平成 30 年度卒業研究論文

アンサンブル学習を用いた文書分類

54234　宮下 塁

指導教員　小田幹雄 教授

2019 年 2 月

久留米工業高等専門学校制御情報工学科

目次

第 1 章

序論

近年, インターネットの普及によりお客様の問い合わせ手段が増え, クレームや問い合わせ数が増加傾向にある [1]. そのため, クレームや問い合わせに対して, 素早い対応を行うには, 問い合わせ対応人員の増大など膨大な人件費が必要である. クレームや問い合わせに対して, 人手で行われている知的作業を自動化, 支援することができれば, 多くの企業にとって有益である [2]. 本研究では, 共同研究を行っている保険会社の問い合わせ文章データをクレームと非クレームに推定し, 精度の向上を目的としている. 推定には, 単一学習器やアンサンブル学習器など様々な手法を用いた.

第 2 章

背景

2.1　自然言語処理

自然言語処理とは, 人間が普段使用している自然言語をコンピュータに処理させる技術のことである. 自然言語処理では, 自然言語をコンピュータが理解し, 人間の生活に役立つことを目標としている. 具体的な流れとして, 形態素解析を行う必要がある. 形態素解析とは, 単語の品詞等の情報を格納した辞書を元に, 文章を意味のある最小単位まで分割することである. また, 日本語の文章では英語やその他の言語のように単語ごとに空白がなく, 1 文が繋がっている. この状態ではコンピュータが処理する際に扱いにくいため, 分かち書きを行い, 単語ごとに空白を入れるのが一般的である. 具体的な流れが表 2.1 である.

文章	吾輩は猫である。
形態素解析	吾輩/は/猫/で/ある/。
分かち書き	吾輩　は　猫　で　ある　。

表 2.1. 文章の形態素解析

形態素解析に用いる辞書は, 主に MeCab や Juman++,Janome などが挙げられる. 本研究では,MeCab の ipadic-NEologd という新語や固有表現により多く対応している辞書を用いた.

2.2　単語・文章のベクトル化手法

形態素解析を用いて文章を単語ごとに分割したが, このままではコンピュータが扱えないため, ベクトルに変換する必要がある. 単語を低次元の意味空間上でベクトル表現で表すものを単語ベクトルという. 単語を単語ベクトルに変換する手法は主に 2 つある.1 つは文章全体での単語の出現頻度や共起などの情報をもとに変換する方法で, 代表的な手法で BoW や TFIDF などが挙げられる. もう 1 つは意味的に類似している単語は似た文脈に出現するという仮定を前提に単語の周辺の文脈から単語ベクトルに変換する方法である. 代表的な手法で Word2Vec がある.

2.2.1　Bag of Words (BoW)

BoW は文書の形態素解析の結果から, 単語ごとに出現回数をカウントしたものを特徴ベクトルに変換する方法である.

文章	A 子供 が 走る B 車 が 走る C 子供 の 脇 を 車 が 走る
出現単語	1:子供 2:が 3:走る 4:車 5:の 6:脇 7:を
変換	A [1110000] B [0111000] C[1111111]

表 2.2. BoW の計算処理

表 2.2 に示すように,BoW では文章の構造は無視し, どの単語がどれだけ使用されているかを考える.

2.2.2　TFIDF

TFIDF はある文章における特徴語を抽出する場合に用いられ, 単語の出現頻度 TF(BoW) と文書全体におけるその単語の出現しにくさ (IDF) から求めることができる. 全体の文書数を N, 文書 d で単語 t が出現する回数を $n_{d,t}$, 単語 t が出現する文書数を df_t, 単語数を T とする. TFIDF は式 (2.1) で定義され,TF および IDF はそれぞれ式 (2.2), 式 (2.3) で表される.

$$TF - IDF_{d,t} = TF_{d,t} * IDF_t \tag{2.1}$$

$$TF_{dt} = n_{d,t} \tag{2.2}$$

$$IDF_t = \log \frac{N}{df_t} \tag{2.3}$$

2.2.3　Word2Vec

ニューラルネットワークに単語を入力し, 各単語の出現確率を出力とすることで, 単語をベクトル表現にする手法が Word2Vec である．Word2Vec には CBoW と Skip-gram の 2 種類の手法がある. 図 2.1 に CBoW のモデルを示す．CBoW は周辺の単語などの文脈を入力とし, ターゲットとしている単語の出現確率が最大となるように学習を行うことで単語ベクトルに変換する. 図 2.2 に Skip-gram のモデルを示す.Skip-gram ではターゲットとしている単語を入力にして, 周辺単語の出現確率の和が最大となるように学習を行うことで単語ベクトルに変換する.

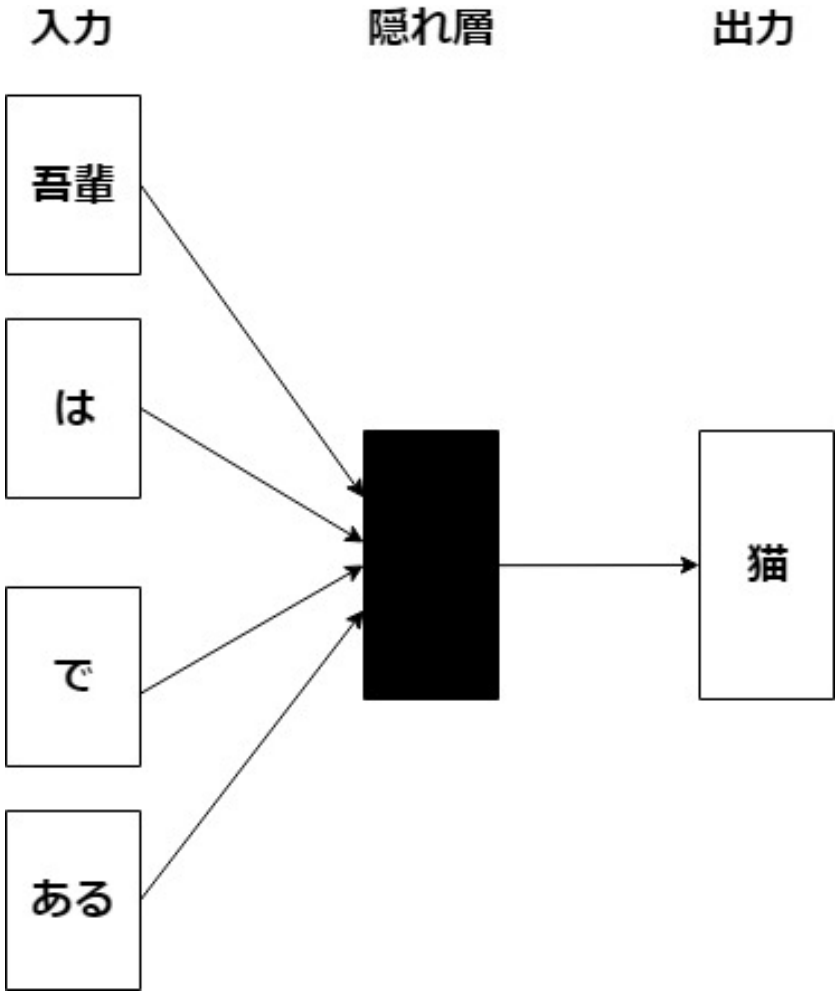

図 2.1. CBoW

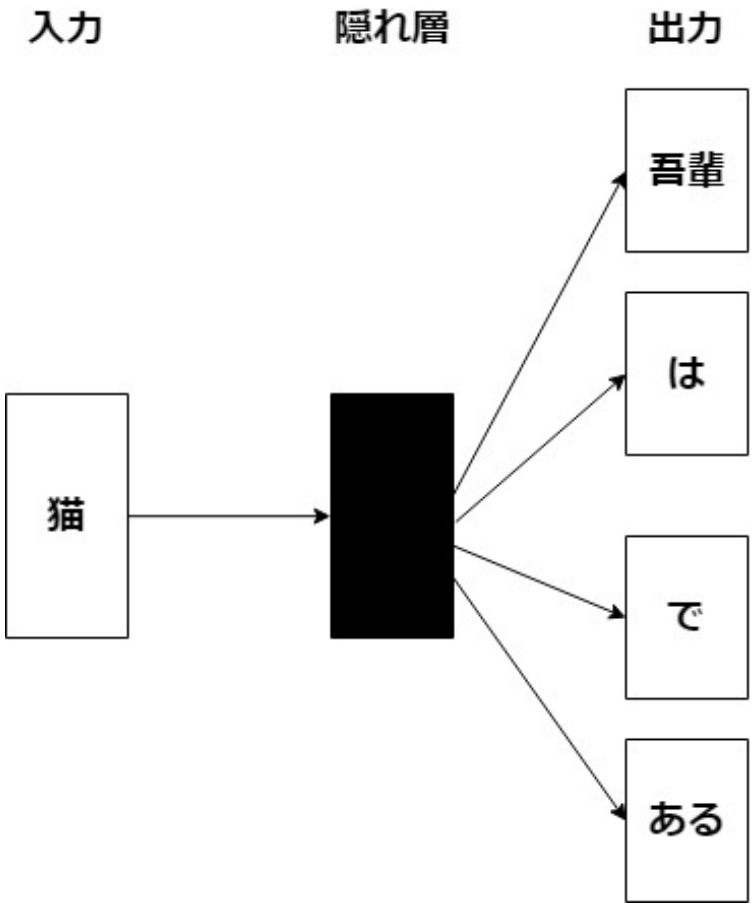

図 2.2. Skip-gram

2.2.4　Sparse Composite Document Vectors(SCDV)

SCDV は文章をベクトル表現になおす場合に次元削減を取り入れた手法である．SCDV のアルゴリズムを以下に示す．

1. Word2Vec を用いて各単語 w_i の単語ベクトル $w\vec{v_i}$ を取得
2. 各単語の IDF 値 $idf(w_i)$ を取得
3. 混合ガウスモデル (GMM) を用いて単語を K 個のクラスタにクラスタリング
4. 単語 w_i が各クラスタ c_k に属する確率 $P(c_k|w_i)$ を取得
5. 単語ベクトルに各クラスタに属する確率を考慮し新たな単語ベクトル $wcv\vec{}_{ik}$ を取得 $wcv\vec{}_{ik} = w\vec{v_i} * P(c_k|w_i)$
6. IDF 値を考慮し新たな単語ベクトル wtv_i を取得 $w\vec{tv}_i = idf(w_i) * \oplus_{k=1}^{K} wcv\vec{}_{ik}$
7. 各文章に含まれる単語ベクトルを足し合わせて平均し文章ベクトルを取得
8. 文章ベクトルをスパースにして SCDV となる

2.3　データ不均衡調整

本研究で扱う問い合わせ文章データは, 非クレーム文章に対してクレーム文章が極端に低く, データが不均衡である. 一般的に不均衡データを学習器で用いると精度が悪くなる傾向があるため, データに対して調整を行い, クレームと非クレームの割合を調整する必要がある. 具体的な手法として少数派データの数を増やすオーバーサンプリング, 多数派データの数を減らすアンダーサンプリング, オーバーサンプリングとアンダーサンプリングの両方を用いるハイブリッドサンプリングの 3 つが挙げられる.

SMOTE
オーバーサンプリングの一つに SMOTE がある.SMOTE は少数派データ点に対して, 同じクラスの K 近傍点との間にデータ点をランダムに生成する方法である.K=3 とした場合の SMOTE を図 2.3 に示す.

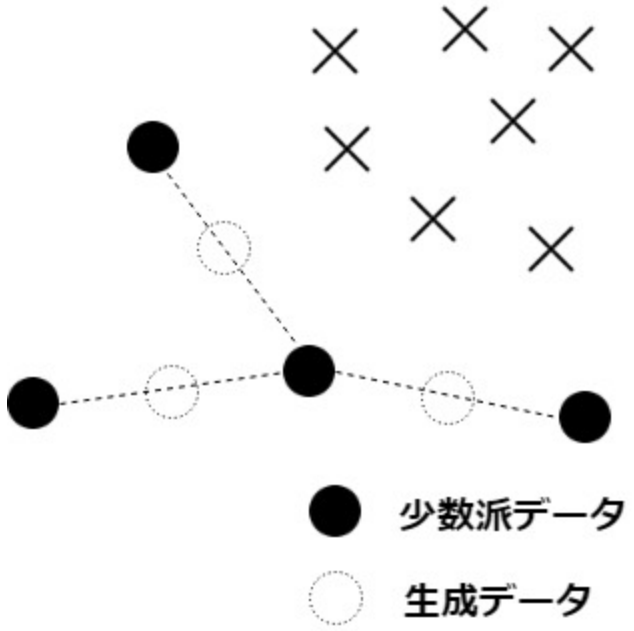

図 2.3. SMOTE(K=3)

SMOTE では, 生成されるデータ点が既存のデータ点を基準にしているため, 同じような特徴をもつデータが増えることになるので, 増やす量に気を付けなければ過学習を起こす可能性がある.

ENN

アンダーサンプリングの一つに ENN がある.ENN は, あるデータ点を考えた時に, 近傍に同じクラスのデータ点が K 個存在しない場合にそのデータ点を削除することで多数派のデータを減らす方法である. K=3 とした場合の ENN を図 2.4 に示す.

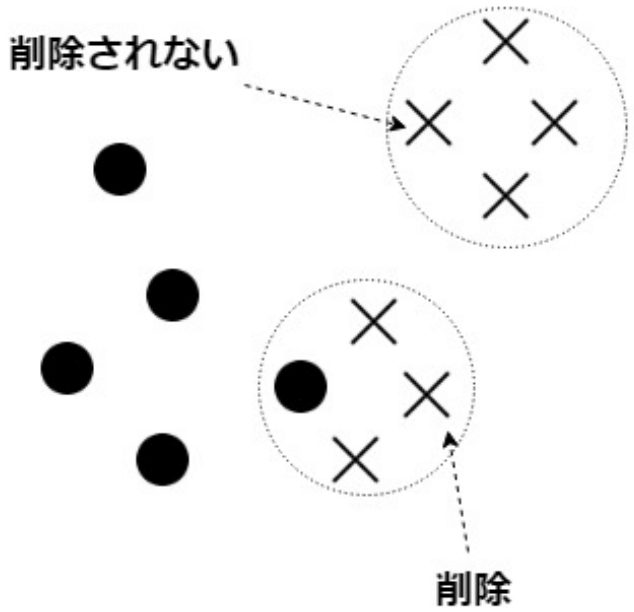

図 2.4. ENN(K=3)

ENN では, 削除するデータ数に気を付けなければ, 重要な特徴を示すデータ点を誤って削除し, 多数派データの特徴が失われる可能性がある.

SMOTEENN

SMOTEENN とは, オーバーサンプリング手法の SMOTE とアンダーサンプリング手法の ENN を組み合わせたハイブリッドサンプリング手法である. 多数派データと少数派データのどちらも調整することでデータの不均衡問題を解決する.

2.4　機械学習

機械学習とは, コンピュータに入力データを学習させ, そこから規則を作り新たなデータに対して作った規則から未来を予測することである. 機械学習の主な手法に教師あり学習がある.

2.4.1　教師あり学習

教師あり学習とは, 入力データに対する正解データを事前に用意しておくことで規則を見出す手法である. 主に, 分類問題に用いられる.

2.4.2　ロジスティック回帰

ロジスティック回帰はベルヌーイ分布に従う変数の回帰モデルである.2 分類問題に対して有効な手法であり, 確率をもとに分類する. ロジスティック回帰はシグモイド関数 (式 (2.5)) を用いて 0.5 以上か 0.5 未満かを判断し分類する. また, 図 2.5 は標準シグモイド関数 (a=1) のグラフである.

$$f(x) = \frac{1}{1 + \mathrm{e}^{-ax}} \tag{2.4}$$

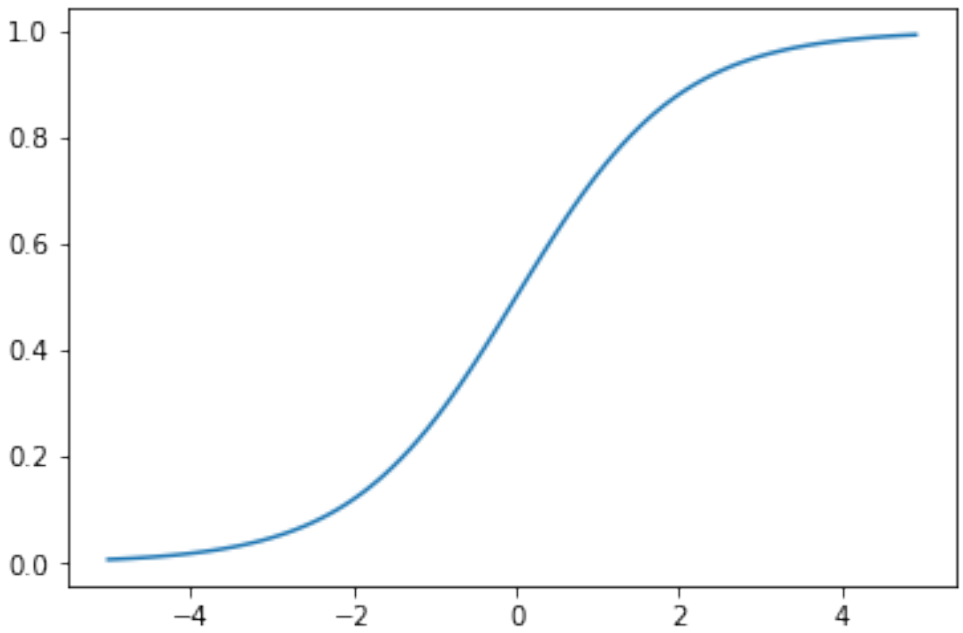

図 2.5. 標準シグモイド関数

2.4.3 K 近傍法

特徴空間の中である範囲 K における各クラスのデータの量で分類を行う. 図 2.6 に K 近傍法のデータ分類の方法を示す.

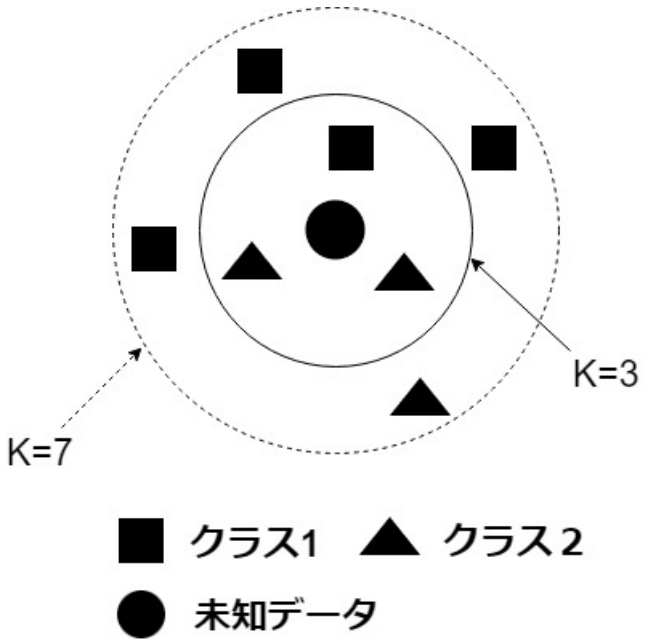

図 2.6. K 近傍法のデータ分類

K=3 の場合未知データはクラス 2 に分類されるが,K=7 の場合はクラス 1 に分類される. このように,K 近傍法ではパラメータ K をどのように設定するかで予測結果が大きく変わるが, 一般的に K が大きいほど外れ値やノイズの影響を受けにくくなり, クラス間の境界が明確にならない傾向がある. また,K=1 のときを最近傍法といい, 最も近いデータのクラスに分類される.

2.4.4 SVM(サポートベクターマシン)

SVM は分類, 回帰どちらにも用いられる学習器である. クラス分類をするときに線形分離の場合超平面は無限にあるが, 各データ点との距離が最大となる超平面を求めることをマージン最大化という. 一般的に高い精度を得られるが, パラメータの調整が難しいことや学習時間が長いことが欠点として挙げられる.

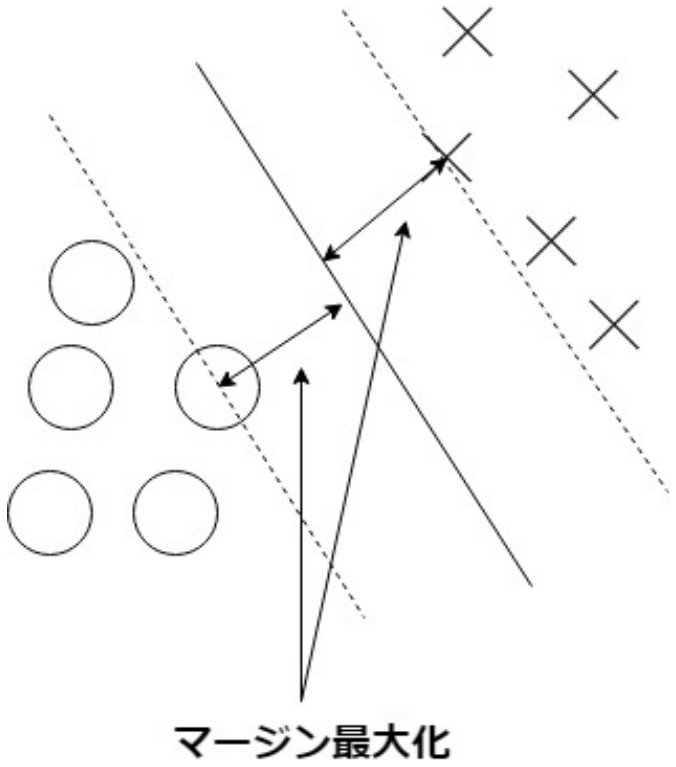

図 2.7. マージン最大化

2.4.5　ナイーブベイズ

ベイズの定理を利用して分類する手法で, 主にベルヌーイ分布, 多項分布, ガウス分布の 3 種類がある. 入力単語を X, 出力クラスを Y とすると, ベイズの定理は式 (2.5) で求められる.

$$P(Y|X_1,\cdots,X_n) = \frac{P(Y)P(X_1,\cdots,X_n|Y)}{P(X_1,\cdots,X_n)} \tag{2.5}$$

今, 単純な条件付き独立仮定を使用すると

$$P(X_i|Y,X_1,\cdots,X_{i-1},X_{i+1},\cdots,X_n) = P(X_i|Y) \tag{2.6}$$

全ての i について考えると,

$$P(Y|X_1,\cdots,X_n) = \frac{P(Y)\prod_{i=1}^{n} P(X_i|Y)}{P(X_1,\cdots,X_n)} \tag{2.7}$$

$P(X_1,\cdots,X_n)$ は定数なので式 (2.7) は次式となる.

$$P(Y|X_1,\cdots,X_n) \propto P(Y)\prod_{i=1}^{n} P(X_i|Y) \tag{2.8}$$

ベルヌーイ分布

ベルヌーイナイーブベイズは多変量ベルヌーイ分布に従い 2 値分類をする手法である．ベルヌーイナイーブベイズの決定規則は次式で表される．

$$P(X_i|Y) = P(i|Y)X_i + (1 - P(i|Y))(1 - X_i) \tag{2.9}$$

多項分布

多項ナイーブベイズは多項分布に従い分類する手法である．1 文に含まれる単語の数を n, 最尤度の平滑化パラメータを θ_y, スムージングパラメータを α とすると,

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n} \tag{2.10}$$

ただし, $N_{yi} = \sum_{x \in T} x_i$, $N_y = \sum_{i=1}^{n} N_{yi}$ である．

ガウス分布

ガウスナイーブベイズはガウス分布に従い分類する手法である．パラメータ σ_y, 最大尤度 μ_y とする．

$$P(X_i|Y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp(-\frac{(Xi - \mu_Y)^2}{2\sigma_y^2}) \tag{2.11}$$

2.4.6 アンサンブル学習

様々な学習器をまとめて一つの学習器とした手法をアンサンブル学習という．一般的にアンサンブル学習を用いるとナイーブベイズや SVM といった単一学習器よりも精度が高くなる傾向がある．アンサンブル学習の主な手法にバギングとブースティングがある．

バギング

複数の異なる学習器の予測から多数決で分類する方法をバギングという．$\mathbf{K}$ 個 ($\mathbf{K}$ は奇数) の学習器のアンサンブル学習を考えるとして, 各学習器の誤分類の確率を一律 θ とする．また,$\mathbf{K}$ 個の学習器のうち,$\mathbf{k}$ 個の学習器の誤分類確率 $\mathbf{P(k)}$ は,

$$P(k) = {}_K C_k \theta^k (1 - \theta)^{K-k} \tag{2.12}$$

となる [3]. 式 (2.12) をもとに,$\mathbf{K=5}$,$\theta\mathbf{=0.3}$ の場合の $\mathbf{P(k)}$ を図 2.8 に示す.

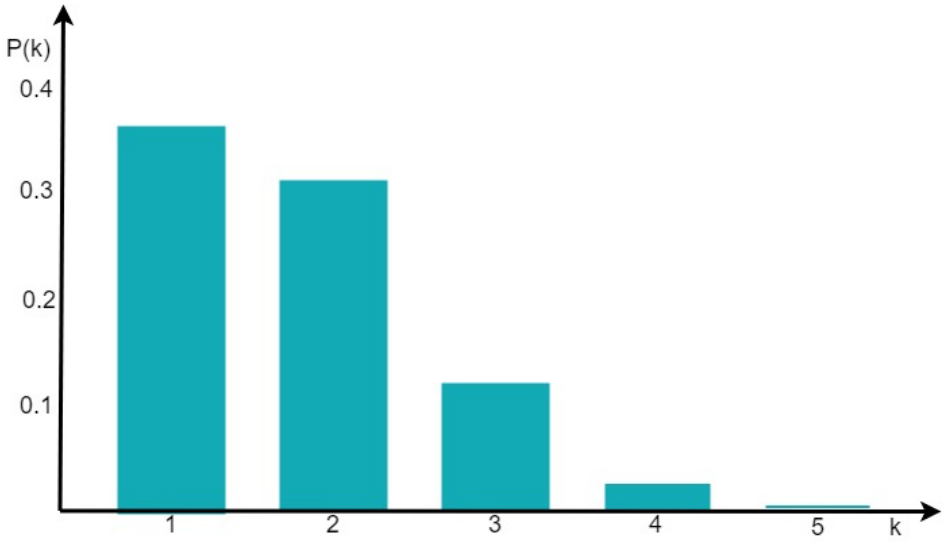

図 2.8. k 個の学習器の誤分類確率

図 **2.8** から, 過半数の学習器が誤分類する確率は, 単一学習器が誤分類する確率 θ**=0.3** より小さいことがわかる. これがバギングの効果である.

ブースティング

同じデータ, 学習器を用いて誤った予測をしたデータに重み付けし, 学習する方法をブースティング法という. 用いる学習器を K 個とし, それぞれ $g_i(x)$ と表し, 誤った予測に対しての重みを α_i とすると重み付き平均したアンサンブル学習器の識別関数は式 **(2.13)** となる.

$$g(x) = \sum_{i=1}^{K} \alpha_i g_i(x) \tag{2.13}$$

2.5　評価指標

機械学習に用いた学習器の評価をするときには主に以下の種類がある．

- 正解率 (正確度)
- 再現率 (真陽性率)
- 適合率 (陽性反応的中度)
- **F1** スコア

2 クラス分類の評価結果を直感的に理解しやすくしたものとして混同行列がある．表 **2.3** に混同行列を示す．

	正解　非クレーム	正解　クレーム
予測　非クレーム	TN(真陰性)	FN(偽陰性)
予測　クレーム	FP(偽陽性)	TP(真陽性)

表 2.3. 混同行列

正解率 (**Accuracy**) とは正解に対して予測結果がどれだけ一致しているかを表し，全体における **TP**(**True Positive**) と **TN**(**True Negative**) を足した数の割合で求められる (式 **(2.14)**)．

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{2.14}$$

再現率 (**Recall**) とは正解がクレームのときに実際にクレームと予測したものの割合を示し，式 **(2.15)** で表される．

$$Recall = \frac{TP}{TP + FN} \tag{2.15}$$

適合率 (**Precision**) とは予測結果がクレームである場合に実際に正解がクレームである割合を示し，式 **(2.16)** で表される．

$$Precision = \frac{TP}{TP + FP} \tag{2.16}$$

F1 スコア (**F1-score**) とは適合率，再現率の調和平均から求められる指標で式 **(2.17)** で表される．

$$F1 - score = \frac{2 * Recall * Precision}{Recall + Precision} \tag{2.17}$$

一般的に学習器の評価をするときは正解率を用いることが多い. しかし, 不均衡なクラスを扱う場合に正解率では正確な評価をすることができない. 表 **2.4** に具体例を示す.

目的:癌の検出	99% の陰性と 1% の陽性からなるデータ
正解率	すべて陰性と答えると正解率 = 99%
問題点	正解率が高く性能の良い学習器に見えるが,1% の陽性反応が検出できていない.

表 2.4. 正解率の評価が正しく扱えない例

このように正解率が使えない場合に代用されるのが **F1** スコアである.

第 3 章

実験

本研究で用いる文章データセットは **8397** 件の非クレームデータと **1416** 件のクレームデータから構成されており, 本節で示す実験結果は同じ実験を **5** 回繰り返しその平均で求めたものであり, 全て単位は % である.

3.1 単一学習器

文章分類で一般的に用いられるナイーブベイズでデータセットを学習し, 評価を行った. 文章のベクトル化には **TFIDF** を用いており,**l2** ノルムで正規化をした. また, 本実験で用いたナイーブベイズは **Scikit-learn** ライブラリの

- **BerunoulliNB**(ベルヌーイ分布)
- **MultinomialNB**(多項分布)
- **GaussianNB**(ガウス分布)

であり, ハイパーパラメータは全てデフォルトである. また, ベルヌーイ分布と多項分布のナイーブベイズを用いてストップワードを作った場合と作っていない場合で実験を行った.

3.1.1 ストップワードなし

実験結果を表 **3.1** に示す.

	再現率 R	適合率 P	F1 スコア
ベルヌーイ分布	63.8408	63.5603	63.6927
多項分布	72.0327	73.7149	72.8500

表 3.1. ストップワードなし

3.1.2　ストップワードあり

データセットから不必要であると判断した単語についてストップワード除去を行い精度の向上を図った．ストップワードに選んだ単語を表 **3.2** に, その実験結果を表 **3.3** に示す．

。 - ( ) XXX , [ ] XX ※ ・

表 3.2. ストップワード

	再現率 R	適合率 P	F1 スコア
ベルヌーイ分布	67.3729	58.6777	62.7104
多項分布	78.5298	63.8044	70.3842

表 3.3. ストップワードあり

3.1.3　特徴抽出

TFIDF

TFIDF 値の上位 **200** 単語を特徴抽出し実験を行った.**TFIDF** 値の上位とは全単語の中で各クラスの特徴を強く示している単語のことである．表 **3.4** に結果を示す．

	再現率 R	適合率 P	F1 スコア
ベルヌーイ分布	65.7477	58.6231	61.9690
多項分布	44.9138	90.0655	59.9319
ガウス分布	77.2577	43.7883	55.8652

表 3.4. 特徴抽出 (TFIDF)200 単語

単語確率

ナイーブベイズは単語が各クラスに属する確率をもとに分類を行っているため, 各単語はクレームクラス, 非クレームクラスに属するそれぞれの確率を所有している. 本実験では,**MultinomialNB** を使い, クレームクラス, 非クレームクラスに属する確率がそれぞれ高い単語を上位から **1000** 語,**2000** 語,**3000** 語,**4000** 語,**5000** 語特徴抽出した場合の精度の変化を比較した. 例えば **1000** 語とはクレームクラスである確率が高い単語上位 **1000** 語と非クレームクラスである確率が高い単語上位 **1000** 語の計 **2000** 語を特徴抽出していることを表す. 表 **3.5** に結果を示す.

特徴抽出単語数	再現率 R	適合率 P	F1 スコア
1000	76.4107	66.2459	70.9331
2000	77.6825	65.0737	70.7801
3000	77.4693	66.4111	71.4499
4000	72.0313	72.2883	72.1224
5000	71.7461	72.169	71.7836

表 3.5. 特徴抽出 (単語確率)

3.2　アンサンブル学習

3.2.1　TF および TFIDF

TF(BoW) および **TFIDF** を用いたバギングで比較実験を行った**.** バギングには **Scikit-learn** の **VotingClassifier** を用いており**,**

- ロジスティック回帰
- ランダムフォレスト
- **K** 近傍法
- **SVM**
- ナイーブベイズ **(GaussianNB)**

の **5** つの学習器を用いた**.** 表 **3.6** にバギングの結果を示す**.**

	再現率 R	適合率 P	F1 スコア
TF	56.8494	96.202	71.4498
TFIDF	59.5322	96.1465	73.5037

表 3.6. ナイーブベイズの精度

3.2.2　SMOTEENN および ENN

SMOTEENN でデータ調整した場合と **ENN** でデータ調整をした場合の精度の比較を行った**.** なお**,SMOTEENN** と **ENN** のどちらもハイパーパラメータはデフォルトである**.** 単語ベクトルへの変換は **Word2Vec** を用いて**,SCDV** で次元削減を行った**.** 表 **3.7** に各学習器とバギングの結果を示す**.**

学習器	再現率 R	適合率 P	F1 スコア
ロジスティック回帰	65.2015	62.0209	63.5714
ランダムフォレスト	54.9451	68.4932	60.9756
K 近傍法	82.0513	39.3673	53.2067
SVM	59.7070	61.0487	60.3704
ナイーブベイズ	74.7253	37.2263	49.6955
バギング	63.7363	60.2076	61.9217

表 3.7. バギングの精度 (SMOTEENN)

学習器	再現率 R	適合率 P	F1 スコア
ロジスティック回帰	75.5245	64.2857	69.4534
ランダムフォレスト	68.1818	71.4286	69.7674
K 近傍法	88.4615	42.8088	57.6967
SVM	52.0979	85.1429	64.6421
ナイーブベイズ	91.2587	24.0775	38.1022
バギング	74.8252	65.8462	70.0491

表 3.8. バギングの精度 (ENN)

3.2.3　XGBoost

ブースティングの一つに **XGBoost** がある**.XGBoost** とはランダムフォレストに勾配ブースティング法を取り入れた学習器である．**XGBoost** には多くのハイパーパラメータがあるため**,** 本実験ではハイパーパラメータがデフォルトの場合と最適化した場合の精度の比較を行った**.** なお**,** ハイパーパラメータ最適化には **optuna** を利用しており試行回数は **100** 回である．表 **3.9** にハイパーパラメータの一覧を示し**,** 表 **3.10** に結果を示す**.**

パラメータ	最適範囲	パラメータ説明
learning_rate	0.01 , 0.5	ステップ数を調整し過学習を防ぐ
max_depth	4 , 12	木の深さ　大きくしすぎると過学習が発生
min_child_weight	1 , 6	葉の重み 大きすぎると過学習防止になるが単調になり精度低下
subsample	0.01 , 1	各ステップで用いるデータの割合
colsample_bytree	0.01 , 1	各ステップごとに用いる特徴量の割合
reg_lambda	1 , 5	L2 正規化項の重み
gamma	0.001 , 5	ノードを追加で分割する

表 3.9. XGBoost ハイパーパラメータ

パラメータ	再現率 R	適合率 P	F1 スコア
デフォルト	68.503	62.5198	65.3473
最適化	75.8503	60.4336	67.27

表 3.10. XGBoost の精度

第 4 章

考察

表 3.1,3.3 よりストップワードを用いた場合再現率が上がり適合率が下がることがわかった．ストップワードをさらに増やして同様の傾向がみられる場合, 再現率はさらに高くなりクレームクラスの誤検出が減るため, 除去する単語を再検討する必要がある．表 3.4 では 3 つのナイーブベイズ学習器を用いたが全て精度が悪かったため, 特徴抽出で次元を削減しすぎると各クラスの特徴が失われ正確に分類できないことがわかった．表 3.5 で特徴量を変化させそのときの精度の比較を行った.F1 スコアで比較すると一番精度が良かったのは 4000 語から 5000 語を特徴抽出したときであるが 1000 語から 5000 語で 2% 程度しか精度は変わっていない. これは研究に用いた文章データに不必要な単語が多く存在していたためである．表 3.6 から単語の出現頻度のみを考慮し単語ベクトルを生成する TF よりその単語の希少性も考慮し重み付けする TFIDF の方が精度が高くなる傾向があることが読み取れる．表 3.7,3.8 より SMOTEENN と ENN を比較すると ENN のほうが精度が高くなっている．一般的に SMOTEENN のほうがデータの不均衡問題を解決でき, 精度が高くなる傾向がある. なので, 本実験では SMOTEENN を用いたバギングではオーバーサンプリングを過剰にしてしまい, 過学習をおこした可能性が考えられる．表 3.10 で,XGBoost を用いて学習器は一般的にハイパーパラメータを調整したほうが精度が良くなるということを実際に確認した. しかし, 表 3.8 と比較すると単一学習器よりも精度が悪いことがわかる. 考えられる原因として, 表 3.9 のハイパーパラメータ最適化におけるパラメータの探索範囲が不適切だったことが挙げられる. 各パラメータの探索範囲を見直し, 精度がより高くなるかどうか確認する必要があると考えた.

第 5 章

まとめ

本研究では, 保険会社のお問い合わせデータを単一学習器, アンサンブル学習や単語ベクトル表現の様々な種類を変えて実験を行い, クレームクラスと非クレームクラスに分類する推定を行った. 実験結果から, 精度を良くするには学習器を変えることよりも単語の特徴量選択などデータの前処理を重要視することが必要であると判断した. 今後の課題として, データセットを特徴エンジニアリングで欠損値を補完することや, 特徴選択で精度を最大限高められるストップワードを作成することなどが挙げられる.

謝辞

本研究において多忙ななか, 貴重な時間を割いて御指導いただいた久留米工業高等専門学校制御情報工学科, 小田幹雄教授及び同研究室のメンバー各人に深く感謝を示します。

参考文献

[1] **OKBIZ.forCommunitySupport,**
“次世代のサポートチャネル”
https://www.okwave.co.jp/business/service/okbiz-cs/option/

[2] 原田 実, 川又 真綱 **“クレーム内容の自動分類”** 言語処理学会第 **14** 回年次大会発表論文集 **pp.293-294,2008**

[3] 上田 修功 **“アンサンブル学習”** 情報処理学会論文誌 **Vol.46 pp11-20,2005**

[4] 吉井 和輝, **Eric Nichols,** 中野 幹生, 青野 雅樹 **“日本語単語ベクトルの構築とその評価”** 情報処理学会 **Vol.2015-NL-221 No.4**

[5] **Dheeraj Mekala, Vivek Gupta, Bhargavi Paranjape, Harish Karnick**
“SCDV:Sparse Composite Document Vectors using soft clustering over distributional representations”
https://arxiv.org/abs/1612.06778

[6] 斎藤 康毅 **“ゼロから作る Deep Learning2** 自然言語処理編**”** オライリー・ジャパン **2018**

[7] 寺田 学, 辻 真吾, 鈴木 たかのり, 福島 真太郎
“Python によるあたらしいデータ分析の教科書**”** 翔泳社 **2018**

[8] **Andreas C. Muller, Sarah Guido “Python** ではじめる機械学習 **-scikit-learn** で学ぶ特徴量エンジニアリングと機械学習の基礎**-” 2018**

付録 A

プログラム

本研究で作成した以下のプログラムを添付する.

- **main.py** 全ての処理を記述したファイル
- **model.py** 実験で用いる学習器についてのファイル
- **scdv_check.py** ストップワードやグラフ作成,**SCDV** などの前処理とその他についてのファイル

Listing A.1. main.py

```
1:  # データの前処理はscdv_cheack.py
2:  # 学習からハイパーパラメータ最適はmodel.py
3:  import argparse
4:  import numpy as np
5:  from scdv_check import SparseCompositeDocumentVectors, build_word2vec,
        plot_scatter, get_resample, delete_stopword
6:  from model import default_voting,default_xgb_object,xgb_object,adaboost
7:  import optuna
8:  import pandas as pd
9:
10: # の引数Word2Vec
11: def parse_args():
12:     parser = argparse.ArgumentParser(
13:         description="とのパラメータの設定Word2VecSCDV"
14:     )
15:     parser.add_argument(
16:         '--embedding_dim', type=int, default=200    #の中間数のノード数NN100推奨
                -500
17:     )
18:     parser.add_argument(
19:         '--min_count', type=int, default=0
20:     )
21:     parser.add_argument(
22:         '--window_size', type=int, default=5
23:     )
24:     parser.add_argument(
25:         '--sg', type=int, default=1  #CBoW =0 , Skip-gram =1
26:     )
27:     parser.add_argument(
28:         '--num_clusters', type=int, default=2
29:     )
30:     parser.add_argument(
31:         '--pname1', type=str, default="gmm_cluster.pkl"
32:     )
33:     parser.add_argument(
34:         '--pname2', type=str, default="gmm_prob_cluster.pkl"
35:     )
36:
37:     return parser.parse_args()
38:
39:
40: def main(args):
41:     # X:Dataset Y:Label
42:     with open("C:/Users/adisonax/Desktop/sotuken env/windowsenv/1/
            trainwakati_1data.txt") as f:
43:         X_train = f.read().split("\n")[:-1]
44:     with open("C:/Users/adisonax/Desktop/sotuken env/windowsenv/1/
            train_1label.txt") as f:
45:         Y_train = f.read().split("\n")[:-1]
46:     with open("C:/Users/adisonax/Desktop/sotuken env/windowsenv/1/
            testwakati_1data.txt") as f:
47:         X_test = f.read().split("\n")[:-1]
48:     with open("C:/Users/adisonax/Desktop/sotuken env/windowsenv/1/
            test_1label.txt") as f:
49:         Y_test = f.read().split("\n")[:-1]
50:     Y_train = [int(i) for i in Y_train]
51:     Y_test = [int(i) for i in Y_test]
52:
53:     # ここから側の処理scdv_check
54:     # を作るWord2Vec は次元 embedding_dim
55:     model = build_word2vec(
56:             X_train,
57:             args.embedding_dim,
58:             args.min_count,
59:             args.window_size,
```

```
60:                 args.sg
61:         )
62:         vec = SparseCompositeDocumentVectors(
63:                 model,
64:                 args.num_clusters,
65:                 args.embedding_dim,
66:                 args.pname1,
67:                 args.pname2
68:         )
69:         # 確率重み付き単語ベクトルを求める
70:         vec.get_probability_word_vectors(X_train)
71:         # 訓練データからを求めるSCDV
72:         train_gwbowv = vec.make_gwbowv(X_train)
73:         # テストデータからを求めるSCDV
74:         test_gwbowv = vec.make_gwbowv(X_test)
75:         # ここまで
76:         X_res_train, Y_res_train = get_resample(train_gwbowv, Y_train)
77:
78:
79:         optuna.logging.set_verbosity(optuna.logging.WARNING)
80:         study = optuna.create_study()
81:         study.optimize(lambda trial: xgb_object(trial,X_res_train,
                Y_res_train,test_gwbowv,Y_test), n_trials=100)
82:
83:         print(study.best_value)
84:         print(study.best_params)
85:         df = study.trials_dataframe()
86:         df.to_csv('C:/Users/adisonax/Desktop/tuning_param.tsv', sep='\t')
87:
88:         default_voting(X_res_train, Y_res_train, test_gwbowv, Y_test)
89:
90:     if __name__ == '__main__':
91:         main(parse_args())
```

Listing A.2. scdv_check.py

```
1:  # データの前処理プログラム
2:  # 除去stopword + で分散処理SCDV + 散布図出力
3:  import re
4:  import logging
5:  import pickle
6:  import numpy as np
7:  from gensim.models.word2vec import Word2Vec
8:  from tqdm import tqdm
9:  from sklearn.mixture import GaussianMixture
10: from sklearn.feature_extraction.text import TfidfVectorizer
11: import matplotlib.pyplot as plt
12: from sklearn.manifold import TSNE
13: from imblearn.combine import SMOTEENN
14: from imblearn.under_sampling import EditedNearestNeighbours
15: from imblearn.over_sampling import SMOTE
16:
17: # ストップワード除去
18: def delete_stopword(X_train):
19:     line2, new_X_train = [], []
20:     print("ストップワード除去")
21:     for line in tqdm(X_train):
22:         vocab = line.split(" ")
23:         for vocab2 in vocab:
24:             vocab2 = re.sub(r'。()[−、[]・※XXXXX()、「」]', "", vocab2)
25:             line2.append(vocab2)
26:         new_X_train.append(' '.join(line2))
27:         line2.clear()
28:     return new_X_train
29:
30: # を使い、単語をベクトル化したものを次元グラフにプロットTSNE2
31: def plot_scatter(X, Y):
32:     print("グラフ出力中.....")
33:     X_reduced = TSNE(n_components=2).fit_transform(X)
34:     plt.scatter(X_reduced[:,0], X_reduced[:,1], c=Y)
35:     plt.savefig('C:/Users/adisonax/Desktop/tsne.png')
36:     print("保存完了")
37:
38: # でデータの不均衡を調節SMOTEENN
39: def get_resample(X, Y):
40:     print("不均衡データの調節開始.....")
41:     #sme = SMOTEENN()
42:     enn = EditedNearestNeighbours()
43:     X_res, Y_res = enn.fit_resample(X, Y)
44:     print("調節終了")
45:     return (X_res, Y_res)
46:
47: def build_word2vec(sentences, embedding_dim, min_count, window_size, sg
        ):
48:     logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message
          )s', level=logging.INFO)
49:     # モデルを作る
50:     model = Word2Vec(sentences, size=embedding_dim, min_count=min_count
          , window=window_size, sg=sg)
51:     return model
52:
53: class SparseCompositeDocumentVectors:
54:     def __init__(self, model, num_clusters, embedding_dim, pname1,
            pname2):
55:         self.min_no = 0
56:         self.max_no = 0
57:         self.prob_wordvecs = {}
58:         self.model = model
59:         self.word_vectors = model.wv.syn0
60:         self.num_clusters = num_clusters
61:         self.num_features = embedding_dim
```

```
62:             self.pname1 = pname1
63:             self.pname2 = pname2
64:
65:         def cluster_GMM(self):
66:             # Initalize a GMM object and use it for clustering.
67:             clf = GaussianMixture(
68:                 n_components=self.num_clusters,
69:                 covariance_type="tied",
70:                 init_params="kmeans",
71:                 max_iter=50
72:             )
73:             # Get cluster assignments.
74:             clf.fit(self.word_vectors)
75:             idx = clf.predict(self.word_vectors)
76:             print("Clustering Done...")
77:             # Get probabilities of cluster assignments.
78:             idx_proba = clf.predict_proba(self.word_vectors)
79:             # Dump cluster assignments and probability of cluster
                    assignments.
80:             pickle.dump(idx, open(self.pname1, "wb"))
81:             print("Cluster Assignments Saved...")
82:             pickle.dump(idx_proba, open(self.pname2, "wb"))
83:             print("Probabilities of Cluster Assignments saved...")
84:             return (idx, idx_proba)
85:
86:         def read_GMM(self):
87:             # Loads cluster assignments and probability of cluster
                    assignments.
88:             idx = pickle.load(open(self.pname1, "rb"))
89:             idx_proba = pickle.load(open(self.pname2, "rb"))
90:             print("Cluster Model Loaded...")
91:             return (idx, idx_proba)
92:
93:         def get_probability_word_vectors(self, corpus):
94:             """
95:             corpus: list of lists of tokens
96:             """
97:             global tfidfmatrix_traindata
98:             # This function computes probability word-cluster vectors.
99:             idx, idx_proba = self.cluster_GMM()
100:
101:            # Create a Word / Index dictionary, mapping each vocabulary
                    word
102:            # to a cluster number
103:            word_centroid_map = dict(zip(self.model.wv.index2word, idx))
104:            # Create Word / Probability of cluster assignment dictionary,
                    mapping
105:            # each vocabulary word to list of probabilities of cluster
                    assignments.
106:            word_centroid_prob_map = dict(zip(self.model.wv.index2word,
                    idx_proba))
107:
108:            # Comoputing tf-idf values
109:            tfv = TfidfVectorizer(dtype=np.float32, token_pattern="(?u)\\b
                    \\w+\\b", norm='l2')
110:            # transform corpus to get tfidf value
111:            corpus = [" ".join(data) for data in corpus]
112:            tfidfmatrix_traindata = tfv.fit_transform(corpus)
113:            featurenames = tfv.get_feature_names()
114:            idf = tfv._tfidf.idf_
115:            # Creating a dictionary with word mapped to its idf value
116:            print("Creating word-idf dictionary for dataset...")
117:            word_idf_dict = {}
118:            for pair in zip(featurenames, idf):
119:                word_idf_dict[pair[0]] = pair[1]
120:
```

```
121:             for word in word_centroid_map:
122:                 self.prob_wordvecs[word] = np.zeros(self.num_clusters *
                         self.num_features, dtype="float32")
123:                 for index in range(self.num_clusters):
124:                     try:
125:                         self.prob_wordvecs[word][index*self.num_features:(
                                 index+1)*self.num_features] = \
126:                             self.model[word] * word_centroid_prob_map[word
                                     ][index] * word_idf_dict[word]
127:                     except:
128:                         continue
129:             self.word_centroid_map = word_centroid_map
130:
131:         def create_cluster_vector_and_gwbowv(self, tokens, flag):
132:             # This function computes SDV feature vectors.
133:             bag_of_centroids = np.zeros(self.num_clusters * self.
                     num_features, dtype="float32")
134:             for token in tokens:
135:                 try:
136:                     temp = self.word_centroid_map[token]
137:                 except:
138:                     continue
139:                 bag_of_centroids += self.prob_wordvecs[token]
140:             norm = np.sqrt(np.einsum('...i,...i', bag_of_centroids,
                     bag_of_centroids))
141:             if norm != 0:
142:                 bag_of_centroids /= norm
143:             # To make feature vector sparse, make note of minimum and
                     maximum values.
144:             if flag:
145:                 self.min_no += min(bag_of_centroids)
146:                 self.max_no += max(bag_of_centroids)
147:             return bag_of_centroids
148:
149:         def plain_word2vec_document_vectors(self, tokens):
150:             bag_of_centroids = np.zeros(self.num_features, dtype="float32")
151:             for token in tokens:
152:                 try:
153:                     temp = self.model[token]
154:                 except:
155:                     continue
156:                 bag_of_centroids += temp
157:
158:             bag_of_centroids = bag_of_centroids / len(tokens)
159:             return bag_of_centroids
160:
161:         def make_gwbowv(self, corpus, train=True):
162:             # gwbowv is a matrix which contains normalized document vectors
                     .
163:             gwbowv = np.zeros((len(corpus), self.num_clusters*self.
                     num_features)).astype(np.float32)
164:             cnt = 0
165:             for tokens in tqdm(corpus):
166:                 gwbowv[cnt] = self.create_cluster_vector_and_gwbowv(tokens,
                          train)
167:                 cnt += 1
168:             return gwbowv
169:
170:         def make_word2vec(self, corpus, model):
171:             self.model = model
172:             w2docv = np.zeros((len(corpus), self.num_features)).astype(np.
                     float32)
173:             cnt = 0
174:             for tokens in tqdm(corpus):
175:                 w2docv[cnt] = self.plain_word2vec_document_vectors(tokens)
176:                 cnt += 1
```

```
177:            return w2docv
178:
179:        def get_word2vec_document_vector(self, tokens):
180:            # tokens: list of tokens
181:            return self.plain_word2vec_document_vectors(tokens)
182:
183:        def dump_gwbowv(self, gwbowv, path="gwbowv_matrix.npy", percentage
                =0.04):
184:            # Set the threshold percentage for making it sparse.
185:            min_no = self.min_no*1.0/gwbowv.shape[0]
186:            max_no = self.max_no*1.0/gwbowv.shape[0]
187:            print("Average min: ", min_no)
188:            print("Average max: ", max_no)
189:            thres = (abs(max_no) + abs(min_no))/2
190:            thres = thres * percentage
191:            # Make values of matrices which are less than threshold to zero
                    .
192:            temp = abs(gwbowv) < thres
193:            gwbowv[temp] = 0
194:            np.save(path, gwbowv)
195:            print("SDV created and dumped...")
196:
197:        def load_matrix(self, name):
198:            return np.load(name)
```

Listing A.3. model.py

```
import optuna
from sklearn.ensemble import RandomForestClassifier, VotingClassifier, AdaBoostClassifier
from sklearn.naive_bayes import BernoulliNB, GaussianNB, MultinomialNB
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
from sklearn.metrics import confusion_matrix,
    precision_recall_fscore_support


#  最適化XGBoost
def xgb_object(trial, X_tra, Y_tra, X_tes, Y_tes):
    #パラメータ探索範囲
    learning_rate = trial.suggest_loguniform('learning_rate', 0.01, 0.5)
    max_depth = trial.suggest_int('max_depth', 4, 12)
    min_child_weight = trial.suggest_int('min_child_weight', 1, 6)
    subsample = trial.suggest_loguniform('subsample', 0.01, 1)
    colsample_bytree =trial.suggest_loguniform('colsample_bytree', 0.01, 1)
    lambda1 = trial.suggest_loguniform('reg_lambda', 1, 5)
    gamma = trial.suggest_loguniform('gamma', 0.001, 5)

    xgb = XGBClassifier(eta=learning_rate, gamma=gamma, max_depth=max_depth,
                        min_child_weight=min_child_weight,
                        subsample=subsample, reg_lambda=lambda1,
                        colsample_bytree=colsample_bytree,
                        )
    xgb.fit(X_tra, Y_tra)
    y_pred = xgb.predict(X_tes)

    print("==========================================================================")
    print("ACC={:.4f}".format(100*accuracy_score(Y_tes, y_pred)))
    print("再現率R={:.4f}".format(100*recall_score(Y_tes, y_pred)))
    print("適合率P={:.4f}".format(100*precision_score(Y_tes, y_pred)))
    print("スコアF1={:.4f}".format(100*f1_score(Y_tes, y_pred)))
    print(confusion_matrix(Y_tes, y_pred))
    print("==========================================================================")
    #スコアで最適化F1 は現在最小化しか対応していないためoptuna1から引くことで最大化.0
    return (1.0 - f1_score(Y_tes, y_pred))

# デフォルトパラメータXGBoost
def default_xgb_object(X_tra,Y_tra,X_tes,Y_tes):
    xgb = XGBClassifier()
    xgb.fit(X_tra, Y_tra)
    y_pred = xgb.predict(X_tes)

    print("==========デフォルトパラメータXGBoost==========")
    print("ACC={:.4f}".format(100*accuracy_score(Y_tes, y_pred)))
    print("再現率R={:.4f}".format(100*recall_score(Y_tes, y_pred)))
    print("適合率P={:.4f}".format(100*precision_score(Y_tes, y_pred)))
    print("スコアF1={:.4f}".format(100*f1_score(Y_tes, y_pred)))
    print(confusion_matrix(Y_tes, y_pred))
    print("==========================================================================")
```

```
        ")
54:
55: #Voting ロジスティック回帰 ランダムフォレスト 近傍法 ナイーブベイズKSVM
56: def default_voting(X_tra,Y_tra,X_tes,Y_tes):
57:     clf1 = LogisticRegression(solver='lbfgs', max_iter=10000)
58:     clf2 = RandomForestClassifier()
59:     clf3 = KNeighborsClassifier()
60:     clf4 = SVC(probability=True, gamma='scale')
61:     clf5 = BernoulliNB()
62:
63:     eclf = VotingClassifier(estimators=[('LR', clf1), ('RF', clf2),
64:                                         ('KNN', clf3), ('SVM', clf4),
65:                                         ('NB', clf5)], voting='hard')
66:     eclf.fit(X_tra, Y_tra)
67:
68:     print("デフォルトパラメータVoting")
69:     for name, estimator in eclf.named_estimators_.items():
70:         print(name, ":{:.4f}".format(100*accuracy_score(Y_tes,
                estimator.predict(X_tes))))
71:         print("再現率R = {:.4f}".format(100*recall_score(Y_tes, estimator
                .predict(X_tes))))
72:         print("適合率P = {:.4f}".format(100*precision_score(Y_tes,
                estimator.predict(X_tes))))
73:         print("スコアF1= {:.4f}".format(100*f1_score(Y_tes, estimator.
                predict(X_tes))))
74:         print(confusion_matrix(Y_tes, estimator.predict(X_tes)))
75:
76:     # 総合評価
77:     print("正答率 = {:.4f}".format(100*accuracy_score(Y_tes, eclf.predict
            (X_tes))))
78:     print("再現率R = {:.4f}".format(100*recall_score(Y_tes, eclf.predict(
            X_tes))))
79:     print("適合率P = {:.4f}".format(100*precision_score(Y_tes, eclf.
            predict(X_tes))))
80:     print("スコア
            F1 = {:.4f}".format(100*f1_score(Y_tes, eclf.predict(X_tes))))
81:     print(confusion_matrix(Y_tes, eclf.predict(X_tes)))
82:
83:
84: def adaboost(X_tra,Y_tra,X_tes,Y_tes):
85:     adb = AdaBoostClassifier(n_estimators=500)
86:     adb.fit(X_tra, Y_tra)
87:     y_pred = adb.predict(X_tes)
88:
89:     print("==========================================デフォルトパラメータ
            AdaBoost==========================================")
90:     print("ACC={:.4f}".format(100*accuracy_score(Y_tes, y_pred)))
91:     print("再現率R={:.4f}".format(100*recall_score(Y_tes, y_pred)))
92:     print("適合率P={:.4f}".format(100*precision_score(Y_tes, y_pred)))
93:     print("スコアF1={:.4f}".format(100*f1_score(Y_tes, y_pred)))
94:     print(confusion_matrix(Y_tes, y_pred))
95:     print("
            ==========================================================================
            ")
```