

4 Term Frequency の実装と実行結果

本章では Term Frequency の実装と実行結果について述べる.

4.1 プログラムの説明

Term Frequency の実装は次の 6 つのクラスを作成して行う. 本節ではこの 6 つのクラスの説明について述べる.

1. Word クラス
2. WordCount クラス
3. TfCount クラス
4. WordCompare クラス
5. TermFrequency クラス
6. NaturalLanguageProcessing クラス (メイン)

4.1.1 Word クラス

リスト 1 に示したように,MeCab の実行結果は分割した品詞の種類, 活用形を代表とする多数の情報を出力する. 表 2 は Mecab の実行時に出力される情報と「渡し」のときの例である.

表 2: Mecab の出力情報

素性情報	「渡し」の例
表層系	渡し
品詞	動詞
品詞細分類 1	自立
品詞細分類 2	*
品詞細分類 3	*
活用型	五段・サ行
活用形	連用形
基本形	渡す
読み	ワタシ
発音	ワタシ

MeCab から出力される素性情報を格納するクラスとして Word クラスを作成する. リスト 2 に Word クラスのソースコードを示す. ただし, フィールド変数のセッターおよびゲッターは省略する. Word クラスはフィールドに private 変数として表 2 の出力結果を持つ. また, メソッドとしてセッター, およびゲッター, および 2 つの変数の一致/不一致を boolean で出力する equals メソッドを設ける. フィールド変数を private として, セッターおよびゲッターを設けるのは外部からの入力で変数を不正な値にされることを防ぐためである. オブジェクト指向において, クラス内のフィールドおよびメソッドを外部から保護することをカプセル化という.

リスト 2: Word クラスのソースコード

```
1 package nlp;
2
3 class Word
4 {
5
6     // セッターおよびゲッターは省略
7
8     public boolean equals(Object obj)
9     {
10         //オブジェクトが null の場合, 不一致
11         if(obj == null)
12         {
13             return false;
14         }
15         //オブジェクトの型が異なる場合, 不一致
16         if(!(obj instanceof Word))
17         {
18             return false;
19         }
20         //オブジェクトの全てのフィールドが一致している場合, 一致
21         if(
22             (((Word)obj).getHyouSoukei() == null && this.getHyouSoukei() == null)
23             ||((Word)obj).getHyouSoukei().equals(this.getHyouSoukei()))
24             && ( (((Word)obj).getHinshi() == null && this.getHinshi() == null)
25                 ||((Word)obj).getHinshi().equals(this.getHinshi()))
26             && ( (((Word)obj).getHinshi1() == null && this.getHinshi1() == null)
27                 ||((Word)obj).getHinshi1().equals(this.getHinshi1()))
28             && ( (((Word)obj).getHinshi2() == null && this.getHinshi2() == null)
29                 ||((Word)obj).getHinshi2().equals(this.getHinshi2()))
30             && ( (((Word)obj).getHinshi3() == null && this.getHinshi3() == null)
```

```

31         ||((Word)obj).getHinshi3().equals(this.getHinshi3()))
32         && ( (((Word)obj).getKatsuyoKata() == null && this.getKatsuyoKata() == null)
33         ||((Word)obj).getKatsuyoKata().equals(this.getKatsuyoKata()))
34         && ( (((Word)obj).getKatsuyoKei() == null && this.getKatsuyoKei() == null)
35         ||((Word)obj).getKatsuyoKei().equals(this.getKatsuyoKei()))
36         && ( (((Word)obj).getGenkei() == null && this.getGenkei() == null)
37         ||((Word)obj).getGenkei().equals(this.getGenkei()))
38         && ( (((Word)obj).getYomi() == null && this.getYomi() == null)
39         || ((Word)obj).getYomi().equals(this.getYomi()))
40         && ( (((Word)obj).getHatsuon() == null && this.getHatsuon() == null)
41         || ((Word)obj).getHatsuon().equals(this.getHatsuon()))
42     )
43     {
44         return true;
45     }
46     //そのほかは不一致
47     return false;
48 }
49
50 private String hyousoukei = null; //表層形
51 private String hinshi = null; //品詞
52 private String hinshi1 = null; //品詞細分類 1
53 private String hinshi2 = null; //品詞細分類 2
54 private String hinshi3 = null; //品詞細分類 3
55 private String katsuyoKata = null; //活用形
56 private String katsuyoKei = null; //活用形
57 private String genkei = null; //原形
58 private String yomi = null; //読み
59 private String hatsuon = null; //発音
60
61 }

```

4.1.2 WordCount クラス

tf 値を計算する際に単語の出現回数の情報が必要であるから、これを記憶する WordCount クラスを作成する。WordCount クラスはフィールドに private 変数として Word 型の変数 word と、単語の出現回数を保持する Integer 型の変数 count を設ける。また、メソッドとしてコンストラクタ、およびフィールド変数のセッター、およびゲッターを設ける。WordCount クラスのソースコードは省略する。

4.1.3 TfCount クラス

tf の計算結果を保持するクラスとして、WordCount クラスを継承して、TfCount クラスを作成する。TfCount クラスのフィールドには tf の計算結果を保持する変数として、private Double 型の変数 tf を設ける。またメソッドとして、コンストラクタ、および変数 tf のセッターおよびゲッターを設ける。TfCount クラスのソースコードは省略する。

4.1.4 WordCompare クラス

tf の計算結果を出力するときに、tf 値で結果をソートしてファイル出力を行う。このため、2 つの WordCount 型変数の出現回数を大小比較するクラスとして WordCompare クラスを作成する。リスト 3 に WordCompare クラスのソースコードを示す。WordCompare クラスは Comparator インターフェースを実装している。compare 関数の仕様は Java API ドキュメント [2] にあるように 2 つのオブジェクト o1, o2 を引数として、最初の引数 o1 が 2 番目の引数 o2 より小さい場合は負の整数 (-1)、両方が等しい場合は 0、最初の引数 o1 が 2 番目の引数 o2 より大きい場合は正の整数 (1) を返す。

リスト 3: WordCompare クラスのソースコード

```

1 package nlp;
2

```

```

3 import java.util.Comparator;
4
5 class WordCompare implements Comparator<WordCount>
6 {
7     @Override
8     public int compare(WordCount wc1, WordCount wc2)
9     {
10         if(wc1.getCount() < wc2.getCount())
11             return 1;
12         if(wc1.getCount() == wc2.getCount())
13             return 0;
14         if(wc1.getCount() > wc2.getCount())
15             return -1;
16         return 0;
17     }
18 }

```

4.1.5 TermFrequency クラス

1つのファイルについて形態素解析およびtfの計算を行い、ファイルに書き込む関数として TermFrequency クラスを実装する。リスト4に TermFrequency クラスのソースコードを示す。TermFrequency クラスはフィールドに TfCount 型のリスト list を持つ。また tf メソッドを持つ。tf メソッドの引数、および戻り値、および処理内容を表3に示す。

リスト 4: TermFrequency クラスのソースコード

```

1 package nlp;
2
3 import java.io.BufferedReader;
4 import java.io.FileWriter;
5 import java.io.IOException;
6 import java.io.InputStreamReader;
7 import java.util.ArrayList;
8
9 public class TermFrequency {
10
11     ArrayList<TfCount> list = new ArrayList<TfCount>();
12
13     public void tf(String inputFilename, String outputFilename){
14
15         //String inputFilename = "data-utf8\001.txt";
16         String[] command = {"cmd.exe", "/C", "mecab", inputFilename };
17
18         try {
19             Process ps = Runtime.getRuntime().exec(command);
20             BufferedReader bReader_i = new BufferedReader(
21                 new InputStreamReader(ps.getInputStream(), "UTF-8"));
22             String targetLine;
23
24             while (true) {
25                 targetLine = bReader_i.readLine();
26
27                 if (targetLine == null) {
28                     break;
29                 }
30                 else if (targetLine.equals("EOS")) {
31                     continue;
32                 }
33                 else {
34                     Word wo = new Word();
35                     String targetArray[] = targetLine.split("[\t|,]");
36
37                     if (targetArray.length >= 1)
38                         wo.setHyoujoukei(targetArray[0]);
39                     if (targetArray.length >= 2)
40                         wo.setHinshi(targetArray[1]);
41                     if (targetArray.length >= 3)
42                         wo.setHinshi1(targetArray[2]);
43                     if (targetArray.length >= 4)

```



```

44         wo.setHinshi2(targetArray[3]);
45     if (targetArray.length >= 5)
46         wo.setHinshi3(targetArray[4]);
47     if (targetArray.length >= 6)
48         wo.setKatsuyoKata(targetArray[5]);
49     if (targetArray.length >= 7)
50         wo.setKatsuyoKei(targetArray[6]);
51     if (targetArray.length >= 8)
52         wo.setGenkei(targetArray[7]);
53     if (targetArray.length >= 9)
54         wo.setYomi(targetArray[8]);
55     if (targetArray.length >= 10)
56         wo.setHatsuon(targetArray[9]);
57
58     //リストの中に同じ語のエントリがあるか調べて、既にエントリがあれば、カウントアップする
59     int i;
60     for (i = 0; i < list.size(); i++) {
61         if (list.get(i).getWord().equals(wo)) {
62             list.get(i).setCount(list.get(i).getCount() + 1);
63             break;
64         }
65     }
66     //リストにエントリが無かったときは、新しい語としてリストに追加する
67     if (i == list.size()) {
68         list.add(new TfCount(wo, Integer.valueOf(1)));
69     }
70
71     //System.out.println(targetLine);
72 }
73 }
74
75 }
76 catch (IOException e) {
77     e.printStackTrace();
78 }
79
80 //今のリストのエントリをすべて表示する
81 int sum = 0;
82 for (int i = 0; i < list.size(); i++){
83     //System.out.println(list.get(i).getWord()
84     .getHyouyoukei()+ ":" + list.get(i).getCount());
85     sum += list.get(i).getCount();
86 }
87
88 list.sort(new WordCompare());
89
90 try {
91     //String outputFilename = "data-utf8\\001tf.txt";
92     FileWriter fw = new FileWriter(outputFilename);
93     //System.out.println(outputFilename);
94     for (int i = 0; i < list.size(); i++){
95         //System.out.println(fileal.get(i).word.getHyouyoukei() +
96         ":" + fileal.get(i).getCount());
97         list.get(i).setTf((double) list.get(i).getCount() / (double) sum);
98
99         fw.write(list.get(i).getWord().getHyouyoukei() + "\t"
100             + list.get(i).getWord().getHinshi() + "\t"
101             + list.get(i).getWord().getHinshi1() + "\t"
102             + list.get(i).getCount() + "\t"
103             + String.format("%.10f", list.get(i).getTf()) + "\n");
104     }
105     fw.close();
106 }
107 catch (IOException ex) {
108     ex.printStackTrace();
109 }
110
111 }
112 }

```

表 3: tf メソッドの概要

引数	入力ファイル名, 出力ファイル名
返り値	void
処理内容	入力ファイルの形態素解析および tf の計算を行い, 出力ファイルに書き出す.

tf メソッドの処理内容の詳細は次の通りである.

1. 入力ファイルについて形態素解析を行う.
2. 形態素解析の結果を 1 行づつ読み込み, 形態素解析の結果をリスト list に書き込み, 単語の出現回数をカウントする.
3. 形態素解析の結果が EOS に辿り着いたとき, リストへの書き込みと単語の出現回数カウントを終了する.
4. リスト list を単語の出現回数で降順にソートする.
5. 必要であれば, リストに含まれている単語を標準出力する.
6. 出力ファイルにリスト list の内容 (表層系, 品詞, 出現回数,tf 値) を書き出す. 区切りはタブとする.

4.1.6 NaturalLanguageProcessing クラス (メイン)

メインクラスとして NaturalLanguageProcessing クラスを作成する. メインクラスの役割は 100 件の文書ファイルについて TermFrequency クラスをインスタンス化し,tf メソッドを実行することである. リスト 5 に NaturalLanguageProcessing クラスのソースコードを示す. main 関数では長さ 100 の tf クラスの配列を用意し,for 文を用いて各文書の解析情報を格納する tf 配列の要素をインスタンス化し,tf メソッドの実行をしている.

リスト 5: NaturalLanguageProcessing クラスのソースコード

```

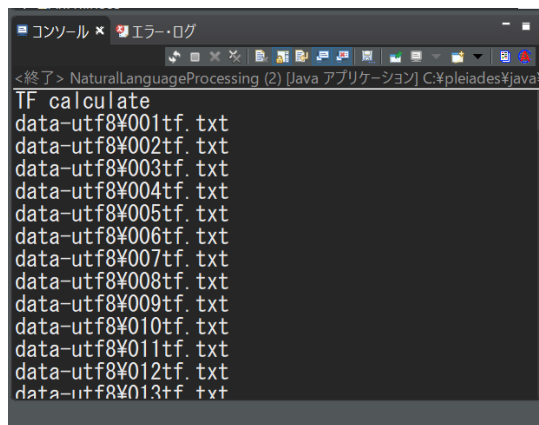
1 package nlp;
2 public class NaturalLanguageProcessing
3 {
4     static public void main(String args[])
5     {
6         System.out.println("TF calculate");
7         TermFrequency[] tf= new TermFrequency[100];
8         for(int i=1; i <= 100;i++)
9         {
10            tf[i-1] = new TermFrequency();
11            String inputFileName = "data-utf8\\" + String.format("%03d", i) + ".txt";
12            String outputFileName = "data-utf8\\" + String.format("%03d", i) + "tf.txt";
13            //System.out.println(inputFileName);
14            System.out.println(outputFileName);
15            tf[i-1].tf(inputFileName,outputFileName);
16        }
17        System.out.println("complete");
18    }
19 }
```

4.2 実行結果

本節では実行結果として, 標準出力および形態素解析と tf の計算結果について述べる.

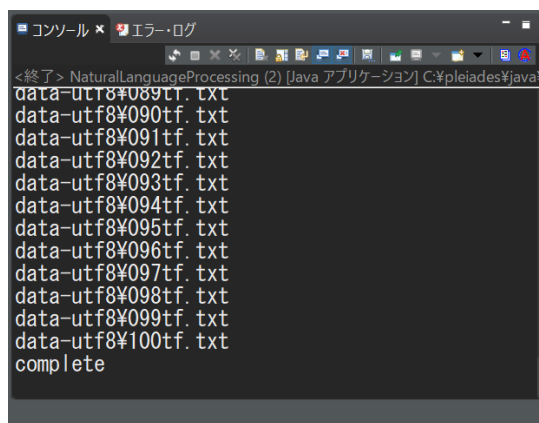
4.2.1 標準出力

リスト 5 に示したように, NaturalLanguageProcessing クラスでは書き込み先ファイルと, 実行終了時に”complete”の文字列を表示する. 図 3 および図 4 は, プログラム実行時の標準出力を示している. 図 3 から 001.txt から順に形態素解析および tf の計算を行い, 図 4 から 100.txt まで同様の処理を行って実行を終了したことがわかる. これより標準出力において, エラーや予期しない挙動が起きることなく処理が行えていることが分かる.



```
<終了> NaturalLanguageProcessing (2) [Java アプリケーション] C:\pleiades¥java\
TF calculate
data-utf8¥001tf.txt
data-utf8¥002tf.txt
data-utf8¥003tf.txt
data-utf8¥004tf.txt
data-utf8¥005tf.txt
data-utf8¥006tf.txt
data-utf8¥007tf.txt
data-utf8¥008tf.txt
data-utf8¥009tf.txt
data-utf8¥010tf.txt
data-utf8¥011tf.txt
data-utf8¥012tf.txt
data-utf8¥013tf.txt
```

図 3: プログラム実行開始時の標準出力



```
<終了> NaturalLanguageProcessing (2) [Java アプリケーション] C:\pleiades¥java\
data-utf8¥089tf.txt
data-utf8¥090tf.txt
data-utf8¥091tf.txt
data-utf8¥092tf.txt
data-utf8¥093tf.txt
data-utf8¥094tf.txt
data-utf8¥095tf.txt
data-utf8¥096tf.txt
data-utf8¥097tf.txt
data-utf8¥098tf.txt
data-utf8¥099tf.txt
data-utf8¥100tf.txt
complete
```

図 4: プログラム実行終了時の標準出力

4.2.2 形態素解析と tf 値の計算結果

形態素解析の結果は 001tf.txt から 100tf.txt に書き込まれている. 例として, リスト 6 に 001.txt のプログラム実行結果を示す. また, リスト 7 に 002.txt の実行結果を示す. リスト 6 およびリスト 7 から表層系, 品詞, 出現回数, tf 値が書き込まれていることがわかる. また文書によって実行結果が異なっていることが読み取れる.

リスト 6: 001tf.txt の抜粋

1	%	名詞	41	0.0764925373
2	天然記念物	名詞	16	0.0298507463
3	E	名詞	16	0.0298507463
4	の	助詞	13	0.0242537313
5	地	名詞	13	0.0242537313
6	〔	記号	11	0.0205223881
7	県	名詞	11	0.0205223881

8]	記号	11	0.0205223881
9	(記号	10	0.0186567164
10)	記号	10	0.0186567164

リスト 7: 002tf.txt の抜粋

1	学,	名詞	44	0.0449438202
2	、,	記号	40	0.0408580184
3	人類,	名詞	38	0.0388151175
4	経済,	名詞	35	0.0357507661
5	の,	助詞	34	0.0347293156
6	・,	記号	26	0.0265577120
7	。,	記号	21	0.0214504597
8	,	記号	19	0.0194075587
9	は,	助詞	17	0.0173646578
10	を,	助詞	16	0.0163432074

出現回数,tf 値の計算が正しく行われていることを確認する。「天然記念物」という単語に着目する。付録のリスト 25 から 001.txt は「天然記念物」が 16 回登場するとわかる。リスト 6 における「天然記念物」の出現回数は 16 になっているから出現回数のカウントは正しく行われていることがわかる。また,excel を代表とするツールを用いると出現回数の総和,つまり全単語数 N は 536 であることがわかる。これより tf 値は $\frac{16}{536} = 0.0298\dots$ と計算できる。リスト 6 における「天然記念物」の tf 値は 0.0298507463 になっているから tf 値の計算も正しく行われていることが確認できた。

次に,tf 値が正規化によって文字数の量に依存しない値になっていることを確認する。tf 値が正規化によって文字数の量に依存しない値になっていることは同じ単語が同じ回数登場する 2 つの文章の全単語数と tf 値を比較することで確認できる。ここでは 001.txt と 002.txt を用いる。表 4 は 001.txt および 002.txt における全単語数, および「出典」の出現回数, および tf 値を示したものである。

表 4: 001.txt と 002.txt の全単語数,「出典」の出現回数と tf 値

項目	001.txt	002.txt
全単語数	536	979
「出典」の出現回数	1	1
「出典」の tf 値	0.0018656716	0.0010214505

表 4 から 001.txt および 002.txt における「出典」の出現回数はどちらも 1 である。しかし,002.txt の全単語数は 001.txt のおよそ 2 倍であるから, tf 値が正規化によって文字数の量に依存しない値であれば,002.txt における「出典」の tf 値は 001.txt における「出典」の tf 値の $\frac{1}{2}$ であることが予想される。表 4 から,002.txt における「出典」の tf 値は 0.0010214505 で,001.txt における「出典」の tf 値は 0.0018656716 であるから, 001.txt における「出典」の tf 値が 002.txt における「出典」の tf 値の 1.8 倍であることが計算できる。これより,tf 値が正規化によって文字数の量に依存しない値であることが確認できた。

5 Inverse Document Frequency の実装と実行結果

本章では Inverse Document Frequency の実装と実行結果について述べる。

5.1 プログラムの説明

Inverse Document Frequency の実装は次の 3 つのクラスを作成, 追記して行う。本節ではこの 3 つのクラスの説明について述べる。

1. DfCount クラス

2. DocumentFrequency クラス

3. NaturalLanguageProcessing クラス (メイン)

5.1.1 DfCount クラス

計算した idf 値を保持するクラスとして, Term Frequency の実装で作成した WordCount クラスを継承して, DfCount クラスを作成する. DfCount クラスはフィールドに idf 値を保持する private Double 型の変数 idf を持つ. またメソッドとしてコンストラクタ, および idf のセッター, ゲッターを設ける. DfCount クラスのソースコードは省略する.

5.1.2 DocumentFrequency クラス

100 件の文書の idf を計算するクラスとして DocumentFrequency クラスを作成する. リスト 8 に DocumentFrequency クラスのソースコードのソースコードを示す. DocumentFrequency クラスではフィールドに DfCount 型のリスト list, および TermFrequency 型の配列 (TermFrequency の実行結果のリスト) を持つ. また, メソッドとしてコンストラクタおよび df メソッドを持つ.

リスト 8: DocumentFrequency クラスのソースコード

```
1 package nlp;
2
3 import java.io.FileWriter;
4 import java.io.IOException;
5 import java.util.ArrayList;
6
7 public class DocumentFrequency
8 {
9     //DF をカウントするためのデータ格納領域の定義
10    ArrayList<DfCount> list = new ArrayList<DfCount>();
11    //DF の元になる TF を受ける
12    TermFrequency tf[];
13    DocumentFrequency(TermFrequency[] tf)
14    {
15        this.tf = tf;
16    }
17    public void df(String outputFilename)
18    {
19        // (ここに処理を定義する)
20        //TF100 ファイル分について繰り返す
21        for(int i=1;i<=100;i++) {
22            System.out.println("df calculate : "+String.format("%03d", i));
23            //TF1 件に含まれる語の分だけ繰り返す
24            for(int j=0;j<tf[i-1].list.size();j++) {
25                //DF のリストの中にエントリがあるか調べる
26                //System.out.println(tf[i-1].list.get(j)
27                .getWord().getHyoujoukei());
28                int k;
29                for(k=0;k<list.size();k++) {
30                    if(list.get(k).getWord().equals(tf[i-1].list.get(j).getWord())) {
31                        list.get(k).setCount(list.get(k).getCount() +1);
32                        break;
33                    }
34                }
35                //リストにエントリが無かったときは, 新しい語としてリストに追加する
36                if (k == list.size()) {
37                    list.add(new DfCount(tf[i-1].list.get(j)
38                        .getWord(), Integer.valueOf(1)));
39                }
40            }
41        }
42        //ソートする
43        list.sort(new WordCompare());
44
45        //今のリストのエントリをすべて表示する
```

```

46 for (int i = 0; i < list.size(); i++){
47     //System.out.println(list.get(i).getWord()
48     .getHyouSoukei() + ":" + list.get(i).getCount());
49 }
50
51 //df, idf の結果をファイルに保存する
52 try {
53     FileWriter fw = new FileWriter(outputFilename);
54     //System.out.println(outputFilename);
55     for (int i = 0; i < list.size(); i++){
56         //System.out.println(fileal.get(i).word
57         .getHyouSoukei() + ":" + fileal.get(i).getCount());
58         list.get(i).setIdf(Math.log10((double) 100 /
59         (double) list.get(i).getCount()+1));
60
61         fw.write(list.get(i).getWord().getHyouSoukei() + "\t"
62                 + list.get(i).getCount() + "\t"
63                 + list.get(i).getWord().getHinshi() + "\t"
64                 + String.format("%.10f", list.get(i).getIdf()) + "\n");
65     }
66     fw.close();
67 }
68 catch (IOException ex) {
69     ex.printStackTrace();
70 }
71 }
72 }

```

コンストラクタでは、文書 100 件分の形態素解析および TermFrequency の計算結果を保持する TermFrequency 型の配列 tf をフィールドの tf に受け取る処理を行う。

df メソッドは tf を用いて df, idf を計算する処理を行う。df メソッドは引数として出力ファイル (String 型) を受け取る。返り値は void 型である。df メソッドの処理内容の詳細は次の通りである。

1. 1 件の文書について、1 つの単語に着目し、その単語がリスト df に含まれていればカウントをインクリメントする。リスト df に含まれていない場合はリストに追加しカウントを 0 にする。
2. 1 件の文書について、すべての単語で同様の処理を行う。
3. 100 件の文書すべてで同様の処理を行う。
4. リスト df を idf 値の降順にソートする。
5. 必要があればリストの内容を表示する。
6. 式 (2) を用いてリスト df のすべての単語について idf 値を計算する。
7. リスト df の全ての単語について、表層系、および出現回数、および品詞、および idf 値を出力ファイルに書き込む。
8. ファイルを閉じて処理を終了する。

5.1.3 NaturalLanguageProcessing クラス (メイン)

tf の計算時に作成したメインクラスに idf の計算の実行を追記する。リスト 9 に NaturalLanguageProcessing クラスのソースコードの追記部分を示す。形態素解析と tf の計算の後に、df クラスとインスタンス化し、df メソッドを実行する処理を追記した。

リスト 9: NaturalLanguageProcessing クラスのソースコード (追記部分)

```

1 System.out.println("DF calculate");
2 DocumentFrequency df = new DocumentFrequency(tf);
3 String outputFilename = "data-utf8\\df.txt";
4 df.df(outputFilename);

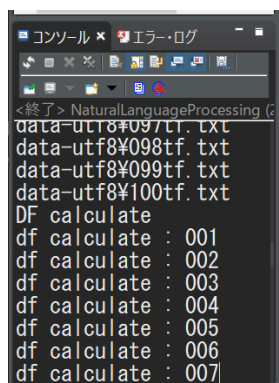
```

5.2 実行結果

本節では実行結果として、標準出力および idf 値の計算結果について述べる。

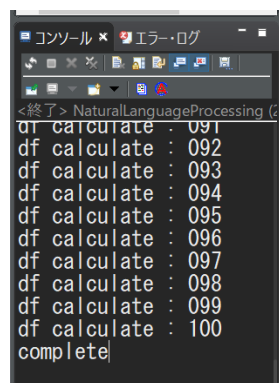
5.2.1 標準出力

NaturalLanguageProcessing クラスおよび DocumentFrequency クラスでは処理を行っている文書の番号と、実行終了時に”complete”の文字列を表示する。図 8 および図 9 に idf の計算実行時の標準出力を示す。図 8 から tf 値の計算終了後に文書 001 から順に idf 値の計算が行われていることがわかる。また、図 9 から文書 100 まで idf 値を計算し、最後に”complete”の文字列を表示して処理を終了している。これより標準出力において、エラーや予期しない挙動が起きることなく処理が行えていることが分かる。



```
<終了> NaturalLanguageProcessing (2)
data-utf8¥097tf.txt
data-utf8¥098tf.txt
data-utf8¥099tf.txt
data-utf8¥100tf.txt
DF calculate
df calculate : 001
df calculate : 002
df calculate : 003
df calculate : 004
df calculate : 005
df calculate : 006
df calculate : 007
```

図 5: idf の計算実行時の標準出力 1



```
df calculate : 091
df calculate : 092
df calculate : 093
df calculate : 094
df calculate : 095
df calculate : 096
df calculate : 097
df calculate : 098
df calculate : 099
df calculate : 100
complete
```

図 6: idf の計算実行時の標準出力 2

5.2.2 idf 値の計算結果

idf の計算結果は df.txt に書き込まれている。リスト 10 に df.txt の冒頭 10 行を示す。また、リスト 11 に df.txt の末尾 10 行を示す。どちらのリストにおいても表層系,df 値, 品詞,idf 値の順になっている。リスト 10 から冒頭 10 行の df 値が全て 100 になっており,idf 値もすべて 1.0 になっていることがわかる。またリスト 11 から末尾 10 行の df 値が全て 1 になっており,idf 値もすべて 3.0 になっていることがわかる。先頭および末尾の 10 行を見ただけでは idf 値の計算結果が正しいかわからない。

リスト 10: df.txt の冒頭 10 行

1	の	100	助詞	1.0000000000
2	(100	記号	1.0000000000

```

3  )   100 記号   1.0000000000
4  、   100 記号   1.0000000000
5  。   100 記号   1.0000000000
6  .   100 名詞   1.0000000000
7  :   100 名詞   1.0000000000
8  は   100 助詞   1.0000000000
9  を   100 助詞   1.0000000000
10 に   100 助詞   1.0000000000

```

リスト 11: df.txt の末尾 10 行

```

1  いた  1  名詞   3.0000000000
2  ずら  1  名詞   3.0000000000
3  風車  1  名詞   3.0000000000
4  シン  1  名詞   3.0000000000
5  フォ  1  名詞   3.0000000000
6  ニア  1  名詞   3.0000000000
7  消防  1  名詞   3.0000000000
8  子ど  1  名詞   3.0000000000
9  も   1  名詞   3.0000000000
10 交響  1  名詞   3.0000000000
11 楽   1  名詞   3.0000000000

```

サクラエディタの grep 検索機能を用いて idf 値の計算結果が正しいか確認する. ここでは「日本人」という単語に着目する.df.txt における「日本人」の idf 値の計算結果は「日本人 6 名詞 2.2218487496」になっている. 図 7 はサクラエディタの grep 検索機能で tf 値の出力結果 (001tf.txt~100tf.txt) のうち「日本人」を含む文書を検索した結果である. 図 7 から「日本人」を含む文書は 6 つあることがわかる. これは df.txt の「日本人」の df 値と一致している. また式 (2) を用いて「日本人」の idf 値を計算すると 2.2218487496 であることがわかる. これより df 値,idf 値の計算が正しいことが確かめられた.

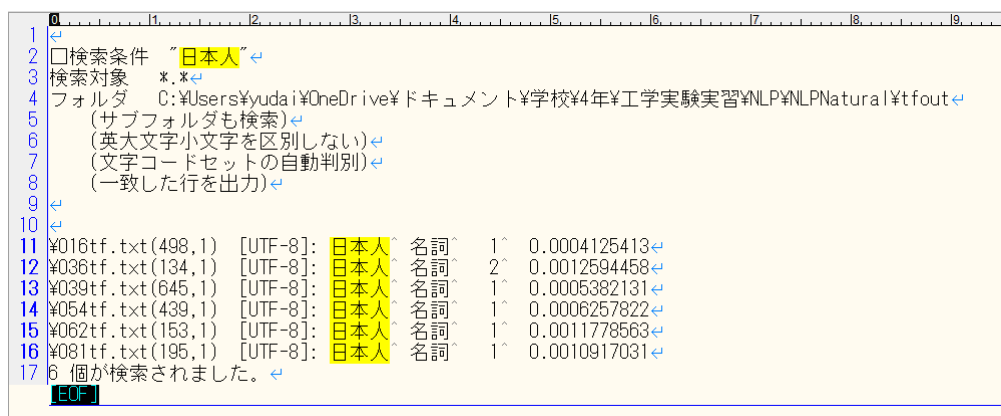


図 7: 「日本人」が出現する文書

6 Term Frequency-Inverse Document Frequency の実装と実行結果

本章では Term Frequency-Inverse Document Frequency の実装と実行結果について述べる.

6.1 プログラムの説明

Term Frequency-Inverse Document Frequency の実装は次の 4 つのクラスを作成, 追記して行う. 本節ではこの 4 つのクラスの説明について述べる.

1. TfIdfCount クラス
2. TfIdfcompare クラス

3. TfIdf クラス

4. NaturalLanguageProcessing クラス (メイン)

6.1.1 TfIdfCount クラス

idf 値および tfidf 値を保持するクラスとして, TfCount クラスを継承した TfIdfCount クラスを作成する. TfIdfCount クラスはフィールドとして, Double 型変数の tfidf, idf を持つ. またメソッドとしてコンストラクタおよびフィールド変数のセッター, およびゲッターを設ける. TfIdfCount クラスのソースコードは省略する.

6.1.2 TfIdfcompare クラス

計算した tfidf 値でリストをソートするために, TfIdfcompare クラスを作成する. リスト 12 に TfIdfcompare クラスのソースコードを示す. TfIdfcompare クラスは TfIdfCount 型の Comparator を実装している. Comparator の実装については WordCompare クラスの説明で行ったため, ここでは省略する.

リスト 12: TfIdfcompare クラスのソースコード

```
1 package nlp;
2
3 import java.util.Comparator;
4
5 public class TfIdfcompare implements Comparator<TfIdfCount>{
6     @Override
7     public int compare(TfIdfCount wc1, TfIdfCount wc2)
8     {
9         if(wc1.getTfidf() < wc2.getTfidf())
10            return 1;
11         if(wc1.getTfidf() == wc2.getTfidf())
12            return 0;
13         if(wc1.getTfidf() > wc2.getTfidf())
14            return -1;
15         return 0;
16     }
17 }
```

6.1.3 TfIdf クラス

文書 100 件分の tfidf を計算するクラスとして TfIdf クラスを作成する. リスト 13 に TfIdf クラスのソースコードを示す. TfIdf クラスはフィールドに TfIdf カウント型のリスト list, および tf, idf の計算結果である tf の配列と df を持つ. またメソッドとしてコンストラクタと tfidf メソッドを持つ.

リスト 13: TfIdf クラスのソースコード

```
1 package nlp;
2
3 import java.io.FileWriter;
4 import java.io.IOException;
5 import java.util.ArrayList;
6
7 public class TfIdf {
8
9     ArrayList<TfIdfCount> list = new ArrayList<TfIdfCount>();
10     TermFrequency[] tf;
11     DocumentFrequency df;
12     TfIdf(TermFrequency[] tf, DocumentFrequency df){
13         this.tf = tf;
14         this.df = df;
15     }
16 }
```

```

17 public void tfidf() {
18     for(int i=1; i<=100;i++) {
19         System.out.println("tf-idf calculate : "+String.format("%03d", i));
20         for(int j=0; j<tf[i-1].list.size();j++) {
21             for(int k=0; k<df.list.size();k++) {
22                 if(tf[i-1].list.get(j).getWord().equals(df.list.get(k).getWord())) {
23                     TfIdfCount tfidfcount = new TfIdfCount(tf[i-1].list.get(j).
24                         getWord(),tf[i-1].list.get(j).getCount(),tf[i-1].list.get(j).
25                         getTf());
26                     list.add(j, tfidfcount);
27                     tfidfcount.setWord(tf[i-1].list.get(j).getWord());
28                     tfidfcount.setTf(tf[i-1].list.get(j).getTf());
29                     tfidfcount.setIdf(df.list.get(k).getIdf());
30                     //System.out.println(df.list.get(j).getIdf());
31                     tfidfcount.setTfidf(tf[i-1].list.get(j).getTf() * df.list.
32                         get(k).getIdf());
33                 }
34             }
35         }
36         list.sort(new TfIdfcompare());
37         try {
38             String outputFileName = "data-utf8\\" +
39                 String.format("%03d", i) + "tfidf.txt";
40             FileWriter fw = new FileWriter(outputFileName);
41             for(int j=0;j<list.size();j++) {
42                 fw.write(list.get(j).getWord().getHyouSoukei() + "\t"
43                     + list.get(j).getWord().getHinshi() + "\t"
44                     + list.get(j).getCount() + "\t"
45                     + list.get(j).getTf() + "\t"
46                     + list.get(j).getIdf() + "\t"
47                     + String.format("%.10f", list.get(j).getTfidf()) + "\n");
48             }
49             fw.close();
50             list.clear();
51         } catch (IOException ex) {
52             ex.printStackTrace();
53         }
54     }
55 }
56 }

```

コンストラクタでは引数として渡された tf および df をフィールドに受け取る処理を行う。
tfidf メソッドは引数、返り値がない。tfidf メソッドの処理内容の詳細を次に示す。

1. 1 件の文書について,1 つの単語に着目し, その単語の tf 値と df 値を検索する。
2. 単語の形態素解析情報,tf 値,idf 値をリストに追加する。
3. tf 値,idf 値から tfidf 値を計算する。
4. 100 件の文書のすべての単語について同様の処理を行う。
5. 100 件の文書について表層系, 品詞, 出現回数,tf 値,idf 値,tfidf 値をファイルに出力する。

6.1.4 NaturalLanguageProcessing クラス (メイン)

メインクラスに tfidf 値の計算の実行を追記する。リスト 14 に NaturalLanguageProcessing クラスのソースコードの追記部分を示す。リスト 14 を idf の計算の後に追加する。NaturalLanguageProcessing クラスでは,idf の計算実行後,TfIdf クラスをインスタンス化し,tfidf メソッドを実行している。

リスト 14: NaturalLanguageProcessing クラスのソースコード (追記)

```

1 System.out.println("Tf-Idf calculate");
2 TfIdf tfidf = new TfIdf(tf,df);
3 tfidf.tfidf();
4 System.out.println("complete");

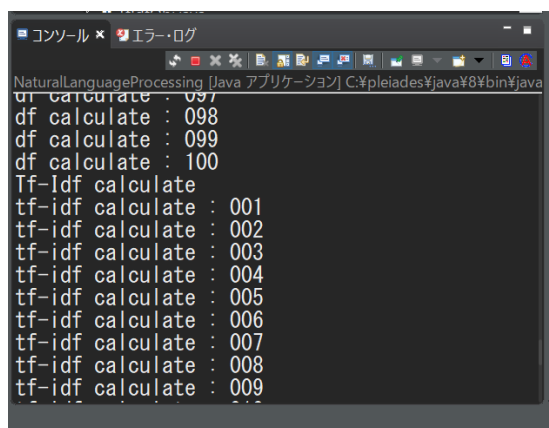
```

6.2 実行結果

本節では実行結果として、標準出力および tfidf 値の計算結果について述べる。

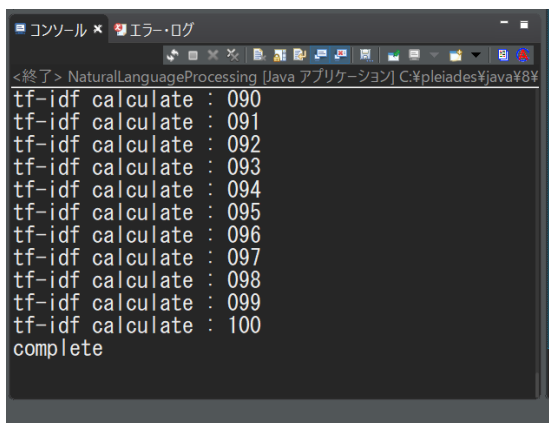
6.2.1 標準出力

リスト 13 に示したように, TfIdf クラスでは tfidf の計算を行っている文書の番号を表示する. またメインクラスでは実行終了時に”complete”の文字列を表示する. 図 8 および図 9 は tfidf の計算実行時の標準出力である. 図 8 から文書 001 から順に tfidf 値の計算を行っていることがわかる. また図 9 から文書 100 まで tfidf 値の計算を実行し,”complete”を出力して処理を終了していることがわかる. これより標準出力において, エラーや予期しない挙動が起きることなく処理が行えていることが分かる.



```
df calculate : 097
df calculate : 098
df calculate : 099
df calculate : 100
Tf-idf calculate
tf-idf calculate : 001
tf-idf calculate : 002
tf-idf calculate : 003
tf-idf calculate : 004
tf-idf calculate : 005
tf-idf calculate : 006
tf-idf calculate : 007
tf-idf calculate : 008
tf-idf calculate : 009
```

図 8: tfidf の計算実行時の標準出力 1



```
<終了> NaturalLanguageProcessing [Java アプリケーション] C:\pleiades\java¥8¥bin¥java
tf-idf calculate : 090
tf-idf calculate : 091
tf-idf calculate : 092
tf-idf calculate : 093
tf-idf calculate : 094
tf-idf calculate : 095
tf-idf calculate : 096
tf-idf calculate : 097
tf-idf calculate : 098
tf-idf calculate : 099
tf-idf calculate : 100
complete
```

図 9: tfidf の計算実行時の標準出力 1

6.2.2 tfidf 値の計算結果

リスト 15 に, tfidf の計算結果を出力した 001tfidf.txt の冒頭 10 行を示す. リスト 15 は各単語について, 左から, 表層系, 品詞, 出現回数, tf 値, idf 値, tfidf 値の順で出力されている. リスト 15 から tfidf 値は降順にソートされていることが読み取れる.

リスト 15: 001tfidf.txt の冒頭 10 行

1	天然記念物	名詞	16	0.029850746268656716	2.6989700043360187	0.0805662688
2	%	名詞	41	0.07649253731343283	1.013228265733755	0.0775044009

3	[記号	11	0.020522388059701493	3.0	0.0615671642	
4]	記号	11	0.020522388059701493	3.0	0.0615671642	
5	地	名詞	13	0.024253731343283583	1.6575773191777938	0.0402024350	
6	両生類	名詞	7	0.013059701492537313	3.0	0.0391791045	
7	生息	名詞	8	0.014925373134328358	2.3979400086720375	0.0357901494	
8	爬虫類	名詞	7	0.013059701492537313	2.6989700043360187	0.0352477426	
9	指定	名詞	9	0.016791044776119403	2.0969100130080562	0.0352093099	
10	E	名詞	16	0.029850746268656716	1.013228265733755	0.0302456199	

tfidf 値の計算結果が正しいことを確認する。ここでは「爬虫類」という単語について注目する。リスト 6 から「爬虫類」の tf 値が 0.0130597015 であることが分かる。また df.txt の出力より「爬虫類 2 名詞 2.6989700043」であるから、「爬虫類」の idf 値が 2.6989700043 であることが分かる。これより「爬虫類」の tfidf 値は 0.0352477426 であることが計算できる。リスト 15 において「爬虫類」の tfidf 値は 0.0352477426 であるから tfidf 値の計算が正しいことが確認できた。

7 実行結果の分析と改良

3 種類の重みづけの方法について実験を行い、正しい出力ができることが確認できた。本章では、実行結果をより細かく分析し、より良い出力結果になるようにプログラムを改良する。ここでの、より良い出力結果とは、プログラム実行による出力結果ファイルを見たときに、元の文書の特徴的な単語が上位に来ていることや、その文書が何について書かれているのかを読み取れる出力結果であることを指す。例えば tf の計算結果の出力ファイルを確認したときに「%」,「を」を代表とするわけのわからない、もしくはどんな文書にも現れるような単語が上位に来るのではなく、「爬虫類」,「人類」を代表とする文書の特徴を表す単語が上位にくることが出力結果の改良にあたる。本章では次に示す実行結果の分析と改良について述べる。

1. Term Frequency の実行結果の分析
2. Inverse Document Frequency の実行結果の分析
3. その他の改良点
4. プログラムの変更点
5. 実行結果

7.1 Term Frequency の実行結果の分析

本節では tf の計算結果の分析として、文書の特徴を表していない語、および欠損値の除去、および品詞別の tf 値について述べる。

7.1.1 文書の特徴を表していない語

tf の出力結果は文書 1 件につき、1 つあるから合計で 100 個の出力ファイルがある。100 個のファイル全てを分析することは難しいから、ここでは 001tf.txt, および 002tf.txt について分析する。図 10 に 001tf.txt, および 002tf.txt の上位の単語とその tf 値を示す。

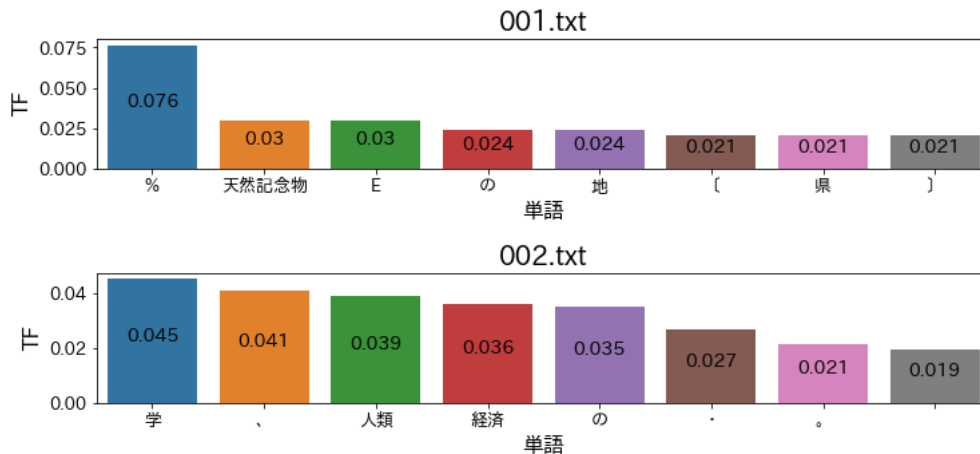


図 10: 001tf.txt, および 002tf.txt の上位の単語とその tf 値

図 10 から,tf 値の高い単語として「天然記念物」,「経済」があることが読み取れる. しかし「%」,「E」,「の」を代表とする文書の特徴を表していない単語の方が上位に多くあることも読み取れる.「の」を代表とする助詞や,「[」は文章中に頻繁に登場するため,tf 値も多くなることが考えられる. また助詞や「[」のように文書を構成する絶対的な単語の tf 値が上位になると,本来文書の特徴を表す単語が相対的に下位になり,出力結果として埋もれてしまう. これらの考えから,助詞を代表とするあらゆる文書に頻繁に登場する単語,および「[」を代表とする文書を構成する絶対的な単語を形態素解析の出力結果から除外する必要があると考える.

7.1.2 欠損値の除去

欠損値 (Missing value) とは出力結果が不正な値やデータになっている値およびデータのことである. tf の計算結果にも標準出力にエラーとして表示されないものの,欠損値が存在する. 例えば 001tf.txt の 52 行目の出力結果は「名詞 2 0.0037313433」である. この出力結果では表層系が空白 (欠損値) になっている. 同様の出力は 002tf.txt の 65 行目でも確認できる. 欠損値が存在すると,001tf.txt を分析する際に上手く読み込みが行えないことや,tf の計算結果を別のプログラムに組み込む際にエラーの原因になりかねない. このため,tf の計算結果から,欠損値が存在する要素を排除すべきであると考え.

7.1.3 品詞別の tf 値

助詞を代表とするあらゆる文書に頻繁に登場する単語,および「[」を代表とする文書を構成する絶対的な単語を形態素解析の出力結果から除外する必要があると述べたが,ここではさらに品詞に着目して分析を行う. 図 11 は 001tf.txt および 002tf.txt における品詞別の tf 値の平均値である.

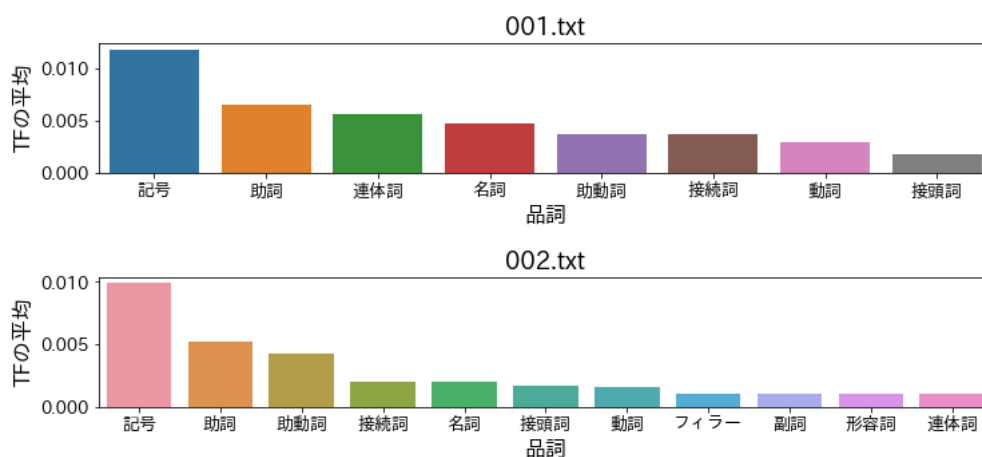


図 11: 001tf.txt, および 002tf.txt の品詞別の tf の平均値

図 11 から 001tf.txt では記号, 助詞, 連体詞の 3 つの tf 値の平均が高く, 002tf.txt では記号, 助詞, 助動詞, 接続詞の 4 つの tf 値の平均が高いことが読み取れる. ここで, 各品詞の国語的意味や役割から, 文書の特徴を表す単語になり得るかを考える. 表 5 に各品詞の代表例を示す.

表 5: 品詞の種類と代表例

品詞	代表例
名詞	東京湾, 富士山, わたし, あれ, 3
動詞	関わる, 行う
形容詞	面白い, 青い
形容動詞	立派な, 便利だ
副詞	ゆっくり, とても, 決して
連体詞	この, あらゆる, 大きな
接続詞	だから, しかし, 最初に, 一方
助詞	は, を, の
助動詞	れる, られる, ます, らしい

表 5 の代表例を用いて各品詞が文書の特徴を表す語か否かを考える. 名詞は「東京湾」, 「富士山」を代表とする固有名詞を含むため, 文書の特徴を表す語になる. 一方で「あれ」, 「3」を代表とする単語は様々な文書に登場するため特徴を表す語にならない可能性があるため, 注意が必要である. 動詞は「関わる」, 「行う」を代表とする動作を表す単語である. 動作を表す単語が文書の特徴を表す単語になる可能性は低いと考える. このため動詞は形態素解析の結果から除外してよいと考える. 形容詞, および形容動詞, および副詞は文書の特徴を表す単語としては評価難しいため今回は除外しない. 連体詞は名詞を修飾する役割があるが, 形態素解析によって文章を分割すると名詞との関係が分からなくなってしまうため除外する. 接続詞は, 前後の分を繋ぐ役割があるが, 連体詞同様に形態素解析によって文章を分割すると前後の文章の関係が分からなくなってしまうため除外する. 助詞, および助動詞は文書中に頻繁に出現する. しかし, これらの助詞, および助動詞はあらゆる文書に頻繁に登場するため, 文書の特徴を表す語ではないと考える. よって助詞, および助動詞を除外する.

7.2 Inverse Document Frequency の実行結果の分析

本節では idf の計算結果の分析として、品詞別の df,idf の平均値、および欠損値の除去、および文書の特徴を表していない語について述べる。

7.2.1 品詞別の df,idf の平均値

図 12 に品詞別の df、および idf の平均値を示す。品詞別の idf の平均値を見ると、名詞、動詞、形容詞、副詞を代表とする品詞の idf の平均値にあまり差がないことが読み取れる。このため、idf の平均値のみを見ると idf による重み付けは品詞と関係ないことが考えられる。また、df の計算結果には品詞が欠損値（ここでは該当なしと表記）が存在することが読み取れる。同様に、品詞が#になっている出力結果が存在することも読み取れる。実験テキストの形態素 ID にも品詞#という項目は存在しない。これらの扱いについては欠損値の除去の節で説明する。

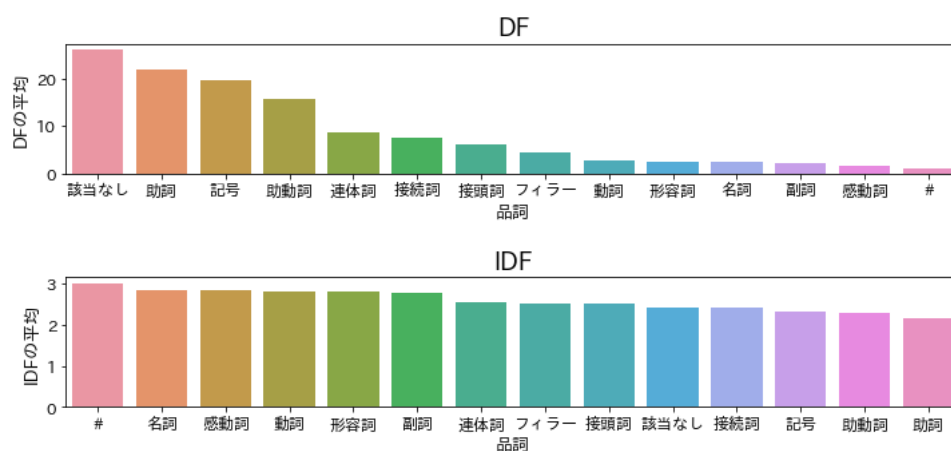


図 12: 品詞別の df および idf の平均値

7.2.2 欠損値の除去

TermFrequency の分析でも欠損値の除去について説明した.idf の実行結果にも欠損値が存在する。表 6 に df.txt における欠損値を含む行とその出力を示す。表 6 から欠損値が生じている項目は単語または品詞であることが読み取れる.df の行数 17176 行に対して、欠損値が生じている単語数は 5 つと十分に小さいから、これらの単語は形態素解析の分析結果から除外する。

表 6: df.txt の欠損値

行数	単語	df	品詞	idf
12	欠損値	100	欠損値	1.00000
4060)	2	欠損値	2.69897
9043	.	1	欠損値	3.00000
10649	”	1	欠損値	3.00000
6956	欠損値	1	#	3.0

7.2.3 文書の特徴を表していない語

idf=1.0 の単語について考える.idf=1.0 の単語 t の df 値は式 (8) より全文書数 N であることがわかる.
 $df(t) = N$ ということはこの単語 t は全文書に必ず登場する単語ということである.

$$idf(t) = \log_{10} \frac{N}{df(t)} + 1 \quad (4)$$

$$1 = \log_{10} \frac{N}{df(t)} + 1 \quad (5)$$

$$\log_{10} \frac{N}{df(t)} = 0 \quad (6)$$

$$\frac{N}{df(t)} = 1 \quad (7)$$

$$df(t) = N \quad (8)$$

具体的にどのような単語が全文書に必ず登場するのか確認する. リスト 16 に idf=1.0 である単語を列挙する. ただし, 単語の区切りはカンマ (,) である. リスト 16 から「の」, 「を」を代表とする助詞が全文書に登場したことが読み取れるが, これについては日本語の文書を構成する絶対的な単語であるから納得できる. 一方で, 「出典」, 「ウィキペディア」, 「編集」という単語が全ての文書に登場する. これらは wikipedia[3] のページを構成する単語であると考えられる. また, これらの単語が全ての文書に登場するということは, 実験の対象にしている文書はすべて wikipedia の何らかのページであると考えられる. ここまでの分析は実験の対象とする文書群がウェブページ, 小説, 論文を代表とするあらゆる種類の文書であるという前提で分析を行ってきたが, 実験の対象としている文書群が全て wikipedia のページであると考えると, wikipedia のページを構成する単語は, 文書の特徴にはなりえないため, 形態素解析の結果から除外して良いと考える. 注意として, 実験の対象とする文書群がウェブページ, 小説, 論文を代表とするあらゆる種類の文書からなる場合は wikipedia のページを構成する単語も文書の特徴を表す語になると考えられる.

リスト 16: idf が 1.0 の単語

```
1 の, (, ), \, ., :, は, を, に, 『, 』, , 「, 」, し, 出典, フリー,  
2 百科, 事典, ウィキペディア, Wikipedia, 移動, ナビゲーション, 検索, た, http,  
3 ://, ja, wikipedia, org, /, wiki, より, 作成, カテゴリ, \%, E, [, ], が,  
4 で, て, 編集, ・, 1, /%, 5, 3, 年, 8, 2
```

7.3 その他の改良点

本節では次に示す 4 つの改良点について述べる.

1. URL の除去
2. ストップワードの除去
3. 大文字, 小文字の統一
4. 数字の扱い

7.3.1 URL の除去

実験の対象とする文書の中には URL (Uniform Resource Locator) を含む文書が存在する. 例えば付録のリスト 25 (001.txt) は URL を含む. 001.txt の形態素解析結果 001.txt を見ると, URL は形態素解析によってリスト 17 に示すように分解される. 形態素解析によって分解された URL はアルファベットや数字の羅列であるから, 文書の特徴を表す文書にふさわしくないと考え, 除外する. しかし, 形態素解析結果から, 元々 URL であった単語を除外するのは難しい. そこで, 元の文書から直接 URL を削除する. wikipedia のページの URL は, "http:" で始まるから, これを文書から検出し, 該当する行を削除することで URL を除外する.

リスト 17: URL の形態素解析結果の例

1	/%	名詞	1	0.0018656716
2	94	名詞	1	0.0018656716
3	F	名詞	1	0.0018656716
4	83	名詞	1	0.0018656716
5	BB	名詞	1	0.0018656716
6	88	名詞	1	0.0018656716
7	AC	名詞	1	0.0018656716
8	99	名詞	1	0.0018656716
9	AB	名詞	1	0.0018656716
10	84	名詞	1	0.0018656716
11	98	名詞	1	0.0018656716
12	BF	名詞	1	0.0018656716
13	89	名詞	1	0.0018656716
14	80	名詞	1	0.0018656716

7.3.2 ストップワードの除去

「機械学習のための特徴量エンジニアリング」[4]によれば,文章によらず一般的に使われる単語を特徴量(文書の特徴を表す語)に加えることはあまり意味がない.このような単語をストップワード(stopword)と呼び,処理の対象外にするのが一般的である.本実験では slothlib[5] という日本語のストップワード辞書を利用し,形態素解析の結果からストップワードを除去する.付録のリスト 26 に slothlib の日本語のストップワード辞書の抜粋を示す.リスト 26 から,ストップワード辞書には,「あちら」,「あれ」を代表とする代名詞や,「年」,「月」を代表とする時間の単位を表す単語,漢数字があることが読み取れる.

また,出現頻度に基づいて文書の特徴に寄与しない単語を除去する方法もある.例えば,カナダ議会での発言を集めた Hansard コーパスでは,“House of Commons”(庶民院)という単語が頻出するため“house”という単語を文書の特徴にすべきではない.このように,通常は有用な単語でも,特定のコーパスでは有用でない単語が存在する.このような単語を見つけ出すために頻度情報を利用する.本実験では df 値が 90 以上の単語を実験で扱うコーパスにおいて有用でないと考え,除去する.この考え方は先に述べた, wikipedai のページからなるコーパスから wikipedia のページを構成する単語を除外すべき,という考え方と同じである.

7.3.3 大文字,小文字の統一

大文字,小文字の統一は英語の場合に有用な方法である.表 7 に idf の計算結果の抜粋を示す.表 7 から“One”と“ONE”が実行結果に混在していることがわかる.英語の場合,文頭の単語の頭文字を大文字にするというルールや,映画の名前によって表記に揺れがあると考えられる.このような表記揺れを抑えるため,元の文書のアルファベットをすべて小文字に置き換える処理を行う.

表 7: 大文字と小文字の混在

行数	単語	df 値	品詞	idf 値
3016	One	2	名詞	2.6989700043
3231	ONE	2	名詞	2.6989700043

7.3.4 数字の扱い

数字の扱いについて考える.数字はリスト 25 に示したように 1 章,2 章.. という章立てや,2020 年,p35 を代表とする多岐にわたる部分で登場する.数字の扱いについては,章立ての場合,これは前述した wikipedia のページを構成する単語であるから除去したい.しかし,「2020 年」,「p35」を代表とする数字との区別が非常に難しい.2 桁以上の数字を除去するという考え方もあるが,章立ての多い wikipedia のページに対応でき

ない。また、形態素解析によって「2020 年」が「2020」と「年」に分解されてしまうため数字のみを見ても単位がわからない。実験結果の改良では出力結果の上位を見て、その文書の特徴がわかるという目標がある。出力結果の数字を見てそれが文書の特徴を表す語であると考え、本実験では数字を形態素解析の結果から除去する。

7.4 プログラムの変更点

分析結果から得られた改良点を次に示す。

1. 記号, 助動詞, 助詞, 接続詞, 動詞, 連体詞, 数字の除去
2. URL の除去
3. ストップワードの除去
4. 大文字, 小文字の統一

以上の改良点を実装するため、既に作成したクラスの改良および新規クラスの作成を行う。本節では次に示すクラスの実装または追記について述べる。

1. Preprocessing クラス
2. Stopword クラス
3. TermFrequency クラス
4. NaturalLanguageProcessing クラス

7.4.1 Preprocessing クラス

URL の除去および大文字, 小文字の統一を行うクラスとして Preprocessing クラスを作成する。リスト 18 に Preprocessing クラスのソースコードを示す。Preprocessing クラスはフィールドに String 型の配列リスト, および入出力ファイル名を持つ。また, メソッドとして, コンストラクタ, および preprocess メソッド, および checkBeforeReadinputFileName メソッドを持つ。

リスト 18: Preprocessing クラスのソースコード

```
1 package nlp;
2 import java.io.BufferedReader;
3 import java.io.File;
4 import java.io.FileNotFoundException;
5 import java.io.FileReader;
6 import java.io.FileWriter;
7 import java.io.IOException;
8 import java.util.ArrayList;
9
10 public class Preprocessing{
11
12     ArrayList<String> list = new ArrayList<String>();
13     private File inputFileName;
14     private File outputFileName;
15     Preprocessing(File inputFileName,File outputFileName){
16         this.inputFileName = inputFileName;
17         this.outputFileName = outputFileName;
18     }
19
20     public void preprocess() {
21         try{
22
23             if (checkBeforeReadinputFileName(inputFileName)){
24                 BufferedReader br = new BufferedReader(new FileReader(inputFileName));
```

```

25
26         String str;
27         while((str = br.readLine()) != null){
28             //System.out.println(str);
29             str = str.toLowerCase();
30             if (!str.contains(" http"))
31             {
32                 list.add(str);
33             }
34
35         }
36
37         br.close();
38     }else{
39         System.out.println("file open error");
40     }
41     }catch(FileNotFoundException e){
42         System.out.println(e);
43     }catch(IOException e){
44         System.out.println(e);
45     }
46
47     try {
48         FileWriter fw = new FileWriter(outputFileName);
49         for(int i=0;i<list.size();i++) {
50             fw.write(list.get(i)+"\n");
51         }
52         fw.close();
53         list.clear();
54     } catch (IOException ex) {
55         ex.printStackTrace();
56     }
57 }
58
59 private static boolean checkBeforeReadinputFileName(File inputFileName){
60     if (inputFileName.exists()){
61         if (inputFileName.isFile() && inputFileName.canRead()){
62             return true;
63         }
64     }
65
66     return false;
67 }
68 }

```

コンストラクタでは、入力ファイル名および出力ファイル名をフィールドに受け取る。checkBeforeReadinputFileName メソッドは引数として与えられたファイル名が存在すれば true, 存在しなければ false を返すメソッドである。preprocess メソッドの処理の詳細は次の通りである。

1. checkBeforeReadinputFileName メソッドを用いて入力ファイルが存在するか確認する。存在しなければ例外処理を行う。
2. 入力ファイルを開く。
3. 入力ファイルを 1 行ずつ読み取り、リストに格納する。ただし、URL(" http") の文字列を含んでいる場合はリストに格納しない。また、リストに格納する際に toLowerCase メソッドを用いて文字列をすべて小文字に変換する。
4. 入力ファイルの終端までたどり着いたらファイルを閉じる。
5. 出力ファイルを開く。出力ファイルは存在しなくても新規に作成されるため、存在するか確認する必要はない。
6. リストの文字列を 1 行ずつ出力する。
7. リストの終端にたどり着いたら出力ファイルを閉じる。

7.4.2 Stopword クラス

ストップワードを読み込み、保持するクラスとして、Stopword クラスを作成する。リスト 19 に Stopword クラスのソースコードを示す。Stopword クラスはフィールドとして String 型の配列リスト stopword および入力ファイルを持つ。また、メソッドとして readStopword メソッド、および checkBeforeReadinputFileName メソッドを持つ。

リスト 19: Stopword クラスのソースコード

```
1 package nlp;
2
3 import java.io.BufferedReader;
4 import java.io.File;
5 import java.io.FileNotFoundException;
6 import java.io.FileReader;
7 import java.io.IOException;
8 import java.util.ArrayList;
9
10 public class Stopword {
11     ArrayList<String> stopword = new ArrayList<String>();
12     public void readStopword(File inputFileName) {
13         try{
14
15             if (checkBeforeReadinputFileName(inputFileName)){
16                 BufferedReader br = new BufferedReader(new FileReader(inputFileName));
17
18                 String str;
19                 while((str = br.readLine()) != null){
20                     //System.out.println(str);
21                     str = str.toLowerCase();
22                     stopword.add(str);
23                 }
24
25                 br.close();
26             }else{
27                 System.out.println("file open error");
28             }
29         }catch(FileNotFoundException e){
30             System.out.println(e);
31         }catch(IOException e){
32             System.out.println(e);
33         }
34     }
35     private static boolean checkBeforeReadinputFileName(File inputFileName){
36         if (inputFileName.exists()){
37             if (inputFileName.isFile() && inputFileName.canRead()){
38                 return true;
39             }
40         }
41
42         return false;
43     }
44 }
```

checkBeforeReadinputFileName メソッドの仕様は Preprocessing クラスと同様である。readStopword メソッドの処理の詳細は詳細は次の通りである。

1. checkBeforeReadinputFileName メソッドを用いて入力ファイルが存在するか確認する。存在しなければ例外処理を行う。
2. 入力ファイルを開く。
3. 入力ファイルを 1 行ずつ読み取り、リストに格納する。
4. 入力ファイルの終端までたどり着いたらファイルを閉じる。

7.4.3 TermFrequency クラス

TermFrequency クラスはコーパスの形態素解析と重みづけ tf を計算するクラスであった。このクラスに記号, 助動詞, 助詞, 接続詞, 動詞, 連体詞, 数字, ストップワードの除去の処理を追加する。リスト 20 に TermFrequency クラスのソースコードの追記部分を示す。リスト 20 のソースコードをリスト 4 の 79 行目に追記する。また, 引数に”Stopword stopwordslist”を追加する。追記部分の処理によって形態素解析の結果から equals メソッドを用いて該当する品詞を検知し, リストから除外する。また引数として受け取ったストップワードのリストに含まれる単語も形態素解析の結果から除外する。

リスト 20: TermFrequency クラスのソースコード (追記)

```
1 // 指定した品詞の語のみを抽出する
2
3 for (int i = 0; i < list.size(); i++){
4     if(list.get(i).getWord().getHinshi().equals("記号")) {
5         //System.out.println(list.get(i).getWord().getHyouSoukei() + "\t");
6         list.remove(i);
7         i--;
8     }else if(list.get(i).getWord().getHinshi().equals("助動詞")) {
9         list.remove(i);
10        i--;
11    }else if(list.get(i).getWord().getHinshi().equals("動詞")) {
12        list.remove(i);
13        i--;
14    }else if(list.get(i).getWord().getHinshi().equals("接続詞")) {
15        list.remove(i);
16        i--;
17    }else if(list.get(i).getWord().getHinshi().equals("助詞")){
18        //System.out.println(list.get(i).getWord().getHyouSoukei() + "\t");
19        list.remove(i);
20        i--;
21    }else if(list.get(i).getWord().getHinshi().equals("連体詞")){
22        list.remove(i);
23        i--;
24    }else if(list.get(i).getWord().getHinshi().equals("数")){
25        list.remove(i);
26        i--;
27    }else if(list.get(i).getWord().getHinshi().length() == 0){
28        list.remove(i);
29        i--;
30    }
31 }
32
33
34 //stopwordの除外
35 for (int i = 0; i < list.size(); i++){
36     for(int j = 0; j<stopwordlist.stopword.size();j++) {
37         if(list.get(i).getWord().getHyouSoukei().equals(stopwordlist.stopword.get(j))) {
38             list.remove(i);
39             i--;
40             break;
41         }
42     }
43 }
```

7.4.4 NaturalLanguageProcessing クラス (メイン)

メインクラスに, コーパスの下処理を追記する。リスト 21 に NaturalLanguageProcessing クラスのソースコードを示す。ただし, idf および tf-idf の計算を行うプログラムは既に示しているから, 省略する。追記部分である 10 行目から 18 行目では 100 件のファイルについて Preprocessing クラスをインスタンス化し, 下処理を行っている。また, 19 行目から 22 行目でストップワードの読み込みを行い, 32 行目で tf メソッドの引数としてオーバーライドした tf メソッドを実行している。

リスト 21: NaturalLanguageProcessing クラスのソースコード (追記)

```

1 package nlp;
2
3 import java.io.File;
4
5 public class NaturalLanguageProcessing
6 {
7     static public void main(String args[])
8     {
9
10        System.out.println("Preprocessing");
11        System.out.println("deleteURL");
12        for(int i=1; i<=100;i++) {
13            File inputFileName = new File("data-utf8\\" + String.format("%03d", i) + ".txt");
14            File outputFileName = new File("data-utf8\\" + String.format("%03d", i)
15            + "changed.txt");
16            System.out.println("data-utf8\\" + String.format("%03d", i) + "changed.txt");
17            Preprocessing pre = new Preprocessing(inputFileName,outputFileName);
18            pre.preprocess();
19        }
20        System.out.println("readstopword");
21        File stopwordName = new File("data-utf8\\stopword.txt");
22        Stopword stopwordlist = new Stopword();
23        stopwordlist.readStopword(stopwordName);
24
25        System.out.println("TF calculate");
26        TermFrequency[] tf= new TermFrequency[100];
27        for(int i=1; i <= 100;i++){
28            tf[i-1] = new TermFrequency();
29            String inputFileName = "data-utf8\\" + String.format("%03d", i) + "changed.txt";
30            String outputFileName = "data-utf8\\" + String.format("%03d", i) + "tfchanged.txt";
31            //System.out.println(inputFileName);
32            System.out.println(outputFileName);
33            tf[i-1].tf(inputFileName,outputFileName,stopwordlist);
34            //tf[i-1].tf(inputFileName,outputFileName);
35        }
36
37        // idfおよびtf-idfのプログラムは省略
38    }
39 }

```

7.5 実行結果

実行結果として、ファイル出力を確認する。リスト 22 に 001tfidf.txt の冒頭 10 行を示す。また、リスト 23 に 002tfidf.txt の冒頭 10 行を示す。リスト 22 およびリスト 23 から、どちらの出力結果においても、リスト 6 やリスト 7 のように「%」を代表とする記号、「の」、「を」を代表とする助詞といった文書の特徴を表していない語が上位にない。また、「天然記念物」、「両生類」、「経済」を代表とする文書の特徴を表す語が上位にあることが読み取れる。

リスト 22: 001tfidf.txt の出力内容 (改良後)

1	天然記念物	名詞	16	0.0851063829787234	2.6989700043360187	0.2296995748
2	両生類	名詞	7	0.03723404255319149	3.0	0.1117021277
3	生息	名詞	8	0.0425531914893617	2.3979400086720375	0.1020400004
4	爬虫類	名詞	7	0.03723404255319149	2.6989700043360187	0.1004935640
5	指定	名詞	9	0.047872340425531915	2.0969100130080562	0.1003839900
6	オオサンショウウオ	名詞	5	0.026595744680851064	3.0	0.0797872340
7	カジカガエル	名詞	3	0.015957446808510637	3.0	0.0478723404
8	ウミガメ	名詞	3	0.015957446808510637	3.0	0.0478723404
9	特別	名詞	4	0.02127659574468085	2.221848749616356	0.0472733777
10	繁殖	名詞	3	0.015957446808510637	2.5228787452803374	0.0402587034

リスト 23: 002tfidf.txt の出力内容 (改良後)

1	人類	名詞	38	0.080338266384778	2.5228787452803374	0.2026837047
2	経済	名詞	35	0.07399577167019028	2.3979400086720375	0.1774374214
3	学	名詞	44	0.09302325581395349	1.7958800173440752	0.1670586063
4	訳	名詞	11	0.023255813953488372	2.221848749616356	0.0516709012

5	社会	名詞	13	0.02748414376321353	1.7695510786217261	0.0486345962
6	文化	名詞	10	0.021141649048625793	1.7695510786217261	0.0374112279
7	市場	名詞	6	0.012684989429175475	2.3010299956639813	0.0291885412
8	ポランニー	名詞	4	0.008456659619450317	3.0	0.0253699789
9	栗本	名詞	4	0.008456659619450317	3.0	0.0253699789
10	慎一郎	名詞	4	0.008456659619450317	3.0	0.0253699789

8 考察

本章では、プログラム改良後の分析、および実験全体の考察について述べる。

8.1 プログラム改良後の分析

プログラム改良後の出力結果を分析し、改良の効果について考察する。単語の除去によってどの程度の単語が除外されたのかを調べる。文書の特徴を表さない単語の tf 値を代表とする重みづけを計算する時間は無駄であるから、定量的に除外された単語の個数を調べることで、どの程度無駄な処理が削減されたかが分かる。表 8 に改良前および改良後における、文書 100 件の単語の分布の統計量を示す。また、図 13 に改良前および改良後における、文書 100 件の単語の分布の箱ひげ図を示す。表 8 および図 13 から、改良前の単語数の分布では平均値が 450 単語付近、中央値が 400 単語付近であることが読み取れる。一方で改良後の単語数の分布は平均が 300 単語付近であるから、単語数全体の傾向として、改良前より平均 150 単語程度減少していることが分かる。また、改良前は外れ値が 6 つ存在するが、改良後は外れ値が 3 つになっている。

表 8: 改良前および改良後の単語数の分布の統計量

統計量	改良前	改良後
平均	452.60	305.950
標準偏差	232.600624	190.211917
最小値	131.00	57.00
第一四分位数	294.500	176.7500
中央値 (第二四分位数)	394.00	251.500
第三四分位数	533.00	398.500
最大値	1323.00	1009.00

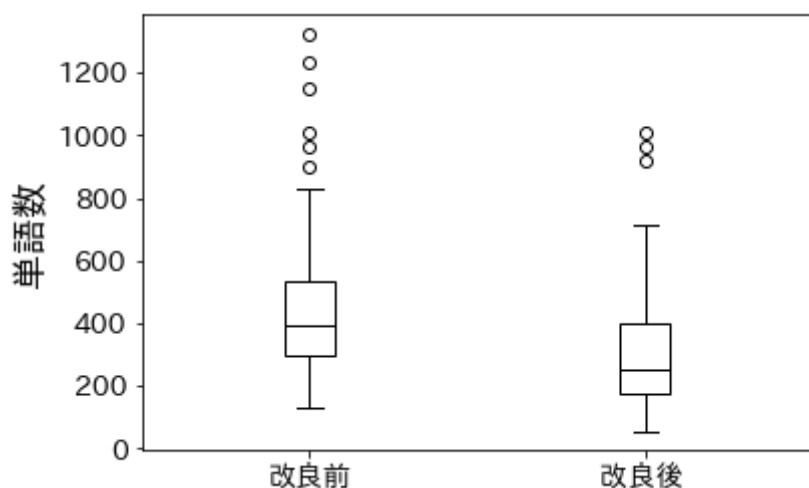


図 13: 改良前と改良後の単語数の分布

改良前の文書の単語数を 1 としたときの, 改良後の文書の単語の割合から, どの程度の単語が除外されたかを確認する. 図 14 に改良前の単語数を 1 としたときの改良後の単語数のヒストグラムを示す. 図 14 から改良後の単語数は, 改良前の単語数のおおよそ 55%から 70%になっていることが読み取れる. これより改良前の単語の 30%から 45%は文書の特徴を表さない意味のない単語であることがわかる. 30%から 45%の単語を除外することによって得られる処理時間の高速化は非常に大きいと考える.

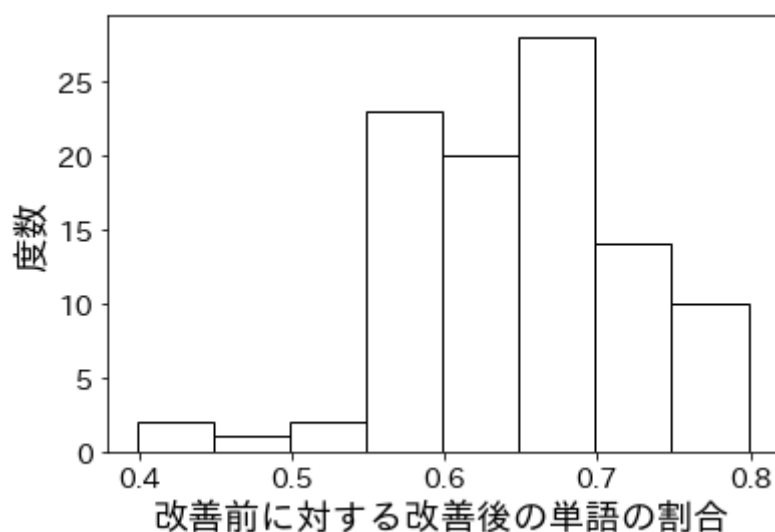


図 14: 改良前に対する改良語の単語の割合

しかし, 改良後も文書の特徴を表さない単語が完全になくなったわけではない. リスト 24 に改良後の df.txt の冒頭 15 行を示す. リスト 24 から「-」, 「.」, 「(」を代表とする記号が残っていることが読み取れる. 記号は除外するように改良したはずである. これらの記号が残っている理由は品詞が名詞になっているためであると考え. 冒頭 15 行以外にも記号が残っているが, df.txt は 14191 単語あるため, これらをひとつずつ確認して除外する作業は困難であると考え.

リスト 24: df.txt の出力内容 (改良後)

1	-	名詞	82	1.0861861476
2	関連	名詞	75	1.1249387366
3	非	接頭詞	75	1.1249387366
4	項目	名詞	72	1.1426675036

5	目次	名詞	72	1.1426675036
6	表示	名詞	72	1.1426675036
7	.	名詞	67	1.1739251973
8	日本	名詞	65	1.1870866434
9	(名詞	59	1.2291479884
10)	名詞	55	1.2596373105
11	/	名詞	54	1.2676062402
12	リンク	名詞	52	1.2839966564
13	外部	名詞	49	1.3098039200
14	現在	名詞	45	1.3467874862
15	編	名詞	42	1.3767507096

8.2 実験全体の考察

形態素解析,Term Frequency,Inverse Document Frequency,tf-idfの実装を通して,自然言語処理における言語処理の基礎的な知識および開発環境を学ぶという目的を達成できたと考える. また,結果を分析し文書の特徴を抽出するように改良を行うという目的は主観的ではあるが達成できたと考える. 客観的に文書の特徴を表す語が抽出できているかを判断する方法としては,Tf-Idfによる重みづけを利用し,2つの文書の類似度を測る方法(コサイン類似度)が存在する. この方法では文書の特徴を表す語が抽出できているか否かが,2文書間の類似度の数値に大きく影響を表すため,文書の特徴を抽出するように改良を行うという目的を評価するよい指標になると考える.

9 付録

本章では付録として 001.txt の内容全文, およびストップワード辞書の内容の抜粋について述べる.

9.1 001.txt の内容全文

リスト 25 に 001.txt の全文を示す.

リスト 25: 001.txt の全文

1	両生類・爬虫類天然記念物一覧
2	出典: フリー百科事典『ウィキペディア (Wikipedia)』
3	移動: ナビゲーション, 検索
4	両生類・爬虫類天然記念物一覧は、日本の文部科学大臣が指定する、天然記念物（特別
5	天然記念物を含む、以下同）のうち、両生類・爬虫類に関わるもののリスト。天然記念
6	物指定基準「動物」に基づき指定されたもののうち、両生類・爬虫類の種および生息地、
7	繁殖地等を掲載する。なお、本項では文化財保護法に基づき国（日本国文部科学大臣）
8	が指定した天然記念物を対象とし、地方自治体指定の天然記念物は対象外とする。
9	
10	目次 [非表示]
11	1 特別天然記念物
12	2 天然記念物
13	2.1 両生類
14	2.2 爬虫類
15	3 関連項目
16	4 参考文献
17	5 外部リンク
18	
19	特別天然記念物 [編集]
20	オオサンショウウオ（種指定）
21	天然記念物 [編集]
22	両生類 [編集]
23	（オオサンショウウオ）
24	オオサンショウウオ生息地〔岐阜県郡上市和良町・八幡町〕
25	オオサンショウウオ生息地〔岐阜県郡上市大和町〕
26	オオサンショウウオ生息地〔岡山県真庭市〕
27	オオサンショウウオ生息地〔大分県宇佐市〕
28	（カジカガエル）
29	湯原カジカガエル生息地〔岡山県真庭市〕

30 南桑カジカガエル生息地〔山口県岩国市〕
 31 (モリアオガエル)
 32 大揚沼のモリアオガエルおよびその繁殖地〔岩手県八幡平市〕
 33 平伏沼モリアオガエル繁殖地〔福島県双葉郡川内村〕
 34 爬虫類【編集】
 35 (アオダイショウ)
 36 岩国のシロヘビ
 37 (アカウミガメ)
 38 御前崎のウミガメおよびその産卵地〔静岡県御前崎市〕
 39 大浜海岸のウミガメおよびその産卵地〔徳島県海部郡美波町〕
 40 キシノウエトカゲ
 41 (クサガメ)
 42 見島のカメ生息地〔山口県萩市〕
 43 セマルハコガメ
 44 リュウキュウヤマガメ
 45 関連項目【編集】
 46 天然記念物
 47 レッドリスト
 48 参考文献【編集】
 49 加藤睦奥雄ら監修『日本の天然記念物』講談社、1995年、ISBN 4-06-180589-4。
 50 外部リンク【編集】
 51 文化庁
 52 国指定文化財等データベース
 53 国宝及び重要文化財指定基準並びに特別史跡名勝天然記念物及び史跡名勝天然記念物指定基準
 54 「<http://ja.wikipedia.org/wiki/%E4%B8%A1%E7%94%9F%E9%A1%9E%E3%83%BB%E7%88%AC%E8%99%AB%E9%A1%9E%E5%A4%A9%E7%84%B6%E8%A8%98%E5%BF%B5%E7%89%A9%E4%B8%80%E8%A6%A7>」より作成
 55 カテゴリ: 両生類・爬虫類天然記念物 | 動物関連の一覧
 57

9.2 ストップワード辞書の内容の抜粋

リスト 26 にストップワード辞書の抜粋を示す。

リスト 26: ストップワード辞書の抜粋

1 あそこ, あたり, あちら, あっち, あと, あな, あなた, あれ, いくつ ...
 2 (中略)
 3 上, 中, 下, 字, 年, 月, 日, 時, 分, 秒, 週, 火, 水, 木, 金, 土, 国, 都, 道, 府, 県, 市, 区, 町, 村...
 4 (中略)
 5 春, 夏, 秋, 冬, 一, 二, 三, 四, 五, 六, 七, 八, 九, 十, 百, 千, 万, 億, 兆, 下記, 上記, 時間, 今回,
 6 前回, 場合, 一つ, 年生, 自分, ケ所, カ所, 力所, 箇所, ケ月...
 7 (後略)

参考文献

- [1] MeCab, <https://taku910.github.io/mecab/>, 閲覧日 2020 年 7 月 12 日
- [2] Java(tm) Platform Standard Edition 8 API, <https://docs.oracle.com/javase/jp/8/docs/api/>, 閲覧日 2020 年 7 月 12 日
- [3] wikipedia, <https://ja.wikipedia.org/wiki/%E3%83%A1%E3%82%A4%E3%83%B3%E3%83%9A%E3%83%BC%E3%82%B8>, 閲覧日 2020 年 7 月 12 日
- [4] Alice Zheng・Amanda Casari, “機械学習のための特徴量エンジニアリング”, 株式会社オーム社, 2019
- [5] slothlib, <http://svn.sourceforge.jp/svnroot/slothlib/CSharp/Version1/SlothLib/NLP/Filter/StopWord/word/Japanese.txt>, 閲覧日 2020 年 7 月 12 日