厚生労働科学研究費補助金
（政策科学総合研究事業（統計情報総合研究事業））

死因統計の精度及び効率性の向上に資する機械学習の検討に関する研究

令和３年度　総括・分担研究報告書

研究代表者　今井　健

（東京大学 大学院医学系研究科）

令和４（２０２２）年３月

目　　次

厚生労働科学研究費補助金(政策科学総合研究事業（統計情報総合研究事業））
総括研究報告書（令和３年度）

死因統計の精度及び効率性の向上に資する機械学習の検討に関する研究

研究代表者　今井　健　（東京大学大学院医学系研究科 准教授）

研究要旨

人口動態調査は国勢調査と並ぶ国の主要統計で公衆衛生施策の中心的資料である。本研究は原死因確定に関する調査を行い、我が国での原死因データ収集における課題を抽出し、ICD-11 における死亡診断書や死亡統計ルールの動向を調査すると共に、原死因確定作業に対する機械学習の適用可能性について調査・検討を行うことを目的とした。まず、死亡票の実データを対象に、文字列処理と自動 ICD-10 コード付与を行った上で、オートコーディングツール IRIS を適用し、約 80%の死亡票に対し、仮原死因を確定した。またこの付帯情報による仮原死因の変更の有無、外因や母側病態のコード追加の有無の割合について明らかにすると共に、機械学習による支援ターゲットとして「何らかの付帯情報による仮原死因の変更の有無」が有効であることを明らかにした。また機械学習により I 欄 II 蘭病名と付帯情報からこの変更の有無を予測し、確信度と共に導出する2値分類器を複数のアルゴリズムで開発し、何らかの付帯情報が存在する死亡票を対象に **Accuracy 95%, ROC-AUC 0.953, PR-AUC 0.857** という非常に高い分類精度を実現した。機械学習の性質上、100%の精度実現は困難であるが、この分類器は付帯情報の影響による原死因コードの変更の有無のみならず、確信度も出力できることから、従来の人手による確認作業の正確性・効率性向上に大いに寄与する支援ツールとなると期待される。

将来的な ICD-11 導入後の原死因確定ツールとしては、Iris、WHO cause of death identification tool、国内のオートコーディングツールの更新の3種が考えられるが、本支援手法はこれらのどのオートコーディングツールとも組み合わせて利用することが可能であり、我が国における ICD-11 適用後においても死因統計の精度・効率性向上に資する極めて有力な手法と考えられる。

研究分担者

香川璃奈

筑波大学医学医療系 講師

研究協力者

大井川仁美

東京大学大学院医学系研究科客員研究員

大江和彦

東京大学大学院医学系研究科 教授

今村知明

奈良県立医科大学公衆衛生学講座 教授

A．研究目的

我が国において人口動態調査は国勢調査と並ぶ国の基幹統計であり、中でも死因統計は最も重要な情報の一つである。今後 ICD-11 を国内適用するにあたっては原死因データを適切に収集・分析し、国際比較可能なデータを提供することが求められている。レセプトや現在普及が進む電子カルテでは標準病名の採用が進められているが、人口動態調査の死因は自由入力病名が元となっており完全な自動集計は困難である。また我が国では高齢化が進み死亡者数の増加が見込まれるこ

とから、より正確で効率の高いデータ収集の方法の検討が求められている。

そこで、本研究は、原死因確定に関する調査を行い、我が国での原死因データ収集における課題を抽出し、ICD-11 における死亡診断書や死亡統計ルールの動向を調査すると共に、原死因確定作業に対する機械学習の適用可能性について調査・検討を行うことを目的としている。今年度は昨年までの成果を元に、追加提供を受けたデータと合わせ、各種機械学習アルゴリズムによる機械学習実験を行い、適用可能性分析を行った。またその他の調査項目についてもアップデートを行った。

Ｂ．研究方法

B-1）原死因確定プロセスにおける課題の抽出

原死因データ収集における課題については、既に本研究班メンバーらは平成 30 年度厚生労働統計協会調査研究委託事業において、ヒアリング調査・関連資料分析などを通じ基礎調査を行ってきている。本研究ではこれを発展させ、より詳細な分析を行うため、厚生労働省関係者へのヒアリングと共に、統計法第 33 条に基づく目的外利用申請によって平成 27 年～令和 2 年の死亡票・死亡個票データの提供を受け、実データを元にした分析を行った。また統計法 22 条により、このうち一部について、厚生労働省にて人手チェックに回った調査票情報の提供を受け、このサンプリング結果に対する分析と合わせることで、原死因確定の流れを明らかにした。

本年度は、昨年度までのデータに加え、令和 1,2 年度の死亡票・死亡個票のデータ、並びに人手チェックに回った調査票情報のデータ(300 件)の追加提供を受けたため、昨年度までの結果をアップデートし、原死因確定プロセス(最終版)の分析を行った。

B-2）機械学習の適用可能性調査

既に昨年度までの研究により、機械学習の効率的な適用のためには、「何らかの付帯情報があり人手による確認処理に回っている、全体の約 35%程度の死亡票」に対し、**付帯情報の内容を考慮した上で「オートコーディングシステムが付与した仮原死因」を変更するべきか否か、を高精度に予測する**ことが重要であることが判明している。しかし、本研究では厚生労働省内部で使用されているオートコーディングシステムを用いることができない制限があり、自らこのシステムを模す必要がある。そのため様々な国で広く採用されているフリーの死亡票オートコーディングシステムである IRIS を用いることとし、以下のような 5 つのステップにて行った。詳細については、**「別添資料1：本研究で構築したシステムの詳細」**を参照されたい。

STEP1）死亡票・死亡個票からの IRIS 入力用データの作成

死亡票・死亡個票実データ(平成 27～令和 2 年、約 800 万件)に対し、各種の前処理を行った上で、死亡個票中の自由入力病名を各種の文字列処理と標準病名マスターを利用して自動 ICD-10 コーディングを行うシステムを開発した。昨年度までに要素技術は既に開発済みであるが、本年度はこれを 6 年分の全データへ適用すると共に、残りの処理に用いた。

STEP2) IRIS での仮原死因確定処理

次に、死亡個票中の全病名に ICD-10 コードが振られたものについて IRIS に入力し、仮原死因コードを決定すると共に、確定原死因コードと比較を行った。その際に、IRIS と国内のコーディングルールの差異を吸収する処理も行った。

STEP3) IRIS 処理結果の解析

IRIS 処理の結果、何らかの修正処理が必要なケース、不要なケースに分類し、その内容について集計を行うと共に、元の「突合 DB」と合わせてこの後の解析用の「統合テーブル」を作成した。

STEP4）機械学習用データセットの作成

次に、統合テーブルの情報を元に、複数のアルゴリズムでの機械学習用データの作成を行った。想定した機械学習アルゴリズムは3種類である

・XGBoost-Simple が最も単純なベースラインで、（共通ベクトル） ＝「性別・年齢、病名の ICD コード、付帯情報の項目の有無」 だけで予測を行う。

・XGBoost-Embed は共通ベクトルに、付帯情報の内容を各手法でベクトル化したものを結合したものを使って予測を行う。

・BERT は、付帯情報の内容を BERT 言語モデルに通した出力を、共通ベクトルと上位層で統合して予測を行う DNN (Deep Neural Network) である。

ここではそれぞれのアルゴリズムに合わせた機械学習用データセットの作成を行った。

(1) [XGBoost-Simple]

I 欄 II 蘭病名と付帯情報の有無(2 値) を用いた勾配ブースティング回帰木(XGBoost)

- I 蘭 II 蘭病名については、使用された ICD10 コード、付帯情報については 22 種の項目に対する記載の有無(22 次元)、これに年齢・性別を加えたベクトルを用いた。
- 最も単純なものであるが、既に昨年度研究において 90.3%の正解率を実現しており、本研究ではその他の手法の比較におけるベースラインとなるものである。
- 後の比較では“BASELINE”として参照されている。

(2) [XGBoost-Embed]

(1) のベクトルに加え、付帯情報の文字列の内容を様々な手法で分散表現（ベクトル）に埋め込んだ(Embedding)結果を統合したベクトルに対する勾配ブースティング回帰木(XGBoost)

- (1)は付帯情報について項目の記載の「有無」だけであったが、内容を加味するため、各項目の記載文字列を分散表現に変換し、(1)のベクトルに加えたものである。
- 付帯情報に対する分散表現の獲得手法により、以下に細分化される。

- TF・IDF

  文章中に出現した単語の重要度を TF (Term Frequency), IDF(Inverse Document Frequency)から算出する古典的な方法である。出現頻度 100 以上の単語だけを用いた。

- LSI

  LSI (Latent Semantic Index) ではトピックという潜在的変数を仮定する。各文書の BOW(Bag of Words) あるいは TF・IDF ベクトルを行とした、文書数× 単語数の行列を特異値分解することで、文書数×トピック数に次元削減するものである。トピックを間に挟む分、TF・IDF よりも類義語多義語にある程度対応できるとされている。

- WORD2VEC

  Word2Vec は文書中の「単語」の意味を分散表現として自動獲得するためのニューラルネットワークを用いた学習方法である。本研究ではある単語の前後の語から対象単語を復元するタスクにより学習する手法 (CBOW) を用い、得られた付帯情報文字列中の各単語の分散表現の平均を「付帯情報文字列全体」の分散表現とした。

- DOC2VEC (PV-DM)
- DOC2VEC (PV-DBOW)

  Doc2Vec は Word2Vec の手法を文章に拡張した方法である。Word2Vec により得られた単語の分散表現を平均する方法とは異なり、文章自体の分散表現を直接得る手法である。

  尚、DOC2VEC については本研究分担研究者の香川璃奈が担当した。

  詳細は、本年度分担研究報告書「死亡に関わる調査票情報提供に基づいた ICD10 コード自動付与ツールの作成（香川璃奈）」を参照されたい。

(3) [BERT]

BERT とは 2018 年に Google から発表された、大量の文書リソース(コーパス)から汎用言語モデルを自動獲得するための手法である。このモデルはさらにファインチューニングを行うことによって様々な自然言語処理タスクに汎用的に用いることができ、従来の自然言語処理タスクの多くにおいて SOTA (State of the Art) を達成したことで近年大きな注目を集めている。一般的な使い方としては、何らかのコーパスにて学習された BERT モデルを(必要に応じてさらに追加で事前学習を行い)、その上で特定のタスクを解くための Deep Neural Network アーキテクチャに組み込んでファインチューニングするという方法が用いられる。本研究でもこれら一般的な方法に則り、付帯情報の中の文字列情報を BERT モデルに通した結果と (1)の情報を全結合層で統合した深層学習モデルを採用した。

- (2) の手法とは異なり、分散表現の獲得には日本語 Wikipedia を元にした BERT 言語モデルを採用し、(1)のベクトルと全結合層で統合したモデルである。

STEP5) 各手法での仮原死因変更有無予測モデルの学習

我が国の原死因確定プロセスにおいては、付帯情報がない場合は、基本的にオートコーディングツールで原死因コードがそのまま確定されるため、これらのモデルに通す必要がない。従って、**何らかの付帯情報が存在する死亡票のみ**を対象に STEP4 での各手法を用いて、**「I 欄 II 欄病名と付帯情報から仮原死因の変更有無を予測し、確信度と共に導出する分類モデル」**を学習した。

また以上の一連の処理は自動化し、Docker 並びに仮想マシンにおいて実行可能なシステムとして実装した。

B-3) ICD-11 における死亡診断書や死亡統計ルールの動向調査

我が国の現行の死亡統計では ICD-10 を元にした WHO による原死因選択ルールが適用されている。しかし 2018 年 6 月に WHO が ICD-11 をリリースした今、ICD-11 における死亡統計の動向は今後の我が国への ICD-11 適用に際し重要である。本年度は昨年度に引き続き WHO 並びに日本 WHO-FIC 協力センターの関係者へのヒアリング、また 10 月に開催された WHO-FIC 会議 ITC (Informatics and Terminology Committee) などへの参加によってこの動向調査を行った。

倫理面への配慮

本研究では個人情報や動物愛護に関わる調査・実験は行わない。但し研究の遂行に当たっては、各種法令や「人を対象とする医学系研究に関する倫理指針」を含め各種倫理指針を遵守した。

C. 研究結果

C-1) 原死因確定プロセスにおける課題

まず本研究で関係者へのヒアリングを通じ、原死因確定プロセスにおける課題として、大きく次の 2 つが挙げられた。

(1) オートコーディングシステムで原死因がルールベースで決定できない事例

- 死亡票の I・II 欄傷病名が自由入力であるため、辞書マッチングで ICD-10 コードが付与できないことがある
- 原死因選択ルールに合致しない、疑義がある(「老衰」が年齢と合っていない、希少疾患である等)

(2) 何らかの付帯情報が存在する場合

- 付帯情報とは、I 欄 II 欄病名以外の何らかの補足的情報(手術解剖の有無と所見、外因死の追加事項、生後 1 年未満死の追加事項、その他付言すべきことがら等)のことである。
- 付帯情報が存在する場合は必ず人手での確認処理に回され、必要があればオートコーディング

システムが出力した仮の原死因コードを修正している。

- これには以下のような多様な事例が存在する。
  - I 欄(ア)に「肺炎」、手術欄に「胃悪性腫瘍切除術・1 週間前」とある。
    →本来 I 欄(イ) に「胃癌」と書くべきとみなしてこれを選択する。
  - I 欄(ア)で「損傷」、手段・状況欄で、自殺，飛び降り、あるいは交通事故とわかるとそれを優先 (外因等)。
  - 「細菌性肺炎」とあるが解剖欄の情報で菌の種類が分かる場合詳細化する。
  - 「飛び降り自殺」とあるが、I 欄内で産後うつによる影響であると分かると妊産婦死亡のフラグを付与する。

次に、原死因確定プロセスの流れを明らかにするため、統計法33 条に基づく目的外利用申請によって提供を受けた死亡票・死亡個票の実データを用い、両者を突合した。以下これを「**突合死亡票データベース (以下突合 DB)**」と呼ぶ。突合方法の詳細については**「令和元年度総括研究報告書」**を参照されたい。

さらに、本研究では統計法22条に基づき、上記の一部について、厚生労働省にて人手確認に回った調査票情報の提供を受けた。これは死亡票の 1 ヶ月分の集計データに対し、人手確認に回った原因を集計したもの(表1) とさらにその中から300 件をランダム抽出し、実際に人手確認でどのような対処が行われたかを集計したデータ (表2) である。またこれらの結果の分析に基づいて推計し、昨年までの結果をアップデートした原死因確定プロセスの流れ (最終版)を図1に示す。

			コーディング疑義	
			あり	なし
			16,631	96,308
付帯情報	あり	32.40%	12,973	23,592
	なし	67.60%	3,658	72,716

表1：人手確認に回った死亡票の内訳

まず、表1の抽出データは全件で 112,939 件あり、「コーディング疑義」とは厚生労働省内のオートコーディングツール(病名に ICD-10 コードを付与し、複数の傷病名から原死因コードを確定するツール)にて、コード化プロセス中に疑義がある、とされたものである。また「付帯情報」とは死亡票死亡個票の I 欄 II 欄病名以外の何らかの補足的情報のことである。表中薄灰色の部分(コーディング疑義あり、もしくは付帯情報あり) の部分が**人手確認に回ったもので、合わせて 40,223 件 (35.6%) 存在した**。毎月約 4 万件程度が人手確認されていることになる。一方、残りの約 64% はオートコーディングシステムが決定した原死因コードがそのまま確定される。

次に人手で確認されたもの(C リスト)からランダムに 300 件サンプリングされたデータに対する「対処内容の内訳」を表2に示す。

		原死因コード修正	
		あり	なし
追加コード(外因・母側病態) の付与	あり	11 (3.6%)	36 (12.0%) (うち母側:2)
	なし	30 (10.0%)	223 (74.3%)
	計	41 (13.6%)	259 (76.3%)

表2：人手確認対処内容の内訳

人手確認の後、原死因が変更されたものは 41 件 (13.6%)、変更なしが 259 件 (86.3%) であった。昨年度までは 100 件ランダム抽出したデータの提供を受

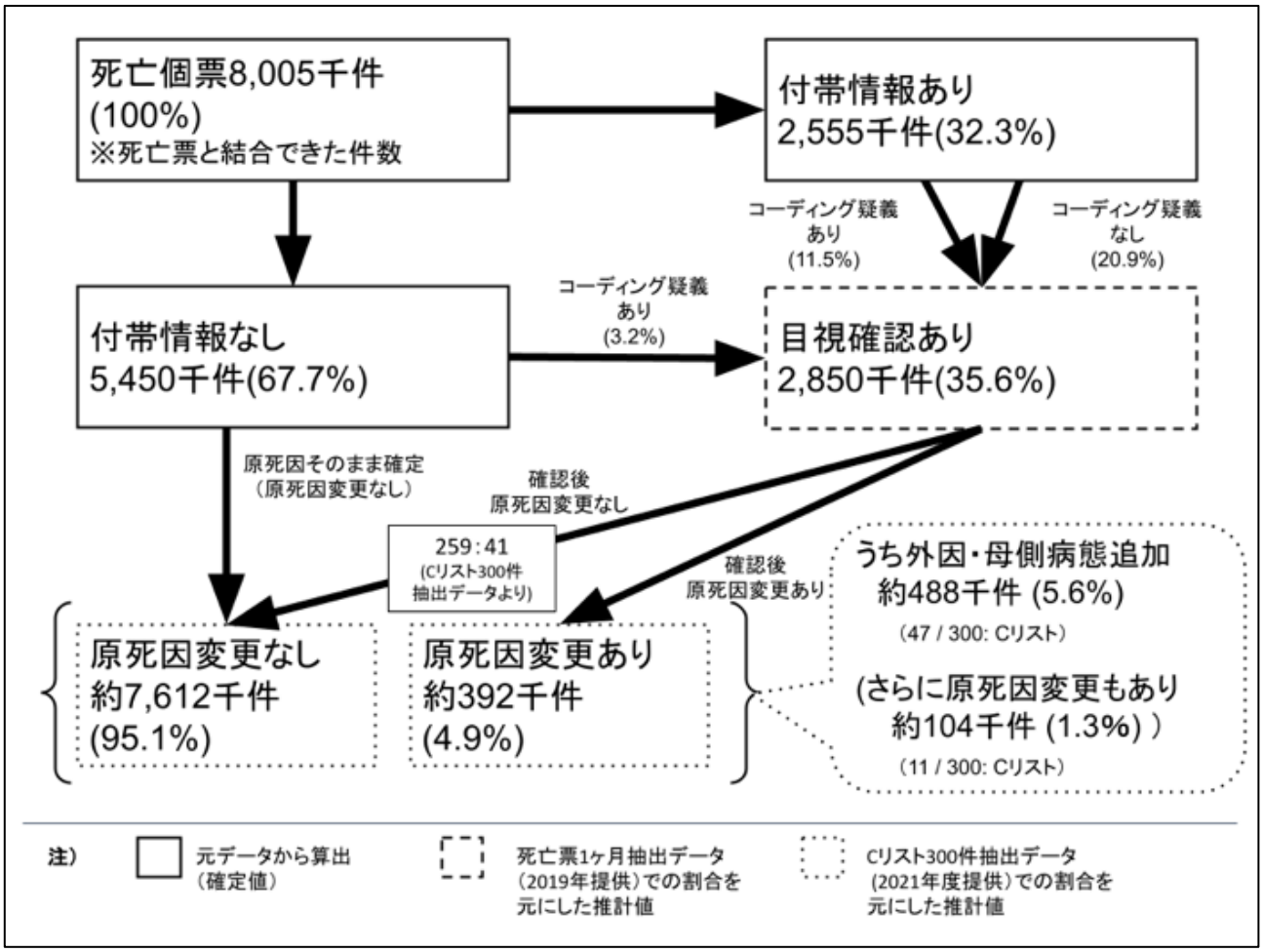


図 1 原死因確定プロセスの流れ（最終版）

け分析していたが、少数であるものの、300 件になったことで統計的信頼性が多少向上している。

以上の分析を元に推計した原死因確定プロセスの流れを図1に示す。図1中の**「実線四角」**は、実際のデータから算出されたもので確定値である。付帯情報があるものは全体の32.3%となっている。

一方、**「破線四角」**は表1の結果からの推定値である。表1の結果から 35.6% が人手確認に回っているため、差し引き 3.2%がコーディング疑義により人手確認に回ったと推計される。

さらに、**「点線四角」**は表2の結果からの推定値である。表2の結果から、人手確認後 259:41 の割合、つまり 300 件中の **13.6%が原死因変更**されている。従ってこの割合を外挿すると、原死因が変更される死亡票は全体の 4.9%ということになる。

以上から、まず **(1)「付帯情報あり」が人手確認の大半を占めており、これを計算機支援する必要があることが判明した**。また、表2から、人手確認されたもののうち、何の対処も必要ない**「原死因コードの修正も追加コードの付与も不要」であるケースは 74.3%** であるので、これを高精度に分離することができれば、人手による作業を大幅に軽減することが可能である。しかし現行のオートコーディングシステムでも追加コードの付与が必要なケースについては自動的に判断できているため、**後は「原死因コードの変更の必要あり／なし」さえ自動的に判断できれば、人手の対処が不要なケースが分離できる**ことになる。人手確認の結果原死因が変更される割合(13.6%)は少なく、大多数は変更ない。従って、**(2) 原死因の変更の有無を高精度に予測する機械学習アルゴリズムの導入が人**

手チェック作業効率化のために有効と考えられた。これは 付帯情報の内容を考慮した上で、原死因の変更あり(13.6%) と 変更なし(86.4%)を分類する2値分類問題と捉えられる。

C-2) 機械学習の適用可能性

上記の結果から、以降は機械学習を用いて「付帯情報があるものを対象とし、その内容から仮原死因の変更の有無を予測するモデル」の開発を行った。本節に関係する各種の詳細な結果は、本統括報告書末尾に「**表 A～F**」としてまとめてある。以降ではこれらを適宜参照、あるいは抜粋しながら述べる。

- **表 A**:突合 DB の項目・記載内容の詳細
- **表 B**:突合 DB(ユニークキーのみ)の仕様とレコード件数
- **表 C**:IRIS 処理結果と死亡票データを合わせた「統合テーブル」仕様
- **表 D**:各処理段階の件数詳細
- **表 E**:BERT モデルを用いた各学習手法の予測精度一覧
- **表 F**:機械学習の結果一覧

STEP1) 死亡票・死亡個票からの IRIS 入力用データの作成

突合 DB とその詳細

平成 27～令和 2 年の 6 年間の死亡票・死亡個票の突合 DB の項目と記載内容の詳細について**表 A** に示す。突合 DB は 63 列からなるテーブルで、8,004,708 件存在した。各項目の詳細仕様は表 A に示す通りである。

突合 DB(ユニークキーのみ)

突合 DB は、「届出地、事件簿番号、処理年月」の組み合わせをキーとして行ったが、このキーが複数回存在するものが 92,768 件存在した。これを除いた結果、突合 DB(ユニークキーのみ)は 7,911,940 件となった。この複数存在するキーはヒアリングの結果、早期提出に起因するものとのことであった。この重複削除処理の結果について各年の件数と突合 DB に対する割合を本報告書末尾の**表 D に示す。**表 D の L10 を見ると、重複キーの削除によって、元の突合 DB の 98.84% の件数となっているが、微減でありほぼ影響はない。

備考欄の前処理

死亡票・死亡個票の各欄からは、入力時の文字数制限により、入り切らない文字列が備考欄に溢れて記入されることがある。この「備考欄に溢れた文字列」を多くの正規表現ルールにより、元の然るべき項目へ可能な限り復元して結合する処理を行った。

内容は、I 欄 II 欄病名とそれぞれの期間、解剖・手術の詳細、傷害が発生したところ・手段及び状況、その他付言すべき事柄、生後 1 年未満での病死に関する詳細、である。元の項目に復元できなかった文字列のみを「備考欄の文字列」として残した。正規表現処理の詳細は、**本報告書全体末尾の「別添資料:備考欄前処理プログラムソース」** を参照されたい。

またこの結果得られた「突合 DB(ユニークキー)」の各項目の件数を**表 B** に示す。

本研究で「付帯情報あり」と判断した基準は表 B に示す通りで、22項目に対する条件で決定している。しかしながら、備考欄に溢れた文字列を復元することでこの条件に合致する件数は変化してしまう。表 B の件数中の「太字・背景薄灰色」の箇所は、この備考欄処理によって、溢れ文字列が元の項目へ復元された結果、件数に変化があった箇所を示しており、特に項番 41 は、備考欄の文字列が大幅に「本来あるべき項目」へ戻されていることを示している。

本処理の結果、**最終的に「付帯情報あり」と判断された件数を表 B の最下段、及び表 D の L13** に示す。年によってバラツキはあるが全体として 32.3%であり、図1の通りである。

I 欄 II 欄病名に対する ICD コーディング

次に、死亡票の全ての I 欄 II 欄病名に対し、読点の削除、複数病名列挙の展開、文字の正規化、などの文字列処理を施した上で標準病名マスターを用いて ICD-10 コーディングを行った。要素技術は既に昨年度開発しており、本年度は 6 年間全てのデータに対しこれを適用した。

標準病名マスターを用いて、全ての I 欄・II 欄病名に対しほぼ原記載のまま ICD-10 コーディング可能だったのは死亡票の約 44% であるが、上記の多様な前処理を十分に行うことで、全ての病名が ICD-10 コード化された死亡票は約 80%にまで増加した。**表 D の L17, L18** にその件数と割合の詳細を示す。**この 80%の死亡票が IRIS での仮原死因決定処理へ入力可能**なものとなった。

<u>STEP2) IRIS での仮原死因確定処理</u>

IRIS は他の STEP と異なり Windows 上で動作するため、解析サーバーの VirtualBox 上の仮想マシン(Windows10)で処理を行い、他の STEP とは共有データ領域を介してデータやりとりを行った。詳細は「**【別添資料1】 本研究で構築したシステムの詳細**」を参照されたい。

IRIS による仮原死因確定処理は、多くのデータに対して一度に行うと極端に遅くなるため、高速化のため各年のデータを 50 分割して処理を行った。この処理により 1 年のデータにつき約1日で処理可能であった。また、IRIS は ICD-10 の国際版に準拠しており、日本国内で適用されている独自コード(詳細 5 桁目分類など)は実装されていない。このような事例については IRIS が出力する仮原死因コード、死亡票における確定原死因コードのそれぞれについて修正処理を行った。この処理は昨年度と同様のため割愛する。(**令和2年度統括研究報告書**を参照のこと)

IRIS 処理の結果、仮原死因が決定できた件数を**表 D** の L19 に、また割合を L20 に示す。突合 DB **全年で 99.26%の死亡票に対し、仮原死因が決定可能**であった。決定できなかったものは IRIS が Reject コードを出力するが (0.74%) 、その原因の内訳は、**表 D** の L21～L30 に示す通りである。

IRIS は Reject コードを出した場合も原死因コードを出力することがある (L31 に件数示す)。これは Rejected (Maybe) などのケースであるが、非常に稀なことと、後の機械学習での悪影響を考慮し、以降の処理では、Rejected 無しのもの (L19) を対象とした。

今回使用した死亡票は、年によって準拠する ICD-10 コードが異なっており、2015,16 年は、ICD-10 2003 年版、2017～2020 年は ICD-10 2013 年版となっている。一方、本研究の自動 ICD-10 コーディングに用いた標準病名マスターは V5.04 (2013 年版準拠)である。**表 D** を見ると、この影響により、2015,16 年に対するコーディング結果が 2017～2020 年に比べ若干悪い結果となっていることが判明した。このような項目について薄灰色で示している。また、L38 や L48 を見ると、仮原死因の変更割合が、2017～2020 と比べ明らかに高くなっており、後の機械学習への悪影響が考えられた。そこで、以降の処理では使用する ICD-10 のバージョンを揃え、**2017～2020 年のデータのみにて実験を行う**こととした。

<u>STEP3) IRIS 処理結果の解析</u>

IRIS 処理により確定された仮原死因と、国内での確定原死因を比較するため、IRIS の出力情報と元の突合 DB の情報を合わせ、この後の解析用の

「統合テーブル」を作成した。仕様を**表 C** に示す。統合テーブルは性別、生年、没年、に加え、IRIS の処理結果、死亡票の確定原死因、各付帯情報の項目の記載の有無などが全て含まれている。

統合テーブルのコード修正処理

表 C の項番 16 は、仮原死因と確定原死因の一致・不一致であり、機械学習の際には正解データとして用いられるものである。しかしながら、国内の原死因の方が IRIS よりも粒度が細かい（桁が多い）、あるいは逆に IRIS 側の方が国内よりも粒度が細かい、また国内では原死因として採用しないコードが仮原死因として採用されている、などの理由により、本来「一致」として良いものが「不一致」となっているケースがあった。この対処のため、粒度が細かい方のコードは桁を落とす、コードを修正する、あるいは対象レコードから外す、など「**IRIS の仮原死因コードと国内原死因コードを合わせる修正処理**」を、頻度が多いパターンを対象に可能な限り行った。このルールは昨年度と同様であるため、詳細は割愛する。

一部対象レコードから外したものもあるが、表 D の L34 の通り、元データの 99.99% が残っており、大きな影響はない。

この結果、再度「一致・不一致」を計算し、仮原死因の変更有無をまとめたものが、表 D の L35～L54 である。

IRIS が仮原死因を決定したもの(L19) に修正処理を加え、対象とされたレコード数が L33 である。この中で**付帯情報があったものは 2017～2020 年データで 29.2% 存在**した。全体では「付帯情報あり」のものは 32.3%であったが、対象を「全病名に ICD-10 コードが付与できた 80%」に絞っているため、この差が生まれたと考えられる。例えば、ICD-10 コーディング不可能な複雑なケースは付帯情報が含まれている可能性が高い、という理由が考えられる。

一方、付帯情報がある 29.2%の中で、**仮原死因の変更があったものは 12.9%、無しは 87.1%** であった。 C-1 で述べた「300 件抽出データに対する分析結果（原死因変更：13.6%）とも大きく矛盾しない。これら 2017～2020 年データでの仮原死因変更割合を表 D から抜粋して表3に記す。

仮原死因	変更あり	変更なし	計
付帯あり (29.2 %)	162,654 (12.9 %)	1,100,078 (87.1 %)	1,262,732
付帯なし (70.8 %)	164,008 (5.4 %)	2,892,357 (94.6 %)	3,056,365

表 3：2017～2020 年データの仮原死因変更の割合

また、原死因の変更以外にも、人手によるチェックで修正が行われるものがある。これが損傷や外因の影響（ICD-10 第19 章）に対する「外因コード」(V01-Y98)の追加、周産期における母側病態コードの追加などの「コード追加」である。つまり原死因には変更がなくてもこのような補足コードが追加されることがあり得る。「付帯情報あり」についてこれらを細分化したものが以下の表4である。C-1) 節での 300 件抽出データに対する同様の分析「**表2：人手確認対処内容の内訳**」とも似たような分布となっている。

国内のオートコーディングシステムとは異なり、IRIS によるシミュレーションではあるが、**大規模実データを対象とした解析においても、やはり「付帯情報あり」のうち、約 8 割のものは「原死因の変更もコードの追加も行う必要がない」** ことが確認された。前述の通りコード追加が必要と考えられるケースは現状のオートコーディングシステムでも自動抽出できていることから、**後は仮原死因の変更の有無が自動判別できれば、人手対処不要な約 8 割のデータを自動排除することが可能**となる。

仮原死因	コード追加 必要	コード追加 不要
変更あり	29,279 (2.3 %)	133,375 (10.6 %)
変更なし	112,185 (8.9 %)	987,893 (78.2 %)

表 4：付帯情報有りのものに対するコード変更・追加の割合

STEP4) 機械学習用データセットの作成

次に、「仮原死因の変更あり/無し」を教師データとし、「I 欄 II 蘭病名の ICD-10 コードと付帯情報」から仮原死因変更の有無を予測する分類器を開発した。

昨年度、50 万件の死亡票サンプリングデータから全病名 ICD−10コードが付与された 32 万件、さらにこの中で付帯情報が有る 8 万件のデータに対して、最もシンプルな［XGBoost-Simple]を適用し、Accuracy 90.3% を得ている。これは付帯情報の各項目について記載の有無しか用いておらず、内容を考慮していない。この「付帯情報の内容」をいかに学習に組み込むか、が本年度の大きなテーマであった。

BERT を用いた予測実験

本年度の機械学習手法で最も期待されたのが［BERT］である。近年様々な自然言語処理タスクで従来の精度を塗り替えており、本研究でも大幅な精度向上が見込まれたため、まず BERT を用いた予測実験を行った。

比較のために昨年度の[XGBoost-Simple］と同じ 8 万件のデータを用い、年齢、性別、使用された ICD-10 コード（1553 列)、各付帯情報の有無を（22 列）からなる 1577 次元の「共通ベクトル」をそのまま用いた。BERT を用いた本研究のアーキテクチャは、「共通ベクトル(1577 次元)」と「BERT モデルに付帯情報の文字列を通した結果得られる意味情報の分散表現(768 次元)」を上位の全結合層で統合し、最終的に2値分類の結果を得る」というもので、他のタスクでも良く用いられる手法である。

これに対し学習率の減衰方法や、インバランスデータへの対応方法など8種類の細分化を行い、学習を行った。詳細は「**別添資料 2:BERT を用いた予測モデルの学習実験**」を参照されたい。また、8種類の細分化手法の結果を「**表 E: BERT モデルを用いた各学習手法の予測精度一覧**」に示す。

結果として事前の期待に反し、最も良かった RedLR でもベースラインである昨年度の Accuracy から大きな向上は見られず、他の細分化手法はベースラインを下回る結果となった。このことから、以降の実験では、BERT 以外の方法（[XGBoost-Embed])に絞って本実験を行うこととした

XGBoost 系手法の機械学習用データセット作成

以降では、BERT 以外の方法に絞って実験を行うため、昨年度と同様に［XGBoost-Simple]、また本年度加えた［XGBoost-Embed］の5手法（TF・IDF、LSI、WORD2VEC、DOC2VEC (PV-DM)、DOC2VEC (PV-DBOW)）についてそれぞれ付帯情報の意味ベクトルを作成/学習し、XGBoost で2値分類するための機械学習用データセットを作成した。詳細を「**表 F:各種機械学習手法の結果一覧**」に記す。

全手法に用いた「共通ベクトル」は年齢、性別、出現した病名に対する ICD10 コード、各付帯情報の項目の記載の有無、からなるもので、これに加えた付帯情報の「意味ベクトル」が各手法で異なっている。

[BASELINE]（XGBoost-Simple）では意味ベクトルが存在せず、0 次元である。[TF・IDF]では出現頻度 100 以上(ストップワード除く）の単語を使っているため、2938 次元、[LSI]は 195 次元、残りは 200 次元ベクトルで表現されている。

共通ベクトルと付帯情報の意味ベクトルを結合したものが学習用ベクトルとなり、次元数は表 F に示した通りである。

STEP5) 各手法での仮原死因変更有無予測モデルの学習

最後に、STEP4 で作成されたデータを用いて各手法での仮原死因変更有無予測モデルの学習を行った。この予測モデルは共通して XGBoost を用いている。ハイパーパラメータは事前に探索を行い決定した。パラメータ詳細は本報告書全体の「**別添資料:分類器学習プログラムソース**」（fit_and_predict_xgboost.py）を参照されたい。また

トレーニングセットとテストセットは 4:1 に分割した。

各手法での結果を「**表 F:各種機械学習手法の結果一覧**」に記す。

最も精度が高かったものは［XGBoost-Embed］の TF・IDF であり、 Best Accuracy（正解率）0.949 を実現した。BASELINE（XGBoost-Simple）は 0.915～0.920 であったことから、大きく精度が向上している。また、例えば 2020 年データで ROC-AUC で 0.953、PR-AUC で 0.857 と非常に高い精度での分類が可能であった。例として 2020 年データの「変更あり」のものを対象とした時の ROC Curve を図2に、Precision(適合率)-Recall(再現率) Curve を図3に示す。

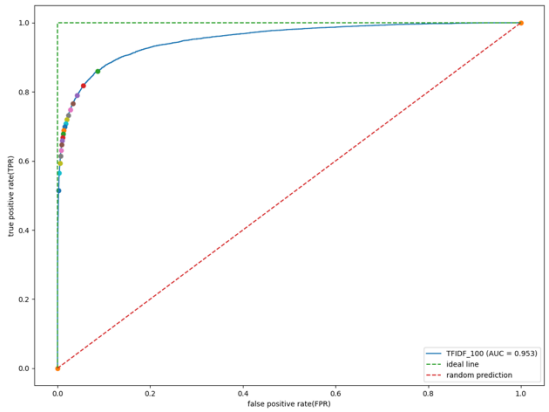


図2 ROC Curve（TF・IDF, AUC 0.953）

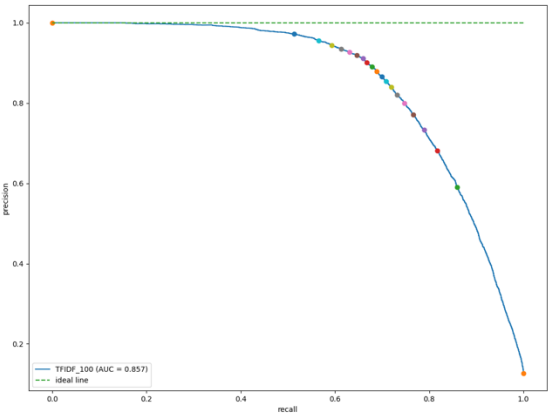


図 3 Precision-Recall Curve
（TF・IDF, AUC 0.857）

原死因の変更があるものが 12.9% しかないという非常に不均衡（インバランス）なデータであり、AUC-ROC は高い値が出やすいが、「変更あり」に注目した時の Precision -Recall 曲線の Area under curve（PR-AUC）も非常に高い値であったことは本手法の有効性を示す大きな特徴である。

2017～2020 年はいずれも ICD-10 2013 年版で統一されており、年による大きな差はない。BASELINE と比較すると、[XGBoost-Embed] の各手法はいずれも大きな精度向上を果たしており、DOC2VEC（PV-DM）が若干悪く、それ以外の LSI, WORD2VEC, DOC2VEC（PV-DBOW）も TF・IDF より僅かに劣るものの、いずれも大差なく同程度に高い精度を実現していた。

上記の手法に共通することとして、システムの最終的な出力は「変更あり（1）」「なし(0)」の2値ではなく、その範囲の動的な数値であるため、現場で使用する目的に応じて閾値を設定することが可能であると同時に、この数値を適切に変換することで出力に対する確信度と解釈することも可能である。

また、閾値の設定によって変化するが、実際の現場での導入に際しては、「変更あり」のものをなるべく高精度にスクリーニングしたいことから、Recall を重視した設定（閾値）が適していると考えられる。

C-3) ICD-11 における死亡診断書や死亡統計ルールの動向

ICD-11 における死亡診断書の動向

2021 年 10 月に開催された WHO-FIC 会議において、WHO が電子的な標準死亡診断書形式の普及を考えているという報告がなされた。WHO の考えでは、各国内で既に同様の電子的な報告システムが導入されている場合、国内のフォーマットを変える必要はないが、この標準形式で出力できるようにすることで、各種ソフトウエアからの可用性の向上を図りたいということであった。各国に対するアンケート結果によると、いくつか各国独自の死亡診断書項目が存在している場合があるが、大まかに WHO 標準形式とそぐっているということであった。今後電子化が十分に浸透すること、また標準形式に含まれる項目について

各国のコンセンサスが得られることが重要で、現状すぐにこの標準死亡診断書形式へ移行する訳ではないが、我が国でもこの動きを注視して国内での報告形式の見直しをする必要があると考えられた。

また、我が国では手書き入力された死亡診断書を元に報告の過程で電子化入力されるという手順を踏むが、フランスやドイツにおいては、既に Web あるいはモバイルアプリにて電子的に入力するシステムが導入されており、その過程でスペルチェックや入力項目間の不整合のチェックがなされている。業務効率化の観点から我が国においても、このような「報告時点」からの電子化と簡易チェックを考えていくことが重要な方向性であると考えられた。

死亡統計ルールの動向

ICD-11 における死亡統計の考え方とルールについては、Web 公開されている WHO ICD-11 Reference にて記載がなされた。現時点では詳細情報について主に第2章 2.17 Mortality Statistics から 2.23 Annex for Mortality Coding までに記載されている。根本となる考え方については、ICD-10 の時を踏襲しているものの、細かなコードや記述法については大幅にアップデートされており、これまでの原死因選択ルールベースについても大幅な更新が必要となっている。図4に ICD-11 にて加えられた全体のワークフロー図を示す。

また、2021 年 10 月に開催された WHO-FIC 会議においては、ITC (Informatics and Terminology Committee)にて、WHO cause of death identification tool についての進捗説明が行われた。これは ICD-10-SMoL (Startup Mortality List) と ICD11 reference guide から得られたルールを用いた、Iris とは異なる WHO 製の「原死因確定ツール」である。SMOL 自体は ICD-10 の時代から存在しており、ICD-10 の 3,4 桁コードを使ったコーディングができない国を対象とした死因統計報告の第一段階として位置付けられる、粒度の粗いコードセットとなっている。本ツールは ICD-10 の粒度、SMOL の粒度、ICD-11 の粒度

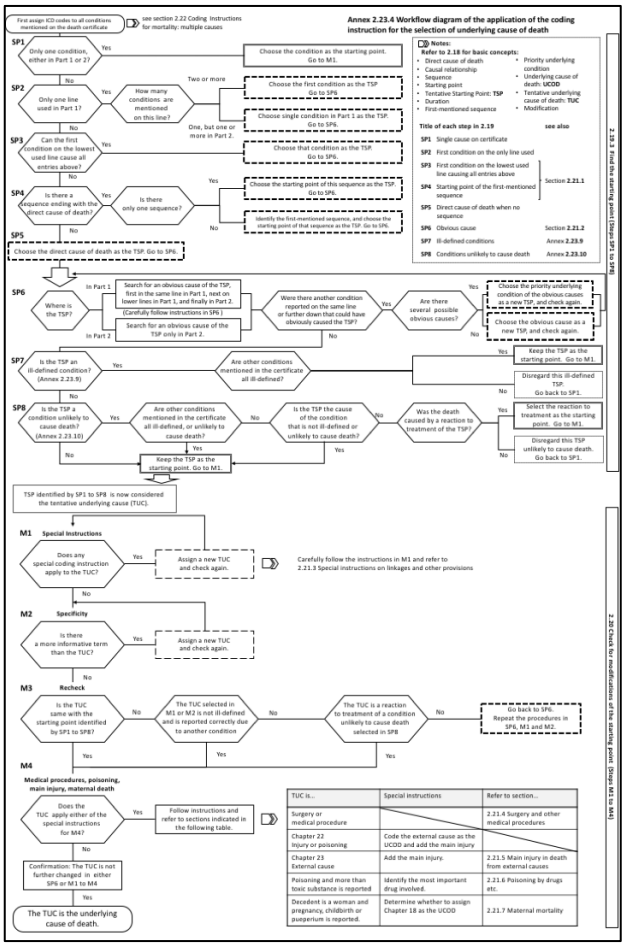


図 4. ICD-11 原死因決定ワークフロー

それぞれで原死因選択ができる Web ベースのツール提供を目指しているということであった。現時点では Reference Guide から抽出されたルールについては一部しか実装できておらず、CDC death certificate を用いたテストにおいては、SMOL ルールを用いた時で 73.5%の精度、Reference Guide からのルールも用いて 84.9%の精度、また ICD-11 でコードされた死亡票について 79.2%の精度と報告があった。但し病名以外の付帯情報(各所見・外因の手段状況・備考欄等)は一切考慮していない。

WHO cause of death identification tool はその元となった粗い粒度である SMOL から発展し今や ICD-10, ICD-11 の粒度での原死因選択ルール全体が実装されようとしている。一方、Iris はフル ICD-10 を用いた原死因選択ツールであり、ICD-11 への移行が進められているがまだ完成には至っていない。WHO

によると、WHO cause of death identification tool はツール間の架け橋になり得るもので、本ツールで実装されるルールは他のツールでも用いることができる、ということであった。このことから、WHO の当該ツールでは、単に SMOL の粒度だけではなく、ICD-10,11 の粒度でも原死因選択が可能な、Iris とも競合するツールに発展させていく意向があると考えられた。但し、WHO の当該ツールも Iris(ICD-11 対応版)も、現時点ではいずれも ICD-11 における原死因選択ツールを完全に網羅している訳ではなく開発途中である。また病名以外の付帯情報を考慮している訳ではないことには注意が必要である。

D. 考察

本研究の成果で、死亡票実データの約 80%を対象として、IRIS による仮原死因確定処理を行い、確定原死因と比較することで、原死因コードの変更の割合、また外因や母側病態コードの追加割合が明らかになった。

何らかの対処が必要なもの(原死因コードの変更、外因や母側病態コードの追加)は両者を合わせて 2 割であり、最初に 1:4 の分類タスクにより「対処の必要がない 8 割」を除去した後に、細かな対処内容の分類を行う 2 段階処理が適していると考えられた。最初の分類タスクが高精度に行えるだけでまず 8 割の人手処理を削減できることが期待でき、また 2 段階目の処理によって残りの 2 割に対する人手作業の効率化が図れると期待できる。

また、これを開発ターゲットに定め、機械学習により I 欄 II 蘭病名と付帯情報からこの変更の有無を予測し、確信度と共に導出する2値分類器を複数のアルゴリズムで開発した。何らかの付帯情報が存在する死亡票を対象として、Accuracy 95%, ROC-AUC 0.953, PR-AUC 0.857 という非常に高い分類精度を実現し、本研究の手法の有効性が示された。

一方、表3の結果より**「付帯情報がない」にも関わらず「仮原死因に変更があった」ものが 5.4%存在している**。これは**本来存在しないはず**のものであり、ICD-10 コード化する際の手法(本研究では標準病名マスター、厚生労働省内では独自の辞書)、あるいは原死因を決定する際のルールテーブルの違い(IRIS / 国内のオートコーディングツール)の影響と考えられ、本研究の枠組みでは除去できない限界である。

本研究の手法での Accuracy 95% という値は、付帯情報の有無に関係なく、オートコーディングツールの仕組みの違いが持っている本質的な“ずれ”がそもそも 5.4% ある、という事実に大きく影響されていると考えられ、達成しうる上限値なのかもしれない。

本研究では使用することができなかったが、厚生労働省内での本格的な導入時には、**国内のオートコーディングツールを元に仮の原死因を決定し、これを教師データとして本研究の手法で学習をやり直す**ことでさらに精度が向上すると考えられる。

機械学習の性質上、100%の精度実現は困難であるが、この本研究の手法は**付帯情報の影響による原死因コードの変更の有無のみならず、確信度も出力することが可能**である。これも参考情報としてユーザーへ提示することで従来の人手による確認作業の正確性・効率性向上に大いに寄与する支援ツールとなると期待される。

BERT が事前の予想に比してあまり効果を発揮しなかった理由としては、付帯情報の文字列は総じて長いものが少なく、BERT モデルへの入力に適していないということが考えられる。寧ろ TF・IDF のような古典的手法が奏功していたことを考えると、高度な言語モデルで「文脈を捉える」処理よりも、「特定の重要な単語の出現をきちんと捉えられているか」という点の方が効いているのではないかと考えられた。また、[XGBoost-Embed] の TF・IDF、LSI、WORD2VEC、DOC2VEC(PB-DBOW)の4手法の間にはそこまで大

きな差はないが、唯一 TF・IDF が異なっているのはベクトルの次元数が他と比べて多いことである。2段階目で XGBoost を用いることを考えると入力となるベクトルの次元数はある程度大きい方が良い、という可能性も考えられる。

ICD-11 における死亡診断書の動向からは、我が国においても**入力時点からの電子化を推進することで作業効率化とスペルチェックや入力項目間の不整合のチェックなどによる質向上を図る**方向性が今後重要と考えられた。項目についても我が国独自の項目（あるいは選択肢の粒度）と WHO 標準形式との整合性を確保していく観点が重要と思われる。多少本研究の本筋から外れるが、そもそも入力欄に文字数限定があり、溢れた文字が備考欄に記載される、という方式自体が前時代的に感じられる。病名や期間、手術、解剖の詳細などの情報が途切れてしまい、AI などを用いた自動解析に極めて適さない仕組みである。海外で、死亡診断書の入力を電子デバイスから行っているシステムで同様の仕様は見当たらなかった。本研究では「備考欄に溢れた文字を元に復元する前処理」を適用したが、その開発に膨大な時間がかかったことを考えると、今後この点を改善することも重要であろう。

さらに、このように一層進んだ電子化方式にて収集された死亡票データに対し原死因を確定するプロセスについては、効率化の観点から”Iris ICD-11 対応版” もしくは ”WHO cause of death identification tool” を採用することも有力な選択肢であることには変わりない。しかし、両者とも開発途中であり今後動向を注視することが必要である。WHO によると当該ツールに導入される「原死因選択ルールセット」は、他のツールでも用いることができるよう共有するということであったため、**将来仮にどちらを用いることとなっても実用上の問題は生じない**と思われた。

これらの2つはこれまで国内(厚生労働省内)で導入されてきたオートコーディングツールを ICD-11 対応とするときの代替候補ということであり、もちろん我が国独自のオートコーディングツールを ICD-11 対応へ更新するという方向性も考えられる。しかし、以上3つのどの方式を取るとしてもそのツール単体では「付帯情報を考慮するような高度な原死因確定処理」は実現できておらず、現在病名以外の各種付帯情報の存在等によって人手による確認が行われている(約35%の死亡票)ステップについては何らかの支援が必要である。本研究の AI 技術による支援手法は、選択ルールセットを実装した何らかのオートコーディングツールにより仮の原死因を決定した後、「**付帯情報を考慮すると仮原死因コードに変更が起こるか否か**」をその確信度と共に高精度に提示するものであるため、<u>どのオートコーディングツールにも組み込むことが可能</u>である。そのため本研究のような <u>AI による支援手法をルールベースのオートコーディングツールと組み合わせて用いることは ICD-11 においても十分に可能</u>であり（ICD-11 コードにおいて再学習させるプロセスは必要であるが）、原死因確定プロセスの正確性・効率性向上のために極めて重要であると考えられる。本研究での成果は今後我が国における原死因決定プロセスへの AI 活用に向けて大いに寄与する貴重な知見であると共に、本成果を元にした現場への AI 支援システム導入が望まれる。

E. 結論

本研究では、死亡票の実データに対する仮原死因確定コーディングツールとして IRIS を使用し、約80%の死亡票に対し仮原死因を決定した上で、これを確定原死因と比較することで得られた教師データを元に、機械学習で「付帯情報の影響により原死因コードが変更されるか否か」を確信度と共に高精度に提示する手法を開発した。結果として

Accuracy 95.0%, ROC-AUC 0.951, PR-AUC 0.859 という非常に高い精度を得た。この支援手法を既存のオートコーディングツールの処理結果に組み合わせることで原死因決定プロセスの正確性・効率性向上に大いに寄与すると期待される。また、この手法は将来の我が国でのオートコーディングツールの選択候補に依存せず、どれであっても組み合わせて使えることから、近い将来の ICD-11 適用後においても有力な手法と考えられる。

F. 健康危険情報

なし

G. 研究発表

1. 大井川仁美, 今井 健, 香川璃奈, 明神大也, 今村知明. 原死因決定プロセスの効率化に資する機械学習による原死因コード変更予測. 医療情報学 41(Suppl.):797-800, 2021.＜第 41 回医療情報学連合大会 研究奨励賞＞

H. 知的財産権の出願・登録状況

なし

表A:「突合DB」の項目・記載内容の詳細

項番	項目名	由来	項目日本語名称	バイト数	意味	記載内容の仕様
0	id		仮名ID		Irisに入れる用の通し番号	
1	sub_area	死亡個票	届出地(8)	8	届け出が出された場所の番号	＜1,2バイト目(都道府県)＞ 01-47　北海道～沖縄県 「都道府県別市区町村符号及び保健所符号一覧」参照のこと。昭和47年1月からは行政管理庁(現総務省)の定める「統計に用いる標準地域コード」を採用した。 ＜3,4バイト目(保健所)＞ 01-29　指定都市が設置した保健所 31-49　保健所を設置する市(指定都市を除く)が設置した保健所 51-98　道府県が設置した保健所 01-69　区部の保健所(東京都) 71-98　区部以外の保健所(東京都) ＜5バイト目(支所符号)＞ A-Z　受け付けた市区町村支所の符号 A支所～Z支所 △　支所なし ＜6バイト目(市区町村:種類)＞ 1　指定都市、特別区 2　市(指定都市を除く) 3-7　町村 ＜7,8バイト目(市区町村:順位)＞ 01-99　都道府県における順位
2	jiken_nm		事件簿番号(4)	4	事件簿番号	0001-9999市区町村(支所を含む)における事件簿番号を表す。
3	shori_ym		処理年月(6)	6	処理年月	ex. 201501-202012　2015年1月処理～2020年12月処理
4	bikou_yn		備考欄記述有無(1)	1	備考欄に記述があるかないか	0　記述なし 1　記述あり
5	death_kbn		「死亡した人の住所」のうち届出地区分(1)	1	届出地の区分	1　届出地と同じ 2　届出地以外 3　外国 V　不詳 △△　未記入または届出地不詳
6	d_tdfk		都道府県(8)	8	死亡したところの都道府県名	漢字
7	d_city		市区町村(12)	12	死亡したところの市、郡、東京都の区	漢字
8	d_town		市区町村(18)	18	死亡したところの町、村、指定都市の区	漢字
9	d_add		コード(7)	7	死亡したところのコード	＜1,2バイト目(都道府県)＞ 01-47　北海道～沖縄県 「都道府県別市区町村符号及び保健所符号一覧」参照のこと。昭和47年1月からは行政管理庁(現総務省)の定める「統計に用いる標準地域コード」を採用した。 △△　未記入または届出地不詳 ＜3バイト目(市区町村:種類)＞ 1　指定都市、特別区 2　市(指定都市を除く) 3-7　町村 △　未記入または届出地不詳 ＜4,5バイト目(市区町村:順位)＞ 01-99　都道府県における順位 △△　未記入または届出地不詳 ＜6,7バイト目(保健所)＞ 01-29　指定都市が設置した保健所 31-49　保健所を設置する市(指定都市を除く)が設置した保健所 51-98　道府県が設置した保健所 01-69　区部の保健所(東京都) 71-98　区部以外の保健所(東京都) △△　未記入または届出地不詳
10	Ia_c		「死亡の原因」に含まれるI欄・ア欄の原因(38)	38	ア欄記載の傷病名	漢字
11	Ia_p		期間(16)	16	期間	漢字
12	Ib_c		「死亡の原因」に含まれるI欄のイ欄の原因(38)	38	イ欄記載の傷病名	漢字
13	Ib_p		期間(16)	16	期間	漢字
14	Ic_c		「死亡の原因」に含まれるI欄のウ欄の原因(38)	38	ウ欄記載の傷病名	漢字
15	Ic_p		期間(16)	16	期間	漢字
16	Id_c		「死亡の原因」に含まれるI欄のエ欄の原因(76)	76	エ欄記載の傷病名	漢字
17	Id_p		期間(32)	32	期間	漢字
18	II_c		「死亡の原因」に含まれるII欄の原因(76)	76	II欄記載の傷病名	漢字
19	II_p		期間(32)	32	期間	漢字
20	ope_flg		手術フラグ(1)	1	手術の有無	1なし 2あり △未記入
21	ope_detail		手術の部位及び所見(116)	116	手術の詳細記述	漢字
22	ope_bk_flg		(手術)備考欄への記載(1)	1	手術に関して備考がある場合はフラグ	1記載あり △記載なし
23	ope_ymd		手術日(8)	8	手術年月日	＜1～4バイト目＞ 0000-9999手術をした年(西暦) △△△△未記入もしくは手術無し ＜5,6バイト目＞ 01～12手術をした月 △△未記入もしくは手術無し ＜7,8バイト目＞ 01～31手術をした日 △△未記入もしくは手術無し

24	ap_flg		解剖フラグ(1)	1	解剖の有無	1解剖なし 2解剖あり △未記入
25	ap_detail		解剖の部位及び所見(116)	116	解剖の詳細記述	漢字
26	ap_bk_flg		(解剖)備考欄への記載(1)	1	解剖に関して備考がある場合はフラグ	1記載あり △記載なし
27	dc_type		死因の種類(2)	2	自然死・不慮の外因死・その他の不詳の死などの種類を表す数字	01　病死・自然死 02　交通 03　点等 04　溺水 05　火災 06　窒息 07　中毒 08　その他 09　自殺 10　他殺 11　不詳の外因死 12　不明の死 △△,00　未記入
28	inj_ymd		傷害が発生したとき(8)	8	障害発生年月日	＜1～4バイト目＞ 0000-9999傷害が発生した年(西暦) △△△△未記入もしくは傷害ではない ＜5,6バイト目＞ 01-12傷害が発生した月 △△未記入もしくは傷害ではない ＜7,8バイト目＞ 01-31傷害が発生した日 △△未記入もしくは傷害ではない
29	inj_hm		傷害が発生したとき(5)	5	障害発生時分	＜1,2バイト目＞ 00-11傷害が発生した時 △△未記入もしくは傷害ではない ＜3,4バイト目＞ 00-59傷害が発生した日 △△未記入もしくは傷害ではない
30	inj_p_type		傷害が発生したところの種別(1)	1	住居・工場・道路などの障害が発生した場所の種類を表す数字	1住居 2工場及び建設現場 3道路 4その他 △未記入もしくは傷害ではない
31	inj_detail		傷害が発生したところその他の記述(40)	40	傷害が発生したところの種別にない場合の記述(その他)	漢字
32	inj_tdfk		傷害発生場所(8)	8	傷害発生都道府県	漢字
33	inj_city		傷害発生場所(12)	12	傷害発生市郡	漢字
34	inj_town		傷害発生場所(18)	18	障害発生区町村	漢字
35	inj_method		手段及び状況(120)	120	障害発生時の手段及び状況に関する詳細記述	漢字
36	inj_biko		(傷害)備考欄への記載(1)	1	傷害発生に関して備考がある場合はフラグ	1記載あり △記載なし
37	inf_detail		「生後1年未満での病死」の病態・異状の詳細	84	妊娠・分娩時における母体の病態または以上に関する詳細記述	漢字
38	inf_biko_yn		備考欄への記載	1	「生後1年未満での病死」に関して備考がある場合はフラグ	1記載あり △未記入もしくは1歳未満ではない
39	sonota		その他付言すべき事柄	60	その他に付言すべきことに関する詳細記述	漢字
40	biko_yn		備考欄外字有無	1	備考欄に外字がある場合はフラグ	0備考欄あり、外字なし 1備考欄あり、外字あり △備考欄なし
41	biko_detail		備考欄	1024	備考に関する詳細記述	漢字
42	ym	死亡票	調査年	2	調査年	西暦の下2桁
43	sub_ym		提出年月	4	死亡票提出年月日	元号2桁
44	jiken_sb		事件簿番号サブ	1	事件簿番号の補助番号	1-9　事件簿番号は原則として同じ番号はないはずであるが、万一同じ番号があったときは1～9をつけて区分する。 △　通常は△
45	sex		性別	1	性別	1男 2女
46	birth_yn		出生年月日フラグ	1	出生年月日あるか	V不詳(出生年月日不詳。以下33～39カラムはVVVVVVV) △出生年月日の記入があるもの
47	birth_ymd		出生年月日時分	7	出生年月日	＜1バイト目＞ 1　明治 2　大正 3　昭和 4　平成 5　令和 V　不詳 ＜2,3バイト目＞ 01-64　元号に対して1年～終止年 VV　不詳 ＜4,5バイト目＞ 01-12　1月～12月 VV　不詳 ＜6,7バイト目＞ 01-31　月に対して1日～終止日 VV　不詳
48	birth_hm		出生年月日時分	4	出生時分	＜1,2バイト目＞ 00-23午前0時～午後11時 △△生存期間が31日以上のとき(調査票記入要領による) VV不詳 ＜3,4バイト目＞ 00-590分～59分 △△生存期間が31日以上のとき(調査票記入要領による) VV不詳
49	death_yn		死亡年月日フラグ	1	死亡年月日あるか	V不詳(死亡年月日不詳。以下45～51カラムはVVVVVVV) △死亡年月日の記入があるもの

50	death_ymd		死亡年月日時分	7	死亡年月日	＜1バイト目＞ 3　　昭和 4　　平成 V　　不詳 ＜2,3バイト目＞ 01-64　　元号に対して1年～終止年 VV　　不詳 ＜4,5バイト目＞ 01-12　　1月～12月 VV　　不詳 ＜6,7バイト目＞ 01-31　　月に対して1日～終止日 VV　　不詳
51	death_hm		死亡年月日時分	4	死亡時分	＜1,2バイト目＞ 00-23午前0時～午後11時 VV不詳 ＜3,4バイト目＞ 00-590分～59分 VV不詳
52	ori_dc		原死因(5)	5	原死因コード	＜1～3バイト目＞ A00-U04　　外部参照。原死因の詳細については「疾病、傷害および死因統計分類提要 ICD－10(2003年版)準拠　第2巻内容例示表」及び「人口動態統計用として追加設定した基本分類細分項目一覧」（平成27年）を参照のこと ＜4バイト目＞ 0-9原死因4桁目設置有りの場合 △原死因4桁目設置無しの場合 ＜5バイト目＞ A-I原死因5桁目設置有りの場合 △原死因5桁目設置無しの場合
53	gaiin		外因符号	4	外因コード	＜1～3バイト目＞ V01-Y89外部参照。原死因S00.0～T98.3についての外因符号。外因符号の詳細については「疾病、傷害および死因統計分類提要　ICD－10(2003年版)準拠 第2巻内容例示表」を参照のこと △△△外因死以外（原死因A00.0～R99・U04.9）の場合 ＜4バイト目＞ 0-9外因符号（上3桁）がV01～Y89で4桁目設置有りの場合の発生場所等、外因の状況コード　○交通事故以外の不慮の事故（外因符号W00～X59）による死亡の場合、発生場所コード。0 家（庭）、1 居住施設、2 学校、施設及び公共の地域、3 スポーツ施設及び競技施設、4 街路及びハイウェイ、5 商業及びサービス施設、6 工業地域及び建築現場、7 農場、8 その他の明示された場所、9 詳細不明の場所　○それ以外の外因（外因符号V01～V98、X60～Y89）による死亡の場合、発生場所等、外因の状況コード。詳細については「疾病、傷害および死因統計分類提要　ICD－10(2003年版）準拠　第2巻内容例示表」を参照のこと △外因死以外（原死因A00.0～R99・U04.9）、外因符号4桁目設置無しの場合
54	mom_sy		母側病態	5	母側病態コード	＜1～3バイト目＞ P00-P99外部参照。母側病態の詳細については「疾病、傷害および死因統計分類提要　ICD－10準拠　第2巻内容例示表」及び「人口動態統計用として追加設定した基本分類細分項目一覧」（平成27年）を参照のこと △△△スペース（1歳以上及び年齢不詳の死亡、0歳で病死以外の死亡の場合） ＜4バイト目＞ 0-9母側病態4桁目設置有りの場合 △スペース（1歳以上及び年齢不詳の死亡、0歳で病死以外の死亡、母側病態4桁目設置無しの場合） ＜5バイト目＞ A-I母側病態5桁目設置有りの場合 △スペース（1歳以上及び年齢不詳の死亡、0歳で病死以外の死亡、母側病態5桁目設置無しの場合）
55	twin_yn		単多胎別	2	単胎か多胎か数字	＜1バイト目＞ 1単胎 2-7双子～7つ子 88つ子以上 △スペース（1歳以上及び年齢不詳の死亡、0歳で病死以外の死亡の場合） ＜2バイト目＞ 1-71番目～7番目 88番目以上 V不詳 △スペース（単胎、1歳以上及び年齢不詳の死亡、0歳で病死以外の死亡の場合）
56	preg_yn		妊娠週数	1	妊娠の有無	0　　妊娠週数記入のあるもの V　　不詳 △　　スペース（1歳以上及び年齢不詳の死亡、0歳で病死以外の死亡の場合）
57	preg_w		妊娠週数	2	妊娠週数	12-9812週～98週 VV不詳 △△スペース（1歳以上及び年齢不詳の死亡、0歳で病死以外の死亡の場合）
58	mon_b_ymd		母の生年月日	7	母の生年月日	＜1バイト目＞ 3昭和 4平成 V不詳 △スペース（1歳以上及び年齢不詳の死亡、0歳で病死以外の死亡の場合） ＜2,3バイト目＞ 01-64元号に対して1年～終止年 VV不詳 △△スペース（1歳以上及び年齢不詳の死亡、0歳で病死以外の死亡の場合） ＜4,5バイト目＞ 01-121月～12月 VV不詳 △△スペース（1歳以上及び年齢不詳の死亡、0歳で病死以外の死亡の場合） ＜6,7バイト目＞ 01-31月に対して1日～終止日 VV不詳 △△スペース（1歳以上及び年齢不詳の死亡、0歳で病死以外の死亡の場合）
59	p_preg_bir		前回の妊娠	2	前回までの出生児数	00-180人～18人 1919人以上 VV不詳 △△スペース（1歳以上及び年齢不詳の死亡、0歳で病死以外の死亡の場合）
60	p_preg_dea		前回の妊娠	2	前回までの死産児数	00-190胎～19胎 2020胎以上 VV不詳 △△スペース（1歳以上及び年齢不詳の死亡、0歳で病死以外の死亡の場合）

61	child_1		子の数	2	今回の出生子含む出生子	01-191人～19人 2020人以上 △△スペース（1歳以上及び年齢不詳の死亡、0歳で病死以外の死亡の場合） VV不詳
62	child_2		子の数	2	出産児	01-391～39胎・人（出産児＝出生子（155～156カラム）＋死産児（妊娠満22週以後）） 4040胎・人以上（出産児＝出生子（155～156カラム）＋死産児（妊娠満22週以後）） △△スペース（1歳以上及び年齢不詳の死亡、0歳で病死以外の死亡の場合） VV不詳

表B:「突合DB（ユニークキーのみ）」の項目仕様とレコード件数

項番	項目名	意味	Bytes	付帯情報の有無の判定基準	2015年		2016年		2017年		2018年		2019年		2020年	
備考欄の処理（溢れ文字列の元項目への結合）					なし	処理後	なし	処理後	なし	処理後	なし	処理後	なし	処理後	なし	処理後
0	id	Iris処理用の仮名ID	7		1236039	1236039	1265327	1265327	1309551	1309551	1358114	1358114	1376694	1376694	1366215	1366215
1	sub_area	届け出が出された場所の番号	8		1236039	1236039	1265327	1265327	1309551	1309551	1358114	1358114	1376694	1376694	1366215	1366215
2	jiken_nm	事件簿番号	4		1236039	1236039	1265327	1265327	1309551	1309551	1358114	1358114	1376694	1376694	1366215	1366215
3	shori_ym	処理年月	6		1236039	1236039	1265327	1265327	1309551	1309551	1358114	1358114	1376694	1376694	1366215	1366215
4	bikou_yn	備考欄に記述があるかないか	1		1236039	1236039	1265327	1265327	1309551	1309551	1358114	1358114	1376694	1376694	1366215	1366215
5	death_kbn	届出地の区分	1		1236015	1236015	1265300	1265300	1309518	1309518	1358079	1358079	1376665	1376665	1366162	1366162
6	d_tdfk	死亡したところの都道府県名	8		1156709	1156709	1183673	1183673	1223805	1223805	1262804	1262804	1282413	1282413	1275576	1275576
7	d_city	死亡したところの市、郡、東京都の区	12		1158995	1158995	1185824	1185824	1226189	1226189	1265079	1265079	1284561	1284561	1278077	1278077
8	d_town	死亡したところの町、村、指定都市の区	18		1124819	1124819	1152259	1152259	1194588	1194588	1236935	1236935	1255436	1255436	1247524	1247524
9	d_add	死亡したところのコード	7		1232357	1232357	1261446	1261446	1305700	1305700	1354030	1354030	1372307	1372307	1363004	1363004
10	la_c	ア欄記載の傷病名	38		1233909	1233909	1263174	1263174	1307433	1307433	1355898	1355898	1374327	1374327	1364810	**1364811**
11	la_p	期間	16		1214814	**1215832**	1244394	**1245571**	1288530	**1289757**	1337818	**1339021**	1355927	**1357072**	1348402	**1349610**
12	lb_c	イ欄記載の傷病名	38		450934	450934	457226	457226	462464	462464	471565	471565	459618	459618	440092	440092
13	lb_p	期間	16		347244	**350195**	354441	**356991**	359010	**361373**	369950	**372359**	361549	**364192**	348866	**351381**
14	lc_c	ウ欄記載の傷病名	38		96485	96485	97574	97574	97091	97091	98826	98827	93878	93878	88240	88240
15	lc_p	期間	16		75853	**76334**	77120	**77503**	76986	**77327**	79260	**79566**	75721	**76110**	71628	**71993**
16	ld_c	エ欄記載の傷病名	76		18173	18173	18223	18223	18038	18038	18459	18459	17259	17259	16345	16345
17	ld_p	期間	32		13544	**13642**	13495	**13599**	13332	**13409**	13804	**13886**	13069	**13151**	12473	**12563**
18	ll_c	ll欄記載の傷病名	76		419695	419695	425148	**425149**	435249	435249	445932	445932	443940	443940	436382	436382
19	ll_p	期間	32		388944	**390494**	395980	**397590**	406795	**408483**	419622	**421251**	418640	**420008**	413205	**414629**
20	ope_flg	手術の有無	1	2の場合「有」	1225149	1225149	1255342	1255342	1300736	1300736	1349277	1349277	1366293	1366293	1357354	1357354
21	ope_detail	手術の詳細記述	116	記載あれば「有」	182103	**183651**	183126	**184509**	185573	**186793**	188201	**189459**	186746	**188132**	183044	**184462**
22	ope_bk_flg	手術に関して備考がある場合はフラグ	1	記載あれば「有」	2	2	1	1	2	2	3	3	3	3	3	3
23	ope_ymd	手術年月日	8	記載あれば「有」	162250	**170428**	162787	**171006**	164631	**173479**	166800	**176037**	165743	**174722**	162755	**172148**
24	ap_flg	解剖の有無	1	2の場合「有」	1224106	1224106	1254352	1254352	1299954	1299954	1348571	1348571	1365596	1365596	1356691	1356691
25	ap_detail	解剖の詳細記述	116	記載あれば「有」	29336	**29644**	30341	**30655**	30135	**30453**	29610	**29869**	28774	**29019**	24894	**25136**
26	ap_bk_flg	解剖に関して備考がある場合はフラグ	1	記載あれば「有」	0	0	1	1	0	0	4	4	7	7	7	7
27	dc_type	自然死・不慮の外因死・その他の不詳の死などの種類を表す数字	2	「空,00,01」以外の記載あれば「有」	1235172	1235172	1264496	1264496	1308575	1308575	1357129	1357129	1375599	1375599	1365430	1365430
28	inj_ymd	傷害発生年月日	8	記載あれば「有」	49959	49959	47870	47870	47829	47829	48061	48061	45781	45781	45373	45373

29	inj_hm	傷害発生時分	5	記載あれば「有」	38950	38950	37433	37433	37525	37525	36994	36994	35044	35044	34479	34479
30	inj_p_type	住居・工場・道路などの傷害が発生した場所の種類を表す数字	1	記載あれば「有」	50014	50014	47978	47978	47954	47954	48179	48179	45897	45897	45593	45593
31	inj_detail	傷害が発生したところの種別にない場合の記述（その他）	40	記載あれば「有」	14020	14020	13527	13527	13435	13435	13450	13450	13359	13359	12951	12951
32	inj_tdfk	傷害発生都道府県	8	記載あれば「有」	48211	48211	46278	46278	46459	46459	46897	46897	44569	44569	44499	44499
33	inj_city	傷害発生市郡	12	記載あれば「有」	47783	47783	45857	45857	46155	46155	46554	46554	44271	44271	44167	44167
34	inj_town	傷害発生区町村	18	記載あれば「有」	20275	20275	19414	19414	19618	19618	19628	19628	18805	18805	18605	18605
35	inj_method	傷害発生時の手段及び状況に関する詳細記述	120	記載あれば「有」	49125	**49894**	47148	**47882**	47111	**47850**	47669	**48409**	45524	**46229**	45230	**45951**
36	inj_biko	傷害発生に関して備考がある場合はフラグ	1	記載あれば「有」	0	0	0	0	0	0	2	2	1	1	0	0
37	inf_detail	妊娠・分娩時における母体の病態または以上に関する詳細記述	84	記載あれば「有」	621	621	661	661	563	563	582	582	548	548	547	547
38	inf_biko_yn	「生後1年未満での病死」に関して備考がある場合はフラグ	1	記載あれば「有」	0	0	0	0	0	0	1	1	0	0	0	0
39	sonota	その他に付言すべきことに関する詳細記述	60	記載あれば「有」	84312	**84105**	83150	**83159**	84982	**85135**	87383	**87572**	84598	**84844**	86204	**86465**
40	biko_yn	備考欄に外字がある場合はフラグ	1	記載あれば「有」	203493	203493	211584	211584	223953	223953	242542	242542	249490	249490	249037	249037
41	biko_detail	備考に関する詳細記述	1024	記載あれば「有」	203492	**34040**	211584	**36226**	223952	**36866**	242542	**40277**	249490	**40039**	249036	**40729**
42	ym	調査年	2		1236039	1236039	1265327	1265327	1309551	1309551	1358114	1358114	1376694	1376694	1366215	1366215
43	sub_ym	死亡票提出年月日	4		1236039	1236039	1265327	1265327	1309551	1309551	1358114	1358114	1376694	1376694	1366215	1366215
44	jiken_sb	事件簿番号の補助番号	1		7	7	30	30	3	3	0	0	0	0	0	0
45	sex	性別	1		1236039	1236039	1265327	1265327	1309551	1309551	1358114	1358114	1376694	1376694	1366215	1366215
46	birth_yn	出生年月日あるか	1		524	524	489	489	578	578	522	522	586	586	584	584
47	birth_ymd	出生年月日	7		1236039	1236039	1265327	1265327	1309551	1309551	1358114	1358114	1376694	1376694	1366215	1366215
48	birth_hm	出生時分	4		915	915	912	912	868	868	858	858	801	801	743	743
49	death_yn	死亡年月日あるか	1		21	21	19	19	24	24	30	30	27	27	19	19
50	death_ymd	死亡年月日	7		1236039	1236039	1265327	1265327	1309551	1309551	1358114	1358114	1376694	1376694	1366215	1366215
51	death_hm	死亡時分	4		1236039	1236039	1265327	1265327	1309551	1309551	1358114	1358114	1376694	1376694	1366215	1366215
52	ori_dc	原死因コード	5		1236039	1236039	1265327	1265327	1309551	1309551	1358114	1358114	1376694	1376694	1366215	1366215
53	gaiin	外因コード	4		65283	65283	63794	63794	66994	66994	68948	68948	66577	66577	66067	66067
54	mom_sy	母側病態コード	5		1783	1783	1815	1815	1675	1675	1703	1703	1592	1592	1445	1445
55	twin_yn	単胎か多胎か数字	2		1783	1783	1815	1815	1675	1675	1703	1703	1592	1592	1445	1445
56	preg_yn	妊娠の有無	1		1783	1783	1815	1815	1675	1675	1703	1703	1592	1592	1445	1445
57	preg_w	妊娠週数	2		1783	1783	1815	1815	1675	1675	1703	1703	1592	1592	1445	1445
58	mon_b_ymd	母の生年月日	7		1783	1783	1815	1815	1675	1675	1703	1703	1592	1592	1445	1445
59	p_preg_bir	前回までの出生児数	2		1783	1783	1815	1815	1675	1675	1703	1703	1592	1592	1445	1445
60	p_preg_dea	前回までの死産児数	2		1783	1783	1815	1815	1675	1675	1703	1703	1592	1592	1445	1445
61	child_1	今回の出生子含む出生子	2		1783	1783	1815	1815	1675	1675	1703	1703	1592	1592	1445	1445

62	child_2	出産児	2		1783	1783	1815	1815	1675	1675	1703	1703	1592	1592	1445	1445
				付帯情報有の個数	402535	**402270**	409253	**409065**	421472	**421300**	440264	**440176**	443578	**443521**	438377	**438271**

注:

(1) 本「**突合DB（ユニークキーのみ）**」とは死亡票・死亡個表の突合結果（**「突合DB」**）に対し、**届出地 (sub_area) ・事件簿番号 (jiken_nm)・処理年月(shori_ym)** の組み合わせをキーとし、重複がないものだけに絞ったものである。(元の「突合DB」の98.84%)

(2) 各欄からは、入力時の文字数制限により、入り切らない文字列が備考欄（biko_detail）に溢れて記入されることがあるが、本研究ではこの**「備考欄に溢れた文字列」を正規表現ルールにより、元の然るべき項目へ可能な限り復元して結合する処理**を行った。

(3) 各年の「なし」は**この処理を行う前**の「突合DB（ユニークキーのみ）」の各項目件数、「処理後」はこの**処理を行った後**の件数を示している。

(4) 5列目「付帯情報の有無の判定基準」とは、「**付帯情報として用いた項目**」について「どういう条件であれば、付帯情報ありとして判定したか」の基準が入力されている

(5) 件数中の「**太字・背景薄灰色**」の箇所は、この備考欄処理によって、溢れ文字列が元の項目へ復元された結果、件数に変化があった箇所を示す。特に項番41 は、**備考欄の文字列が大幅に「本来あるべき項目」へ戻されている**ことを示している。

表C：IRIS処理結果と死亡票データ（突合 DBユニークキーのみ）を合わせた「統合テーブル」仕様

配列番号	項目	意味	機械学習用基本ベクトルでの使用	備考
0	CertificateKey	仮名ID	YES	突合DB（ユニークキー）における仮名ID
1	DateBirth	生年	YES	機械学習用基本ベクトルでは、これらを合わせて、死亡時年齢に変換。（小数点以下2位で四捨五入）
2	DateDeath	没年	YES	
3	Sex	性別	YES	1: 男性 2: 女性
4	Iris（反転）	IRIS による仮原死因コード	YES	外因コードがある場合、日本仕様に合わせ死因コードと順番を入れ替え済み
5	MainInj	IRIS が付与した外因コード	NO	(例) V494 など
6	Reject	IRIS: Reject されたか否か & 種別	NO	No, Maybe, Code, Syntax, MainInjury, Muse
7	SelectedCodes	IRIS: コード置換・修正処理前の ICD10コード	NO	(例) Ia: R54(2Hours)/II: D531(30Months) など
8	SubstitutedCodes	IRIS: IRIS内辞書により置換されたICD10コード	NO	(例) R54(2Hours)*D531(30Months) など
9	ErnCodes	IRIS: N/A	NO	IRIS Ver5 以降使用されていない
10	AcmeCodes	IRIS: MUSE 処理後の ICD10コード	YES	(例) R54*D531 など。 **機械学習用の「用いられたICD-10コードに基づいた数値ベクトル」には仮原死因コードと合わせて、この欄の処理結果を用いる。**
11	MultipleCodes	IRIS: 多重分類分類用の ICD10コード	NO	(例) D531 R54 など。アルファベットソートされている。
12	ToDoList	IRIS: GUI インタフェースで入力されたTODO	NO	使っていない。
13	確定	日本: 死亡票に基づく確定原死因コード	NO	(例) D531 など
14	外因	日本: 死亡票に基づく外因コード	NO	(例) V494 など
15	母側	日本: 死亡表に基づく母側病態コード	NO	(例) P008C など
16	**一致/不一致**	**仮原死因コードと確定原死因コードの一致**	**YES**	**0: 一致、1: 不一致。機械学習では正解データとして使用**
17	Ia_c	ア欄記載の傷病名	NO	記載あり：1、記載なし：0
18	Ia_p	期間	NO	記載あり：1、記載なし：0
19	Ib_c	イ欄記載の傷病名	NO	記載あり：1、記載なし：0
20	Ib_p	期間	NO	記載あり：1、記載なし：0
21	Ic_c	ウ欄記載の傷病名	NO	記載あり：1、記載なし：0
22	Ic_p	期間	NO	記載あり：1、記載なし：0
23	Id_c	エ欄記載の傷病名	NO	記載あり：1、記載なし：0
24	Id_p	期間	NO	記載あり：1、記載なし：0
25	II_c	II欄記載の傷病名	NO	記載あり：1、記載なし：0
26	II_p	期間	NO	記載あり：1、記載なし：0
27	ope_flg	手術の有無	YES	付帯情報項目
28	ope_detail	手術の詳細記述	YES	付帯情報項目
29	ope_bk_flg	手術に関して備考がある場合はフラグ	YES	付帯情報項目
30	ope_ymd	手術年月日	YES	付帯情報項目
31	ap_flg	解剖の有無	YES	付帯情報項目
32	ap_detail	解剖の詳細記述	YES	付帯情報項目
33	ap_bk_flg	解剖に関して備考がある場合はフラグ	YES	付帯情報項目
34	dc_type	自然死・不慮の外因死・その他の不詳の死などの種類を表す数字	YES	付帯情報項目
35	inj_ymd	障害発生年月日	YES	付帯情報項目
36	inj_hm	障害発生時分	YES	付帯情報項目
37	inj_p_type	住居・工場・道路などの障害が発生した場所の種類を表す数字	YES	付帯情報項目
38	inj_detail	傷害が発生したところの種別にない場合の記述（その他）	YES	付帯情報項目
39	inj_tdfk	傷害発生都道府県	YES	付帯情報項目
40	inj_city	傷害発生市郡	YES	付帯情報項目
41	inj_town	障害発生区町村	YES	付帯情報項目
42	inj_method	障害発生時の手段及び状況に関する詳細記述	YES	付帯情報項目
43	inj_biko	傷害発生に関して備考がある場合はフラグ	YES	付帯情報項目
44	inf_detail	妊娠・分娩時における母体の病態または以上に関する詳細記述	YES	付帯情報項目
45	inf_biko_yn	「生後1年未満での病死」に関して備考がある場合はフラグ	YES	付帯情報項目
46	sonota	その他に付言すべきことに関する詳細記述	YES	付帯情報項目
47	biko_yn	備考欄に外字がある場合はフラグ	YES	付帯情報項目
48	biko_detail	備考に関する詳細記述	YES	付帯情報項目
49	付帯有無	付帯情報の有り無し	NO	27～48 に「付帯情報有りの基準」を満たすものがあれば1、それ以外0。 **これが1のものだけを機械学習の対象とした。**

注:

(1) 死亡個票中の全病名は項番17-26 に相当するが、機械学習用基本ベクトルにはこのままではなく、IRIS でのICD-10コード修正結果である AcmeCodes（項番 10）を用いた。
を用いた。IRIS でのコード修正機能により、期間表現を利用して「急性」のコードに変換する、などの処理がなされるためである。

表D:各処理段階の件数詳細

L	西暦年			2020	2019	2018	2017	2016	2015	2017-20	ALL
1	元号			令和2年	平成31年/令和1年	平成30年	平成29年	平成28年	平成27年	N/A	N/A
2	国内使用 ICD-10 バージョン			2013年版				2003年版		2013年版	N/A
3	死亡票件数			1384300	1393504	1374469	1351944	1319030	1301379	5504217	8124626
4	死亡個票件数			1384568	1393900	1374780	1325955	1280853	1249469	5479203	8009525
5	突合DB	レコード数		**1384263**	**1392578**	**1373589**	**1325413**	**1280808**	**1248057**	5475843	8004708
6		男性		713650	714048	705560	683577	661501	645711	2816835	4124047
7		女性		670613	678530	668029	641836	619307	602346	2659008	3880661
8	重複排除	キー(届出地＋事件簿番号＋提出年月日)の重複により排除したレコード数 (L5-L9)		18048	15884	15475	15862	15481	12018	65269	92768
9	突合DB	レコード数		**1366215**	**1376694**	**1358114**	**1309551**	**1265327**	**1236039**	5410574	7911940
10	(ユニークキー)		割合(／L5)	**98.70%**	**98.86%**	**98.87%**	**98.80%**	**98.79%**	**99.04%**	**98.81%**	**98.84%**
11		男性		703561	705117	697105	674859	652853	639005	2780642	4072500
12		女性		662654	671577	661009	634692	612474	597034	2629932	3839440
13		付帯情報あり(備考欄文字列前処理後)		438271	443521	440176	421300	409065	402270	1743268	2554603
14			割合(／L5)	**32.1%**	**32.2%**	**32.4%**	**32.2%**	**32.3%**	**32.5%**	**32.2%**	**32.3%**
17	IRIS処理	IRIS処理に回した件数(全傷病名がICD-10コーディング可能)		**1104767**	**1111603**	**1088872**	**1047946**	**1008941**	**984617**	4353188	6346746
18			割合(／L9)	**80.86%**	**80.74%**	**80.18%**	**80.02%**	**79.74%**	**79.66%**	**80.46%**	**80.22%**
19		IRIS で仮原死因決定可能(Reject なし)		**1095698**	**1102884**	**1080590**	**1040397**	**1002020**	**977952**	4319569	6299541
20			割合(／L17)	99.18%	99.22%	99.24%	99.28%	99.31%	99.32%	99.23%	99.26%
21		IRIS による Reject(と原因の内訳)		9069	8719	8282	7549	6921	6665	33619	47205
22			割合(／L17)	0.82%	0.78%	0.76%	0.72%	0.69%	0.68%	0.77%	0.74%
23			Syntax	102	115	119	109	97	97	445	639
24			Code	1014	1014	949	891	887	842	3868	5597
25			MultipleCause	0	0	0	0	0	0	0	0
26			MayBe	7820	7461	7064	6417	5791	5552	28762	40105
27			Muse	55	62	66	60	63	61	243	367
28			Interval	0	0	0	0	0	0	0	0
29			MainInjury	78	67	84	72	83	113	301	497
30			Coder	0	0	0	0	0	0	0	0
31		仮原死因(UCコード)が付与された件数		1104535	1111357	1088639	1047738	1008741	984422	4352269	6345432
32		外因コードが付与された件数		36603	36364	37180	35741	34973	36069	145888	216930
33	解析用統合テーブル	コード修正・非対象レコード削除後 (L19中)		**1095585**	**1102777**	**1080458**	**1040277**	**1001497**	**977469**	4319097	6298063
34	(BasicTable)		割合(／L19)	99.99%	99.99%	99.99%	99.99%	99.95%	99.95%	99.99%	99.98%
35		付帯情報あり		**319434**	**322559**	**317598**	**303141**	**292776**	**287358**	**1262732**	**1842866**
36			割合(／L33)	**29.2%**	**29.2%**	**29.4%**	**29.1%**	**29.2%**	**29.4%**	**29.2%**	**29.3%**
37		仮原死因変更あり		40493	40945	41516	39700	48190	48134	162654	258978
38			割合(／L35)	**12.7%**	**12.7%**	**13.1%**	**13.1%**	**16.5%**	**16.8%**	**12.9%**	**14.1%**
49			追加コードあり	7288	7307	7696	6988	7601	7699	29279	44579
40			追加コードなし	33205	33638	33820	32712	40589	40435	133375	214399
41		仮原死因変更なし		278941	281614	276082	263441	244586	239224	1100078	1583888
42			割合(／L35)	**87.3%**	**87.3%**	**86.9%**	**86.9%**	**83.5%**	**83.2%**	**87.1%**	**85.9%**
43			追加コードあり	27739	27876	28509	28061	26389	27371	112185	165945
44			追加コードなし	251202	253738	247573	235380	218197	211853	987893	1417943
45		付帯情報なし		776151	780218	762860	737136	708721	690111	3056365	4455197
46			割合(／L33)	**70.8%**	**70.8%**	**70.6%**	**70.9%**	**70.8%**	**70.6%**	**70.8%**	**70.7%**
47		仮原死因変更あり		40471	41391	41090	41056	63104	63032	164008	290144
48			割合(／L45)	**5.2%**	**5.3%**	**5.4%**	**5.6%**	**8.9%**	**9.1%**	**5.4%**	**6.5%**
49			追加コードあり	2065	2057	2088	1975	1706	1795	8185	11686
50			追加コードなし	38406	39334	39002	39081	61398	61237	155823	278458
51		仮原死因変更なし		735680	738827	721770	696080	645617	627079	2892357	4165053
52			割合(／L45)	**94.8%**	**94.7%**	**94.6%**	**94.4%**	**91.1%**	**90.9%**	**94.6%**	**93.5%**
53			追加コードあり	5537	5386	5396	5149	4113	3821	21468	29402
54			追加コードなし	730143	733441	716374	690931	641504	623258	2870889	4135651
55	機械学習用データ	対象レコード数 (＝L35)		**319434**	**322559**	**317598**	**303141**	**292776**	**287358**	**1262732**	**1842866**
56		ICDコード種類数 (Acme)		2061	2091	2066	2050	2006	2024	2693	2853

注:

(1) 2015, 2016 年と、2017～2020年 で原死因コーディングに使用されたICD-10のバージョンが異なる。(それぞれ 2003年版準拠、2013年版準拠)

(2) 本研究では全体を通じて、病名のICD-10コーディングに「標準病名マスターv5.04 (ICD-10 2013年版準拠)」を用いている。

(3) そのため、「2015,2016年」は「2017～2020年」と比べて、「全病名にICD-10コーディング可能な割合」「仮原死因変更の割合」など、いくつかの項目で低くなっている。このような使用するICD-10バージョンによる影響が確認された値は薄灰色の網掛けで表示してある。

(4) 機械学習以降の処理については、別表「各種機械学習手法の結果一覧」を参照のこと

表E：BERTモデルを用いた各学習手法の予測精度一覧

手法		RedLR	BalRedLR	BalRedLR_ BertOnly	BalRedLR_ Norm	CwRedLR	CyLR	BalCyLR	CwCyLR
Accuracy		**0.914**	0.866	0.819	0.863	0.877	0.900	0.858	0.882
F1-score		**0.591**	0.535	0.391	0.548	0.543	0.564	0.510	0.549
Precision		**0.756**	0.481	0.343	0.475	0.517	0.641	0.457	0.537
Recall		0.485	0.602	0.455	**0.648**	0.571	0.504	0.576	0.561
Confusion Matrix (正解, 予測)	(0, 0)	13918	12890	12422	12750	13130	13656	12817	13235
	(0, 1)	328	1356	1824	1496	1116	590	1429	1011
	(1, 0)	1078	833	1140	737	897	1038	888	918
	(1, 1)	1014	1259	952	1355	1195	1054	1204	1174

表F:各種機械学習手法の結果一覧

対象年			2020	2019	2018	2017	2016	2015
使用ICD-10バージョン			2013年版				2003年版	
機械学習用データ		対象レコード数	**319434**	**322559**	**317598**	**303141**	**292776**	**287358**
		ICDコード種類数 (Acme)	2061	2091	2066	2050	2006	2024
	共通ベクトルの次元数(ID, 正解を除く)		2085	2115	2090	2074	2030	2048
	付帯情報意味ベクトルの次元数							
		Embedding手法						
		BASELINE	0	0	0	0	0	0
		TFIDF	2938	2938	2938	2938		
		LSI	195	195	195	195		
		WORD2VEC	200	200	200	200		
		DOC2VEC (PV-DM)	200	200	200	200		
		DOC2VEC (PV-DBOW)	200	200	200	200		
	学習用ベクトルの次元数							
		BASELINE	2085	2115	2090	2074	2030	2048
		TFIDF	5023	5053	5028	5012		
		LSI	2280	2310	2285	2269		
		WORD2VEC	2285	2315	2290	2274		
		DOC2VEC (PV-DM)	2285	2315	2290	2274		
		DOC2VEC (PV-DBOW)	2285	2315	2290	2274		
学習結果 (予測精度)	評価尺度	Embedding手法						
	Accuracy (最良)	BASELINE	0.920	0.917	0.915	0.915	0.907	0.904
		TFIDF	**0.949**	**0.948**	**0.949**	**0.949**		
		LSI	0.945	0.944	0.944	0.945		
		WORD2VEC	0.948	0.942	0.942	0.943		
		DOC2VEC (PV-DM)	0.933	0.930	0.931	0.930		
		DOC2VEC (PV-DBOW)	0.944	0.941	0.942	0.943		
	ROC-AUC	BASELINE	0.925	0.923	0.922	0.921	0.927	0.928
		TFIDF	**0.953**	**0.949**	**0.951**	**0.950**		
		LSI	0.945	0.943	0.942	0.943		
		WORD2VEC	0.943	0.940	0.941	0.942		
		DOC2VEC (PV-DM)	0.933	0.929	0.932	0.930		
		DOC2VEC (PV-DBOW)	0.944	0.943	0.943	0.946		
	PR-AUC	BASELINE	0.742	0.727	0.735	0.735	0.793	0.791
		TFIDF	**0.857**	**0.847**	**0.859**	**0.859**		
		LSI	0.835	0.828	0.834	0.839		
		WORD2VEC	0.828	0.817	0.826	0.830		
		DOC2VEC (PV-DM)	0.782	0.771	0.785	0.780		
		DOC2VEC (PV-DBOW)	0.830	0.821	0.831	0.837		

【別添資料1】 本研究で構築したシステムの詳細

本研究で構築したシステムは下記の5つの処理ステップからなる。

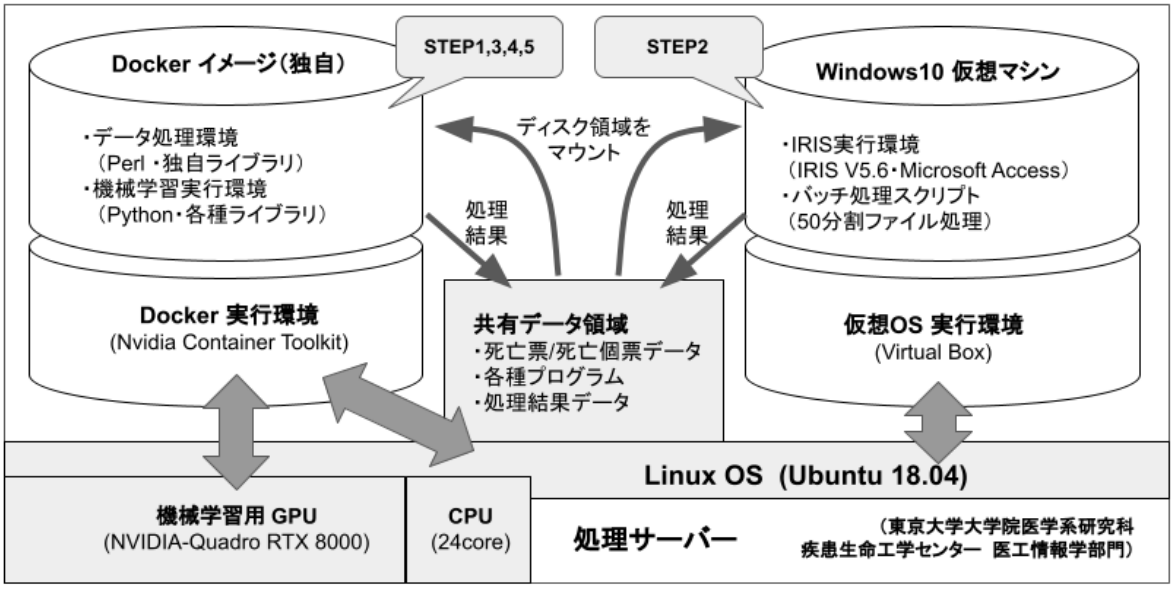
STEP1: 死亡票・死亡個票からの **IRIS** 入力用データの作成
STEP2: **IRIS** での仮原死因確定処理
STEP3: **IRIS** 処理結果の解析
STEP4: 機械学習用データセットの作成
STEP5: 各種機械学習での仮原死因変更有無予測モデルの構築

上記の処理はLinuxサーバ (Ubuntu 18.04) 上のDocker環境にて、本研究にて構築された Docker イメージを用いて行った。Docker環境が用意されているサーバであれば、同イメージを用いることで同様の処理がどのマシンでも可能である。

また STEP2 の部分だけは IRISを動作させる必要上 Windows 環境が必要であったため、Linux サーバー上に仮想OS環境 (Virtual Box) を構築し、その上で動作する Windows 10 仮想マシンにて処理を行った。同一マシンにて処理が完結するため、共有フォルダを介して、データのやり取りが可能であり、1台のサーバーにて完結するシステムである。

共有データ領域にあるプログラムをDockerイメージに梱包して含める構成も可能であり、この場合は実行させるための十分なスペックを持つマシンが有れば、同Dockerイメージを移植することで処理環境だけをどこでも再現することができ、可搬性が高い。一方、IRIS実行環境を含むWindows10仮想マシンの方も移動が可能であるが、OSライセンス (Windows) の問題から自由に共有することは難しい。

下記に構成図を示す。



（図1：システムの概要と処理サーバー内の構成）

以下各ステップでの処理の詳細を示す。

【STEP1】死亡票・死亡個票からの IRIS 入力用データの作成　(Linux サーバ)

■ 01_突合DB (ユニークキー) の作成

- [INPUT]
  - 統計法33上に基づき提供を受けた死亡票・死亡個票 (2015～2020: 6年分)
- [処理]
  - 死亡個票からは「処理年月、届出地、事件簿番号」
    死亡票からは「調査年、提出年月、届出地、事件簿番号」を用いて、両者の結合処理（JOIN）を行う (結果を「突合**DB**」と称する)
  - 「処理年月、届出地番号、事件簿番号」の組み合わせを各死亡案件のキーとした際に、複数回存在するものが存在（ヒアリングの結果、早期提出、また事件簿番号が9999までカウントアップするとまた0に戻る仕様になっていることに起因するもの）するため、これらは処理対象から除外
  - 結果として「処理年月、届出地番号、事件簿番号」の組み合わせで一意に定まるレコードのみを抽出、「突合**DB**（ユニークキー）」と称する。
  - これは、突合**DB**の **98.84%** に相当する。
- [OUTPUT]
  - 突合DB (ユニークキー)　”new_shibo_join.tsv”

■ 02_備考欄前処理

- [INPUT]
  - 突合DB (ユニークキー)
- [処理]
  - 死亡票・死亡個票の各欄からは、入力時の文字数制限により、入り切らない文字列が備考欄に溢れて記入されることがあるが、この「備考欄に溢れた文字列」を多くの正規表現ルールにより、元の然るべき項目へ可能な限り復元して結合する処理
  - 内容は、I欄II欄病名とそれぞれの期間、解剖・手術の詳細、傷害が発生したところ・手段及び状況、その他付言すべき事柄、生後1年未満での病死に関する詳細
  - 元の項目に復元できなかった文字列のみを「備考欄の文字列」として残す
- [OUTPUT]
  - 突合**DB (**ユニークキー・備考欄前処理後**)**
    - “03_new_shibo_join2.tsv”

■ 03_IRIS用入力データ作成

- [INPUT]
  - 突合**DB (**ユニークキー・備考欄前処理後**)**
    - “02_備考欄前処理/03_new_shibo_join2.tsv”
- [処理]
  - 処理年月日が指定された年のもののデータを読み込み
  - 期間表現の修正
  - 標準病名マスター (v5.04) と文字列処理を用いた独自の自動ICD-10コーディングを行う
  - 標準病名マスターのICD-10コードの中で、IRIS では用いられていないICD-10コードを置換する処理
    - 頻度上位のものにつき手動で置換ルールを作成
  - 全てにICD-10コードが当たった死亡票を対象に、生年、没年、性別等をIRISフォーマットに変換
  - IRIS 処理の速度向上のため、50ファイルのIdent, MedCod テーブルに分割して出力

    - 統計情報の出力
        - 「突合DB（ユニークキーのみ）: new_shibo_join2.tsv」の各データ件数のカウント
        - 突合DBで使用された原死因コード、外因コード、母側病態コードの一覧
        - 自動ICDコーディングの結果付与されたICDコードの一覧
- [OUTPUT]
    - IRIS 処理に回せた仮名IDのリスト
        - 03_IRIS用入力データ作成/Result/$year/50分割/ForIris.txt
    - IRIS サーバーへ結果をエクスポート

【STEP2】IRIS での仮原死因確定処理　(Virtual Box 上 仮想 Windows10 マシン)

- **IRIS** バッチ処理（分割ファイル分全て）
    - 50分割されたテキストファイルをAccessDB へインポート
    - コマンドラインでの IRIS バッチ処理
    - 処理結果のテキストデータをLinuxとの共有データ領域へ吐き出し
      (TestIdent1～50)

【STEP3】IRIS 処理結果の解析　(Linuxサーバ)

■ 04_IRIS 処理結果の解析

- [INPUT]
    - Iris 処理用マシンからIris処理結果をインポート
    - Irisの処理結果 (TestIdent1～50.txt) を読み込み
- [処理]
    - Iris処理結果内の不正なセル内改行コード (<CR>) の除去
    - 分割された結果の統合
    - 統計情報の出力（年ごと）（統計情報 / stat${year}.dat)
        - UCcode (8列目) 件数
        - MainInjury (9列目) 件数
        - UC, MainInjury の両方存在
        - IRIS 処理結果 (Final, Initial, Reject) 件数
        - IRIS Reject の内訳件数
- [OUTPUT]
    - IRIS処理結果（TestIdentKekka_${year}.txt）
      （IRIS 処理結果の年別まとめファイル）

■ 05_解析用統合テーブルの作成

- [INPUT]
    - 突合**DB (**ユニークキー・備考欄前処理後**)**
        - 02_備考欄前処理/03_new_shibo_join2.tsv)
    - IRIS へ入力した仮名死亡票ID
        - 03_IRIS用入力データ作成/Result/$year/50分割/ForIris.txt
    - IRIS処理結果
        - 04_IRIS処理結果/TestIdentKekka _${year}.txt
- [処理]

- 死亡票データから各付帯情報の記載の有無テーブルを作成
  - [Script] 1_parseShiboJoin.pl
  - [OUTPUT]
    - Results/Material/${year}/FORIRIS_withCodeHutai.txt
- IRIS処理結果と死亡表データ（突合DBユニークキー）を合わせた「統合テーブル」を作成
  - IRIS の処理結果を読み込み、死亡票由来情報(上記)と突合
  - IRIS による仮原死因コードを修正
    - IRIS では死因符号・外因符号の両方が存在する場合、外因符号を原死因として選択し、国内と逆であるため。
  - [Script] 2_parseIrisOutput.pl
  - [OUTPUT]
    - IRIS処理結果と死亡表データ（突合DBユニークキー）を合わせた「統合テーブル」(50列)
    - Results/${year}/Testkekka_Irekae.txt
- 「統合テーブル」のコード修正
  - IRIS仮原死因コード、国内原死因コードの粒度を合わせるための修正を頻度が多いものを対象に可能な限り行う。
    - IRIS仮原死因コード
      - IRIS側の方が粒度が細かい (桁が多い) ので落とす処理
      - 25ルール
    - 国内原死因コード
      - 国内の方が粒度が細かい (桁が多い) ので落とす処理
        - (例): A048A => A048
      - 原死因には用いない分類のコードの修正
        - (例): C77-C79 => C80
        - 「e-Stat－人口動態統計－分類表－2019年－9死因基本分類表」の「備考」参照
      - 研究期間の途中で準拠するICD-10のバージョンが変更されたことに起因する不一致
        - 平成28年度以前は ICD-10 2003年版
        - 平成29年度以降は ICD-10 2013年版
        - 元データから削除する
      - 397ルール

  - 両者の修正を終えた後、一致/不一致 (0/1) を再度修正
  - [Script] 3_modifyCodes.pl
  - [OUTPUT]
    - **IRIS**処理結果と死亡表データ（突合**DB**ユニークキー）を合わせた「統合テーブル」(50列, コード修正済み)
    - 詳細仕様は「表**C**」参照
    - Results/${year}/BasicTable.txt

- 統計情報の出力
  - 付帯情報の有無 x 仮原死因変更の有無のクロス集計表

- [OUTPUT]
  - **IRIS**処理結果と死亡表データ（突合**DB**ユニークキー・備考欄前処理後）を合わせた「統合テーブル」（**BasicTable)**
    - ${year}/BasicTable.txt

【STEP4】機械学習用データセットの作成　(Linuxサーバ)

■ 06_機械学習用基本データ

- [INPUT]
  - 統合テーブル（Step3で作成）
- [処理]
  - IDに対する共通ベクトルを作成
    - AGE, 性別 (要素数: 2)
    - 使用された ICD10コードに対する数値ベクトル (要素数: 可変)
      - 使用された全ICD10コードを桁数として、使用されたICDコードの所だけ数値が入力されているもの。
      - 桁数は IRIS 出力結果の ACME コードを解析した結果、用いられていた全ICDコードの種類数として別途計算
        - ACMEコードとは、IRIS 内部での修正処理が加わった、I欄・II欄病名に対応するICD10コードセット
        - IRIS が出力する Identテーブルの11列目に該当。
        - IRISが仮原死因コードと選択したものは、必ず「1」
        - その他のコードは「I欄のエ→ア、II欄病名」という優先順位で 0.85, 0.7, 0.55, 0.4, 0.25, 0.1 と「0.15 ずつ減算する形」で傾斜スコアリング。
        - 使用されていないICDコードは「0」。
      - 使用する年数が増えると、出現するICDコードは変化する。
        - 詳細は「表★:各種機械学習手法の結果一覧」を参照
    - 付帯情報の項目の有無 (要素数: 22)
      - TestKekka_syuusei.txt の 27～48列目
  - ID, 共通ベクトル, 仮原死因からの変更の有無(1/0, 正解データ) を組み合わせた学習用基本データを作成
- [OUTPUT]
  - 学習用基本データ
    - RESULT/learningData_${year}.txt
    - 2015, 2016,... 2020 年までの各年データと、2017-2020 年 (ICD-10 2013年版準拠)の通算データを作成

■ 07_付帯情報Embedding

- [INPUT]
  - 学習用基本データ
- [処理]
  - 各種 Embedding 手法により、付帯情報の文字列の内容をベクトル（分散表現）へ変換
  - 学習用基本データと結合し、疎行列表現として出力
- [OUTPUT]
  - 各種Embedding手法によるXGBoost学習用データ
    - BASELINE:
    - TFIDF
    - LSI
    - WORD2VEC
    - DOC2VEC (PV-DM)
    - DOC2VEC (PV-DBOW)

【STEP5】各手法での仮原死因変更有無予測モデルの学習　(Linuxサーバ)

■ 08_機械学習

- STEP4 までで作成された各種Embedding手法による学習用データに対し、勾配ブースティング決定木 (XGBoost) にて「付帯情報を考慮した上で、仮原死因変更有無を予測する」モデルを学習
- BERT による同予測モデルは、別途事前実験にて検証、結果精度がBASELINEを大きく改善しないことから、本実験からは外しているため割愛
    - 詳細は「別添資料2：**BERT**を用いた予測モデルの学習実験」参照
- 結果は「表**F**：各種機械学習手法の結果一覧」参照

【別添資料２】BERT を用いた予測モデルの学習実験

1. はじめに

分類器の精度向上のためには、付帯情報の各項目に対する記載の有無だけでなく、その意味内容を用いることが重要である。本研究では全ての手法に共通して機械学習用の共通ベクトル（年齢、性別、使用された ICD-10 コード、各付帯情報の項目の記載の有無）を用いているが、これに「付帯情報の文字列の意味内容」のベクトル表現を加えることにより、精度の向上が期待できる。

本実験では、近年深層学習ベースの自然言語処理の各種タスクにおいて従来手法の精度に優ったことから注目されている言語モデル獲得手法である BERT を用いて予測精度が向上するか試行を行った。

2. BERT とは

BERT(Bidirectional Encoder Representations from Transformers)は 2018 年に Google から発表された汎用言語モデル獲得のための手法である。Transformer という要素技術を元にしており、特定の領域のコーパス（大量の文章ソース）から別途の正解づけを必要とせず自動的に言語モデルが学習できることが大きな特徴である。このモデルは多くの自然言語処理に汎用的に用いることができるモデルであると言われており、一度獲得した BERT モデルを他のタスクに転用することが可能である。

そのため、一般的な使い方としては、何らかのコーパスにて学習された BERT モデルを（必要に応じてさらに追加で事前学習を行い）、その上で特定のタスクを解くための Deep Neural Network アーキテクチャに組み込んでファインチューニングする（重みを変更する）という方法が用いられる。本研究でもこれら一般的な方法に則って行った。

3. 対象データ

本実験では比較のために、昨年度ベースラインモデルとして開発した**「使用された病名 ICD-10 コードと、付帯情報の項目の有無」だけから仮原死因の変更の有無を XGBoost を用いて予測するモデル**の学習に用いたものと同じデータを用いた。

これは、まず突合 DB からランダムに抽出された 50 万件のうち、「全ての病名に ICD コードが付与でき、IRIS 処理が完了したもの」320,008 件を対象とし、さらに「何らかの付帯情報が存在するもの」81,688 件を実験用に抽出したデータである。

すでに昨年度、「年齢、性別、使用された ICD-10 コード（1553 列）、各付帯情報（２４列）」の計 1577 個の変数（**共通ベクトル**）を用いて、「確定原死因は、IRIS が決定し

た仮原死因から変更があるか否か」を勾配ブースティング決定木の一種である XGBoost を用いて学習し、付帯情報の意味内容を用いていないにも関わらず、Accuracy 90.3% を実現している。本年度では、付帯情報の意味内容まで考慮することで、この精度をさらに向上させるべく、前述の BERT モデルを用いた手法の検討を行った。

4. 用いた BERT ベースの学習手法

BERT によって言語モデルを獲得するためには、ドキュメント内に複数文書が存在する、という形のコーパスが必要である。例えば Wikipedia における日本語解説記事などが該当する。これは BERT の学習が、2 つの文章を入力として与え、マスクされた語を復元するタスクと共に、2 文の前後関係を解く、というタスクを同時に学習させる仕組みであることによるものである。

当初、突合 DB における「付帯情報の文字列」を用いて BERT モデルを学習させることを考えたが、死亡個票内の付帯情報は短い記述が非常に多く、複数文章が書かれているケースが十分に集まらないため、突合 DB を用いた BERT モデルの学習は断念し、既に一般に公開されている日本語 Wikipedia から学習済みである BERT モデルをそのまま利用することとした。日本語 Wikipedia は広く一般的な文章を含んでいるため、元となる言語モデルとしては、適切と考えられた。

本研究で用いた DNN（Deep Neural Network）のアーキテクチャは、**「共通ベクトル(1577 次元)」と「BERT モデルに付帯情報の文字列を通した結果得られるベクトル(768 次元)」を上位の全結合層で統合し、最終的に 2 値分類の結果を得る**もので、他のタスクでも非常に良く用いられるモデルである。しかし、実際の学習モデルの構築上は複数の考慮点がある。

1） 安定した結果を得るために学習率をどのように変化させるか
   - これについては学習率の減衰方法を複数施行した。

2） インバランスデータへの対応
   - 「1：変更あり」が「0：変更なし」に比べて圧倒的に多いインバランスデータであることから、データオーギュメンテーション、あるいは Class Weight を用いた重みづけ学習を試行した。

これらを踏まえ、本研究では 8 種類の手法を試し、結果を比較した。
以下にその詳細を記す。

A） RedLR

- 学習データはそのまま使用
- 学習率を 0.0006 で開始し、2 エポック改善が無ければ学習率を 1/10 にする。

B） BalRedLR

- データオーギュメンテーションを行ったもの。

学習データのうち結果が 1(変更あり)のデータを複製し、結果が 0(変更なし)のデータと同数にしている。

- 学習率を 0.0006 で開始し、2 エポック改善が無ければ学習率を 1/10 にします。

C) BalRedLR_BertOnly

- 共通ベクトルを用いず、付帯情報の文字列だけから、BERT のみで変更予測モデルを作成したもの。(付帯情報文字列以外の 1577 次元の共通ベクトルを削除)
- 学習データのうち結果が 1 のデータを複製し、結果が 0 のデータと同数にしている。
- 学習率を 0.0006 で開始し、2 エポック改善が無ければ学習率を 1/10 にする。

D) BalRedLR_Norm

BERT からの出力結果テンソルを BatchNormalization で正規化した後、共通ベクトル（付帯情報以外の 1577 次元）と結合したもの。

- 学習データのうち結果が 1 のデータを複製し、結果が 0 のデータと同数にしています。
- 学習率を 0.0006 で開始し、2 エポック改善が無ければ学習率を 1/10 にします。

E) CwRedLR

- 学習データはそのまま使用
- インバランスへの対応のため、データオーギュメンテーションを用いるのではなく、class_weight を指定し、学習時のクラスごとの重みづけを変更したもの。({0: 1.0、 1: 4.3})
- 学習率を 0.0008 で開始し、2 エポック改善が無ければ学習率を 1/10 する。

F) CyLR

- 学習データはそのまま使用。
- 学習率を 0.0001 から 0.0008 までの範囲で上下させ、サイクルさせたもの。

G) BalCyLR

- 学習データのうち結果が 1 のデータを複製し、結果が 0 のデータと同数にしています。
- 学習率を 0.0001 から 0.0008 までの範囲で上下させ、サイクルさせています。

H) CwCyLR

- 学習データはそのまま使用。
- class_weight を指定し、学習時のクラスごとの重みづけを変更。
  ({0: 1.0、 1: 4.3})
- 学習率を 0.0001 から 0.0008 までの範囲で上下させ、サイクルさせたもの。

5. 結果

上記 8 種類の手法での「仮原死因の変更有無」の予測結果を下記表 1 に示す。
（本報告書中の**表 E** と同じである）

表 1：各学習手法の予測精度一覧

手法		RedLR	BalRedLR	BalRedLR_BertOnly	BalRedLR_Norm	CwRedLR	CyLR	BalCyLR	CwCyLR
Accuracy		**0.914**	0.866	0.819	0.863	0.877	0.900	0.858	0.882
F1-score		**0.591**	0.535	0.391	0.548	0.543	0.564	0.510	0.549
Precision		**0.756**	0.481	0.343	0.475	0.517	0.641	0.457	0.537
Recall		0.485	0.602	0.455	**0.648**	0.571	0.504	0.576	0.561
Confusion Matrix (正解, 予測)	(0, 0)	13918	12890	12422	12750	13130	13656	12817	13235
	(0, 1)	328	1356	1824	1496	1116	590	1429	1011
	(1, 0)	1078	833	1140	737	897	1038	888	918
	(1, 1)	1014	1259	952	1355	1195	1054	1204	1174

6. 考察とまとめ

当初の予想では､BERT モデルによって｢付帯情報の文字列の意味｣を分散表現に変換し、上位層で共通ベクトルと合わせることで、共通ベクトル単体での学習に比べ大幅に精度が向上すると考えていたが、予想に反して昨年度のベースライン手法 （共通ベクトル単体での XGBoost、Accuracy 90.3%）と大差ない結果となった。

唯一 RedLR は Accuracy がベースラインに比べ 1%程度向上したが、後述する他の手法（TFIDF, LSI, WORD2VEC, DOC2VEC 等）での分散表現埋め込み（Embedding）を用いた方が効果が高く、期待した程ではない。また RedLR 以外の手法は全て昨年度のベースライン手法を下回る成績となっている。**このことから、以降の実験では、BERT 以外の方法で本実験を行うこととした**（XGBoost-Embed の各手法に絞った)。

また BalRedLR_BerOnly が示す通り､「共通ベクトル」の情報を全く使わず、付帯情報の文字列だけから判断したものは大幅に精度が低下していたことから、共通ベクトルの重要性が明らかとなった。

尚、前述の通り、死亡個票に書かれている付帯情報（手術の部位及び所見、解剖の部位及び所見、手段及び状況、その他付言すべき事柄、備考欄等）の文字列は短い記述が非常に多く、文章読解よりも寧ろ、特定の単語の出現の有無を端的に検出する方が有効である可能性が高い。そのため今回のタスクでは BERT の有効性が十分に発揮できなかったと考えられた。

厚生労働科学研究費補助金(政策科学総合研究事業（統計情報総合研究事業））
「死因統計の精度及び効率性の向上に資する機械学習の検討に関する研究」
分 担 研 究 報 告 書（令和 3 年度）

死亡に関わる調査票情報提供に基づいた ICD10 コード自動付与ツールの作成

研究分担者　香川璃奈　（筑波大学医学医療系・講師）

研究要旨

我が国において人口動態調査は国勢調査と並ぶ国の基幹統計であり、中でも死因統計は最も重要な情報の一つである。診療報酬請求や現在普及が進む電子カルテでは標準病名の採用が進められているが、人口動態調査の死因は自由入力病名が元となっており完全な自動集計は困難である。

本研究では、死因確定作業において目視確認に回る理由の中でも、死因を ICD-10 コードに変換できないという点に焦点をあてている。昨年度までに、標準病名と完全一致しない死因の記載を標準病名または ICD-10 コードに変換するルールを作成した上で、そのルールを利用することで、80.06%の症例に ICD10 コードを自動付与できたことを確認した。

本年度はこの結果を利用して、機械学習手法の開発を行った。機械学習を利用する際には、死因の自然文記載をベクトル化、すなわち数学的に利用可能な分散表現に変換する必要がある。ベクトル化の手法として doc2vec を利用したところ、付帯情報の記載によって確定原死因 ICD-10 コードが IRIS による仮原死因コードから変化する症例を、約 94%の正確性で同定できた。

A．研究目的

我が国において人口動態調査は国勢調査と並ぶ国の基幹統計であり、中でも死因統計は最も重要な情報の一つである。診療報酬請求や現在普及が進む電子カルテでは標準病名の採用が進められているが、人口動態調査の死因は自由入力病名が元となっており完全な自動集計は困難である。

我々は令和元年度に平成 27 年～平成 30 年の死亡票とオンライン申請された死亡個票の調査票情報の結合を行なった。結合した情報のことを、以下、突合死亡票 DB(データ数：5,169,031 件)と呼ぶ。これを利用して、標準病名マスターを用いて、全ての Ⅰ欄・II 欄病名に対しほぼ原記載のまま、また助詞、接続詞の除去/展開と言い換えなどの比較的簡便な文字列処理を施すことで、約 65%のⅠ欄・II 欄病名の自動 ICD10 コーディングが可能であるという感触を得た。さらに、Ⅰ欄・II 欄病名を ICD10 コードに変換できたものは約 9 割であった。さらに令和 2 年度は実際の死亡票に記載された病名を ICD10 コードに変換するツールを作成した。独自の対応ルールを利用することで、Ⅰ欄・II 欄病名のすべての自然言語記載病名に ICD10 コードに変換できた件数が全体の 8 割を超えた。

そこで令和 3 年度は、付帯情報に記載されている自然言語記載の内容を数学的に扱うために文(文章)を分散表現で表す手法を利用して、機械学習による死因の ICD10 コード自動付与の精度が変化するか確認した。

さらに、付帯情報が記載されていないにも関わらず iris が付与した原死因と確定原死因が異なる症例が生じる原因を考察することが、将来的な自動での死因 ICD10 コード自動付与ツールの開発に有用な可能性がある。そこで、付帯情報が記載されていないにも関わらず iris が付与した原死因と確定原死因が異なる症例について検討を行なった。

B．研究方法

【実験 1】

自然言語記載の内容を数学的に扱うために文(文章)を分散表現で表す手法を、今後「ベクトル化」と呼ぶ。ベクトル化手法による機械学習精度の違いを検討した。

ベクトル化手法として doc2vec(pvdm)(以下、単に doc2vec と呼ぶ)を利用にした際に、付帯情報の有無によって確定原死因が原死因から変更になるかどうかの 2 値分類を実施し、精度を検討した。

doc2vec の実施プログラムは別添 1 の通りである。

＜＜doc2vec とは＞＞

Doc2vec は任意の長さの文章を固定長のベクトルに変換する技術である。文中の語順や前後に出現する単語を加味したベクトル化が可能である。すなわち、単語をベクトル化した結果の組み合わせだけでは表現できない文章の特徴を表現できる。

Doc2vec にはベクトルに変換するために必要な学習における手法の違いにより 2 種類の手法が存在する[1]。本研究ではそれぞれ利用して結果を比較した。

(1) doc2vec(pv-dm)

文書の id と複数の単語(すなわち、文脈)から直後に続き単語を予測することで文書ベクトルを学習する。

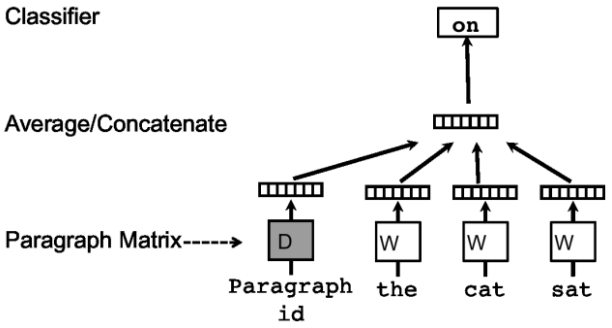


図 1：doc2vec(pv-dm) における学習のイメージ図。 [1]より転載。

(2) doc2vec(pv-dbow)

文書 id のみを入力として、語順を無視して文書に含まれる単語を予測するための学習を行う。単語ベクトル列を学習しないため学習速度が速いとされている。

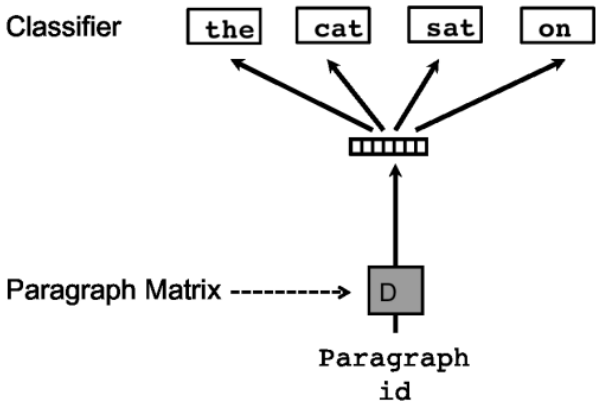


図 2：doc2vec(pv-dbow) における学習のイメージ図。 [1]より転載。

＜＜doc2vec で得られた分散表現を用いた 2 値分類器の学習＞＞

上記 doc2vec で得られた付帯情報に対する分散表現と、共通ベクトル（「性別・年齢、病名の ICD コード、付帯情報の項目の有無」）を結合し、xgboost にて「Iris が決定した仮原死因が変更されるか否か」を予測する 2 値分類器の学習を行った。この詳細については本年度「統括研究報告書」を参照されたい。

【実験 2】

iris が付与した原死因と確定原死因が異なる症例が全部で 17,337 例(iris が付与した原死因と確定原死因の ICD1-コードの組み合わせのユニーク数としては 3,134 種類)存在した。iris が付与した原死因と確定原死因の組み合わせの出現回数の上位 29 件(該当症例 100 例以上の組み合わせ)を目視で確認した。

【実験 1 および 2 に共通すること】

倫理面への配慮

本研究では統計法 33 条に基づき申請したデータを利用した。申請の通り、インターネットに繋がない端末上でのみデータの閲覧作業を行うことで個人情報に配慮した。

C．研究結果

【実験 1】

精度は以下の通りであった。

表 1：ベクトル化手法の違いに基づく機械学習精度の違い。

評価尺度	Embedding 手法	2020	2019	2018	2017
Accu-racy	DOC2VEC (PV-DM)	0.933	0.930	0.931	0.930
	DOC2VEC (PV-DBOW)	0.944	0.941	0.942	0.943
ROC-AUC	DOC2VEC (PV-DM)	0.933	0.929	0.932	0.930
	DOC2VEC (PV-DBOW)	0.944	0.943	0.943	0.946
PR-AUC	DOC2VEC (PV-DM)	0.782	0.771	0.785	0.780
	DOC2VEC (PV-DBOW)	0.830	0.821	0.831	0.837

この結果から、doc2vec(pv-dm)と doc2vec(pv-dbow)はいずれも高い精度を示すこと、doc2vec(pv-dbow)の方が僅かに高い精度を示すことを確認できた。

【実験 2】

iris が付与した原死因と確定原死因の ICD10 コードの間の関係性を、以下の 6 通りに分類した。

(A)医学的にはほぼ同義

(B)確定原死因が iris 原死因の下位概念（病変部位が特定されている、など）

(C)iris 原死因が確定原死因の下位概念（病変部位が特定されている、など）

(D)確定原死因が原因となって iris 原死因が生じたと想定される症例

(E)iris 原死因が原因となって確定原死因が生じたと想定される症例

(F)その他

なお以下の表 1 における iris 原死因の自然言語記載病名は各 ICD10 コードについて死亡表において出現回数が最も多い記載を中心として選択したものである。

この結果から、iris 原死因と確定原死因が異なる場合に、iris 原死因と全く異なる確定原死因が付与される事例はほぼ存在しないことを確認できた。

また表 1 に記載されている事例について、コーディングマニュアルに記載されている、原死

因コーディングのための注や連鎖表も確認したが、考察に活用できる知見を見つけることはできなかった。

表 2：iris が付与した原死因と確定原死因の ICD10 コードが異なる組み合わせ

確定原死因	Iris 原死因	該当個数	確定原死因名	Iris 原死因名	関係性の分類
K566	K567	707	Ｓ状結腸狭窄症	亜イレウス	B
I639	I638	504	虚血性脳卒中	出血性脳梗塞	F
C240	C248	460	下部胆管癌	#N/A	F
J189	F03	382	急性肺炎	原発性認知症	E
G309	G301	376	アルツハイマー型認知症	アルツハイマー型老年認知症	A
J189	J440	328	急性肺炎	下気道感染を伴う慢性閉塞性肺疾患	C
I252	I258	299	陳旧性下壁心筋梗塞	冠状動脈炎	E
I693	I639	264	小脳梗塞後遺症	虚血性脳卒中	B
J189	G20	251	急性肺炎	一側性パーキンソン症候群	E
J449	J440	245	慢性閉塞性肺疾患	下気道感染を伴う慢性閉塞性肺疾患	C
R99	空白	237	原因不明の死亡	#N/A	F
J189	G301	221	急性肺炎	アルツハイマー型老年認知症	E
E149	E146	202	糖尿病・糖尿病性合併症なし	高血糖高浸透圧症候群	D
I638	I635	193	出血性脳梗塞	延髄梗塞	C
J849	J841	168	間質性肺炎	炎症後肺線維症	D
I639	I635	163	虚血性脳卒中	延髄梗塞	C
J189	J439	153	急性肺炎	萎縮性肺気腫	D
J690	F03	152	胃分泌物嚥下性肺炎	原発性認知症	E
E142	E147	150	キンメルスチール・ウイルソン症候群	#N/A	F
I219	I258	147	ＳＴ上昇型急性心筋梗塞	冠状動脈炎	E
I500	J189	146	右室不全	急性肺炎	F
A319	B948	135	非結核性抗酸菌症	ジフテリア後麻痺	
R54	I509	135	老化	急性心不全	D
G318	G239	129	ＨＤＬＳ	基底核変性症	F
C220	K746	122	肝癌	萎縮性肝硬変	E
C928	C920	121	骨髄異形成関連変化を伴う急性骨髄性白血病	ＲＡＥＢ－ｔ	C
C220	B182	111	肝癌	Ｃ型肝炎	E
N189	I509	111	慢性腎臓病	急性心不全	D or E
S065	I620	100	外傷性硬膜下水腫	若年性慢性硬膜下血腫	C

D．考察

実験 1 の結果から、付帯情報により、iris が付与した原死因名と確定原死因が異なるかを機械学習で分類する問題について、doc2vec(pv-dbow)が有効であることを確認した。

また、実験 2 については、将来的に、このようなインストラクションマニュアルに記載されていないコーディングのルールが明文化されることで、自動での死因 ICD10 コード自動付

与ツールの開発にも有用であると考える。

E．結論

本年度研究では、付帯情報により、iris が付与した原死因名と確定原死因が異なるかを機械学習で分類する問題について、doc2vec が有効であること、また pv-dm と pv-dbow の２種類では pv-dbow の方が有効であることを確認した。

F．健康危険情報

なし

G．研究発表

なし

H．知的財産権の出願・登録状況

なし

参考文献

[1]Le, Quoc, and Tomas Mikolov. ″Distributed representations of sentences and documents.″ International conference on machine learning. PMLR, 2014.

本研究で用いたプログラムソースは、本報告書全体の添付資料「**機械学習用データセット作成プログラムソース Doc2Vec (PV-DM / PV-DBOW)**」を参照されたい。

III. 研究成果の刊行に関する一覧表

書籍

著者氏名	論文タイトル名	書籍全体の編集者名	書籍名	出版社名	出版地	出版年	ページ
なし							

雑誌

発表者氏名	論文タイトル名	発表誌名	巻号	ページ	出版年
大井川仁美, 今井　健, 香川璃奈, 明神大也, 今村知明	原死因決定プロセスの効率化に資する機械学習による原死因コード変更予測	医療情報学	41(Suppl.)	797-800	2021

別添資料

本統括・分担研究報告書全体に係る別添資料を示す。
順に、以下の構成となっている。

- 備考欄前処理プログラムソース（主要な部分抜粋）
  - applyregexp.py　（以下の処理の実行）
    - regexp14.py　　（”死亡の原因”　用）
    - regexp16.py　　（”外因死の追加事項”用）
    - regexp18.py　　（”その他特に付言すべきことがら”用）
    - regexpOther.py　（上記以外の全て用）

- 機械学習用データセット作成プログラムソース
  - TFIDF
  - LSI
  - Word2Vec
  - Doc2Vec (PV-DM / PV-DBOW)

- 分類器学習プログラムソース
  - fit_and_predict_xgboost.py

備考欄前処理プログラム (apply_regexp.py)

```
from my_util import exec_time

from datetime import datetime
import re
import regexp14
import regexp16
import regexp18
import regexpOther
import copy

@exec_time.printer
def main():
  regs14 = regexp14.get()
  regs16 = regexp16.get()
  regs18 = regexp18.get()
  regsOther = regexpOther.get()
  max_cnt = 0
  fis = open('./01_new_shibo_join_kv.csv', 'r')
  fos = open('./02_new_shibo_join_kv2' + ( '' if max_cnt <= 0 else '_' + str(max_cnt) ) + '.csv', 'w')
  f_removed = open('./02_new_shibo_join_removed_words' + ( '' if max_cnt <= 0 else '_' + str(max_cnt) ) + '.tsv', 'w')

  cnt = 0
  for sbh in sbh_data_generator(fis):
    cnt += 1
    if cnt == max_cnt: break
    rowid = sbh['rowid']
    values = sbh['value']
    # 14,1: 死亡原因　Iア死因　14,2: 死亡原因　Iア期間　14,3: 死亡原因　Iイ死因　14,4: 死亡原因　Iイ期間　14,5: 死亡原因　Iウ死因　14,6: 死亡原因　Iウ期間　14,7: 死亡原因　Iエ死因　14,8: 死亡原因　Iエ期間　14,9: 死亡原因　II死因　14,10: 死亡原因　II期間　14,12: 死亡原因 手術　14,14: 死亡原因 手術年月日　14,16: 死亡原因 解剖　16,4: 障害が発生したところ その他　16,8: 手段及び状況　17,1: 生後１年未満 詳細　18,1: その他付言　99,1: 備考
    # for sbhnum, sbhnum2 in [('14', '12'), ('14', '16'), ('16', '4'), ('16', '8'), ('17', '1'), ('18', '1'), ('99', '1')]:
    # 18,1: その他付言　99,1: 備考
    for sbhnum, sbhnum2 in [('18', '1'), ('99', '1')]:
      col = copy.deepcopy(values[sbhnum][sbhnum2])
      for k in col.keys():
        if k=='colnum': continue
        value = col[k]
        if len(value):
          org_value = value
          # 14用の正規表現でパース
          for r in regs14:
            # 14の区切り位置を[col14]で置換する
            t = fw_reg(r[0]).subn(r[1], value)
            if t[1]: value = t[0]
          # 16用の正規表現でパース
          for r in regs16:
            # 16の区切り位置を[col16]で置換する
            t = fw_reg(r[0]).subn(r[1], value)
            if t[1]: value = t[0]
          # 18用の正規表現でパース
          for r in regs18:
            # 18の区切り位置を[col18]で置換する
            t = fw_reg(r[0]).subn(r[1], value)
            if t[1]: value = t[0]
          # 上記以外用の正規表現でパース
          for r in regsOther:
            # 区切り位置を[colxx_0]で置換する
            t = fw_reg(r[0]).subn(r[1], value)
            if t[1]: value = t[0]
          # スプリッタ（[colxx]）で切り分け
          value_arr = fw_reg(r'(\[col[^\]]+\])([^(\[$)]*)').findall(value)
          if value_arr:
            len14_1 = len(values['14']['1']) - 1
            len14_2 = len(values['14']['2']) - 1
            len14_3 = len(values['14']['3']) - 1
            len14_4 = len(values['14']['4']) - 1
            len14_5 = len(values['14']['5']) - 1
            len14_6 = len(values['14']['6']) - 1
```

```
            len14_7 = len(values['14']['7']) - 1
            len14_8 = len(values['14']['8']) - 1
            len14_9 = len(values['14']['9']) - 1
            len14_10 = len(values['14']['10']) - 1
            len14_12 = len(values['14']['12']) - 1
            len14_14 = len(values['14']['14']) - 1
            len14_16 = len(values['14']['16']) - 1
            len16_4 = len(values['16']['4']) - 1
            len16_8 = len(values['16']['8']) - 1
            len18_1 = len(values['18']['1']) - 1
            # OPTIMIZE: いったん組み直して正規表現化することでスプリッタを復元する
            # NOTE: 行ごとに固有の正規表現を組むためfw_regを使用しない。
            splited_value_regexp = '(' + fw_reg(r'\[col[^\]]+\]').sub(r')
(.*?', value) + ')'
            splited_value_arr = re.findall(splited_value_regexp, org_value)
            for i, (spliter, v) in enumerate(value_arr):
              # 連結前の整形
              tmp = fw_reg(r'^(は)?「(.*)」(である)?(。)?$').search(v)
              if tmp: v = tmp.groups()[1]
              v = fw_reg(r'( (続く) )?\n?$').sub('', v)
              if spliter == '[col18_top]':
                sbh['value']['18']['1'][len18_1] = v
                len18_1 += 1
              elif spliter == '[col18]':
                sbh['value']['18']['1'][len18_1 + 100] = v
                len18_1 += 1
              #elif spliter == '[col16_4]':
              # sbh['value']['16']['4'][len16_4] = v
              # len16_4 += 1
              elif spliter == '[col16]' or spliter == '[col16_8]':
                sbh['value']['16']['8'][len16_8] = v
                len16_8 += 1
              elif spliter == '[col14_1]':
                sbh['value']['14']['1'][len14_1] = v
                len14_1 += 1
              elif spliter == '[col14_2]':
                sbh['value']['14']['2'][len14_2] = v
                len14_2 += 1
              elif spliter == '[col14_3]':
                sbh['value']['14']['3'][len14_3] = v
                len14_3 += 1
              elif spliter == '[col14_4]':
                sbh['value']['14']['4'][len14_4] = v
                len14_4 += 1
              elif spliter == '[col14_5]':
                sbh['value']['14']['5'][len14_5] = v
                len14_5 += 1
              elif spliter == '[col14_6]':
                sbh['value']['14']['6'][len14_6] = v
                len14_6 += 1
              elif spliter == '[col14_7]':
                sbh['value']['14']['7'][len14_7] = v
                len14_7 += 1
              elif spliter == '[col14_8]':
                sbh['value']['14']['8'][len14_8] = v
                len14_8 += 1
              elif spliter == '[col14_9]':
                sbh['value']['14']['9'][len14_9] = v
                len14_9 += 1
              elif spliter == '[col14_10]':
                sbh['value']['14']['10'][len14_10] = v
                len14_10 += 1
              elif spliter == '[col14_12]':
                sbh['value']['14']['12'][len14_12] = v
                len14_12 += 1
              elif spliter == '[col14_14]':
                sbh['value']['14']['14'][len14_14] = v
                len14_14 += 1
              elif spliter == '[col14_16]':
                sbh['value']['14']['16'][len14_16] = v
                len14_16 += 1
              else: # ここまでに合致しないスプリッタの文言は捨てる
                f_removed.write(rowid + '\t' + spliter + '\t' +
```

```
splited_value_arr[0][i+1].rstrip('\n') + '\n')
            # 切り分けた先頭は元の欄に残す
            try:
              sbh['value'][sbhnum][sbhnum2][k] = fw_reg(r'^([^\
[]+)').search(value).group() + '\n'
            except AttributeError:
              sbh['value'][sbhnum][sbhnum2][k] = '\n'
    sbh_data_writer(fos, sbh)

  fis.close()
  fos.close()
  f_removed.close()

def sbh_data_generator(fis):
  l = fis.readline()
  bef_rowid = l.split(',')[0]
  sbh = {'rowid': bef_rowid, 'value': {}}
  while l!='':
    rowid, colnum, sbhnum, sbhnum2, colrownum, value = l.split(',')
    if bef_rowid != rowid:
      yield sbh
      sbh = {'rowid': rowid, 'value': {}}
    # キーを変更してdictにする
    # sbh.update({'rowid': rowid, 'value': {'sbhnum': sbhnum, 'value':
{'sbhnum2': sbhnum2, 'sbhnum': sbhnum, 'value': {sbhrownum: value }}}})
    # sbh.update({'rowid': rowid, 'value': {sbhnum: {sbhnum2: {'colnum':
colnum, colrownum: value }}}})
    if not sbhnum in sbh['value']: sbh['value'][sbhnum] = {}
    if not sbhnum2 in sbh['value'][sbhnum]: sbh['value'][sbhnum][sbhnum2] =
{'colnum': colnum}
    sbh['value'][sbhnum][sbhnum2][colrownum] = value
    bef_rowid = rowid

    l = fis.readline()
  yield sbh

def sbh_data_writer(fos, sbh):
  # rowid     : new_shibo_join.tsvの先頭カラム、行ごとのID
  # colnum    : new_shibo_join.tsvのカラム番号
  # sbhnum    : 死亡診断書の欄番号。new_shibo_join.tsvには存在しない。
  # sbhnum2   : 死亡診断書の欄番号の枝番号。new_shibo_join.tsvには存在しない。
  # colrownuma: 死亡診断書の欄番号、枝番号ごとの内容が複数行に渡る場合の行番号。
  # value     : new_shibo_join.tsvの2カラム目以降の値。
  rowid = sbh['rowid']
  out_arr = []
  for sbhnum, sbhnum_col in [(str(sbhnum), sbh['value'][str(sbhnum)]) for
sbhnum in sbh['value'].keys()]:
    if sbhnum == 'rowid': continue
    for sbhnum2, sbhnum2_col in [(sbhnum2, sbhnum_col[sbhnum2]) for sbhnum2
in sbhnum_col.keys()]:
      colnum = sbhnum2_col['colnum']
      for colrownum, value in [(colrownum, sbhnum2_col[colrownum]) for
colrownum in sbhnum2_col.keys()]:
        if colrownum == 'colnum': continue
        out_arr.append([rowid, colnum, sbhnum, sbhnum2, str(colrownum),
value])
  out_arr.sort(key=lambda item: (int(item[1]), int(item[4])))

  bef_colnum = out_arr[0][1]
  l = ','.join(out_arr[0])
  for l_arr in out_arr[1:]:
    if bef_colnum != l_arr[1]:
      fos.write(fw_reg(r'( (続く) )?\n?$').sub('', l) + '\n')
      l = ','.join(l_arr)
      bef_colnum = l_arr[1]
    else:
      l = l.rstrip('\n') + l_arr[5]
  fos.write(l)

fw_reg_cache = {}
def fw_reg(p):
  """
  初めて取得するパターンはコンパイルして返す。
```

```
  初めてでないパターンはキャッシュから返す。
  """
  try:
    return fw_reg_cache[p]
  except KeyError:
    r = re.compile(p)
    fw_reg_cache[p] = r
    return r


if __name__ == '__main__':
  main()
```

備考欄前処理プログラム　(regexp14.py)

```
def get():
  return [[r'（?ｌ４）?([ーーー「(])?(ｌ|1|I|i)?([「(])?(ア)+([」）])?(の|　|）)?(死因)+(欄|ラン|らん)?(続き|つづき)?(は)?([　、。・:，ーー）])?',r'[col14_1]'],
    [r'（?ｌ４）?([ーーー「(])?(ｌ|1|I|i)?([「(])?(ア)+([」）])?(死亡までの|の|　)?(期間|年月日|年月|年|月日|月|日)+(欄|ラン|らん)?(続き|つづき)?(は)?([　、。・:，ーー）])?',r'[col14_2]'],
    [r'（?ｌ４）?([ーーー「(])?(ｌ|1|I|i)?([「(])?(イ)+([」）])?(の|　|）)?(死因)+(欄|ラン|らん)?(続き|つづき)?(は)?([　、。・:，ーー）])?',r'[col14_3]'],
    [r'（?ｌ４）?([ーーー「(])?(ｌ|1|I|i)?([「(])?(イ)+([」）])?(死亡までの|の|　)?(期間|年月日|年月|年|月日|月|日)+(欄|ラン|らん)?(続き|つづき)?(は)?([　、。・:，ーー）])?',r'[col14_4]'],
    [r'（?ｌ４）?([ーーー「(])?(ｌ|1|I|i)?([「(])?(ウ)+([」）])?(の|　|）)?(死因)+(欄|ラン|らん)?(続き|つづき)?(は)?([　、。・:，ーー）])?',r'[col14_5]'],
    [r'（?ｌ４）?([ーーー「(])?(ｌ|1|I|i)?([「(])?(ウ)+([」）])?(死亡までの|の|　)?(期間|年月日|年月|年|月日|月|日)+(欄|ラン|らん)?(続き|つづき)?(は)?([　、。・:，ーー）])?',r'[col14_6]'],
    [r'（?ｌ４）?([ーーー「(])?(ｌ|1|I|i)?([「(])?(エ)+([」）])?(の|　|）)?(死因)+(欄|ラン|らん)?(続き|つづき)?(は)?([　、。・:，ーー）])?',r'[col14_7]'],
    [r'（?ｌ４）?([ーーー「(])?(ｌ|1|I|i)?([「(])?(エ)+([」）])?(死亡までの|の|　)?(期間|年月日|年月|年|月日|月|日)+(欄|ラン|らん)?(続き|つづき)?(は)?([　、。・:，ーー）])?',r'[col14_8]'],
    [r'（?ｌ４）?([ーーー「(])?(２|2|Ⅱ|ⅱ)+([」）])?(の|　)?(死因)+(欄|ラン|らん)?(続き|つづき)?(は)?([　、。・:，ーー）])?(傷病名)?',r'[col14_9]'],
    [r'（?ｌ４）?([ーーー「(])?(２|2|Ⅱ|ⅱ)+()?([」）])?(死亡までの|の|　)?(期間|年月日|年月|年|月日|月|日)+(欄|ラン|らん)?(続き|つづき)?(は)?([　、。・:，ーー）])?(傷病名)?',r'[col14_10]'],
    [r'（?ｌ４）?(欄中?)?([ーーー，、])?(の|　)?(解剖)+(欄|ラン|らん|　)?(主要所見|所見)?(の|　)?(続き|つづき)?(は)?([　、。・:，ーー）])?', r'[col14_16]'],
    [r'（?ｌ４）?(欄中?)?([ーーー，、])?(の|　)?(手術|手術部位及び主要所見追加)+(?!解剖|期間|年月日|年月|年|月日|月|日|施行日)(欄|ラン|らん)?(の|　)?(続き|つづき)?(は)?([　、。・:，ーー）])?', r'[col14_12]'],
    [r'（?ｌ４）?(欄中?)?([ーーー「(，、])?(の|　)?(手術期間|手術年月日|手術年月|手術年|手術月日|手術月|手術日|手術施行日)+(欄)?([」）])?(の|　)?(続き|つづき)?(は)?([　、。・:，ーー）])?', r'[col14_14]'],
    [r'(（ｌ４）)+(欄中?)?(の|　)?([ーーー(，、])?(期間の記載|死亡したとき)?(続き|つづき)?(は)?([　、。・:，ーー）])?',r'[col14]'],
    [r'（ｌ４）(ｌ|1|I|i|２|2|Ⅱ|ⅱ)?([ーーー，、])?(「(ア)直接原因」|「(イ)（ア）の原因」|「(ウ)（イ）の原因」|「(エ)（ウ）の原因」|「I欄に影響を及ぼした傷病名等」)?(の|　)?(「死亡までの期間」)+(　)?',r'[col14]'],
    [r'（?ｌ４）?(手術・解剖|手術解剖)+(　)?',r'[col14]'],
    [r'（?ｌ４ｌ(ｌ|２|３|４|５)?）?(欄|ラン|らん)?(続き|つづき)?(続)?(は)?([　、。・:，ーー）])?',r'[col14]']]#分類できないものはこのキーに置く
    #記号控え([　、。・:，ーー）])

def _assert(r, s):
  import re
  # [print(re.subn(rr[0], rr[1], s)[1]) for rr in r]
  print('開始[' + s + ']')
  result = ''
  for rr in r:
    t = re.subn(rr[0], rr[1], s)
    if t[1]:
      if len(result.replace('0', '')): print('経過[' + s + ']')
      s = t[0]
      result += '1'
    else:
      result += '0'
  # l = list(filter(lambda result: result[1] > 0, [re.subn(rr[0], rr[1], s) for rr in r]))
  print('結果[' + s + ']\n')
  # print([result[0] + '\n' for result in l])
  return len(result.replace('0', ''))


if __name__ == '__main__':
  r = get();

  print('開始 ----------==========')


  print('==========---------- 終了')

```

備考欄前処理プログラム　(regexp16.py)

```
def get():
  return [[r'（?１６）?(欄中?)?(　)?）?([の（])?(続き|続|つづき|追記)([　、・:，ーー
ー）])?', r'[col16]'],#１６の情報として抽出
    [r'（?１６）?(欄中?)?([ーー一の「])?(傷害|障害|死亡)?(発生|が発生した|した)+(日時|時
刻|時分|とき|時間)+(欄)?([の（）」])?(続き|続|つづき|追記|時分)?(は)?([　、・:，ーー
ー）])?', r'[col16_4]'],#傷害が発生した時刻として抽出
    [r'（?１６）?(欄中?)?([ーー一の「])?(傷害|障害|死亡)?(発生|が発生した|した)+(日|時)+
(欄)?([の（）」])?(続き|続|つづき|追記|)?(は)?([　、・:，ーー一）])?',
r'[col16_4]'],#傷害が発生した時刻として抽出
    [r'（１６）(欄中?)?(「)?(手段及び状況|状況|手段および状況)?(欄)?([の（」　])?(続き|
続|つづき|追記)?(において|は)?([　、・:，ーー一）])?', r'[col16_8]'],#手段及び状況とし
て抽出と結合
    [r'（?１６０５）?([の（])?(続き|続|つづき|追記)?([は）])?([　、・:，ーー一])?',
r'[col16_8]'],#手段及び状況として抽出と結合
    [r'「?１６枠外」?([　、・:，ーー一])?', r'[col16]'],#１６の情報として抽出
    [r'（１６追加）', r'[col16]'], #１６の情報として抽出
    [r'１６[　）：欄]', r'[col16]'],#１６の情報として抽出
    [r'１６(欄)?(ーー一)',r'[col16]'],#１６の情報として抽出
    [r'(外因死)?(の)?(追加事項の)?：?(手段及び状況)+(欄)?(について)?(の)?(続き|続|つづ
き|追記)?(は)?([　、。・:，ーー一）])?',r'[col16_8]']]#手段及び状況として抽出と結合

def _assert(r, s):
  import re
  # [print(re.subn(rr[0], rr[1], s)[1]) for rr in r]
  print('開始[' + s + ']')
  result = ''
  for rr in r:
    t = re.subn(rr[0], rr[1], s)
    if t[1]:
      if len(result.replace('0', '')): print('経過[' + s + ']')
      s = t[0]
      result += '1'
    else:
      result += '0'
  # l = list(filter(lambda result: result[1] > 0, [re.subn(rr[0], rr[1], s)
for rr in r]))
  print('結果[' + s + ']\n')
  # print([result[0] + '\n' for result in l])
  return len(result.replace('0', ''))


if __name__ == '__main__':
  r = get();

  print('開始 ----------==========')


  print('==========---------- 終了')
```

備考欄前処理プログラム　(regexp18.py)

```
def get():
  return [[r'１８続き、', r'[col18_top]'],
    [r'（?１８）?(欄)?(　)?）?([の（])?(続き|続|つづき|追記|：)([　、・:，ーー—）])?', r'[col18]'],
    [r'（?１８）?(欄)?([ーー—])?(その他|その他特に付言すべき事柄)(欄)?([の（）])?(続き|続|つづき|追記)?([　、・:，ーー—）])?', r'[col18]'],
    [r'（１８(欄)?）(欄)?([の（])?(続き|続|つづき|追記)?([　、・:，ーー—）])?', r'[col18]'],
    [r'（?１８０１）?([の（])?(続き|続|つづき|追記)?([　、・:，ーー—])?', r'[col18]'],
    [r'「?１８枠外」?([　、・:，ーー—])?', r'[col18]'],
    [r'（１８欄その他特に付言すべきことがら）', r'[col18]'],
    [r'１８その他特に付言すべきことがらの続きは、', r'[col18]'],
    [r'（１８追加）', r'[col18]'],
    [r'１８[　）；欄]', r'[col18]'], ]

def _assert(r, s):
  import re
  # [print(re.subn(rr[0], rr[1], s)[1]) for rr in r]
  print('開始[' + s + ']')
  result = ''
  for rr in r:
    t = re.subn(rr[0], rr[1], s)
    if t[1]:
      if len(result.replace('0', '')): print('経過[' + s + ']')
      s = t[0]
      result += '1'
    else:
      result += '0'
  # l = list(filter(lambda result: result[1] > 0, [re.subn(rr[0], rr[1], s) for rr in r]))
  print('結果[' + s + ']\n')
  # print([result[0] + '\n' for result in l])
  return len(result.replace('0', ''))


if __name__ == '__main__':
  r = get();

  print('開始 ----------==========')

  print('==========---------- 終了')
```

備考欄前処理プログラム (regexpOther.py)

```
def get():
  return [# （3）生年月日
    [r'（3）(欄|続き)?', r'[col03_0]'],
    [r'(?<!1)3(続き|続|つづき)[ 、・:，ーー]?', r'[col03_0]'],
    [r'（3(続き|続|つづき)）', r'[col03_0]'],
    # （4）死亡したとき
    [r'（4）死亡したとき[ 、・:，ーー]?', r'[col04_0]'],
    [r'（4）(欄|続き)?', r'[col04_0]'],
    [r'(?<!1)4(続き|続|つづき)[ 、・:，ーー]?', r'[col04_0]'],
    [r'（4(続き|続|つづき)）', r'[col04_0]'],
    # （5）死亡したところ
    [r'（5）(欄|続き)?', r'[col05_0]'],
    [r'(?<!1)5(続き|続|つづき)[ 、・:，ーー]?', r'[col05_0]'],
    [r'（5(続き|続|つづき)）', r'[col05_0]'],
    # （6）住所
    [r'（6）(欄|続き)?', r'[col06_0]'],
    [r'(?<!1)6(続き|続|つづき)[ 、・:，ーー]?', r'[col06_0]'],
    [r'（6(続き|続|つづき)）', r'[col06_0]'],
    # （7）本籍
    [r'（7）(欄|続き)?', r'[col07_0]'],
    [r'(?<!1)7(続き|続|つづき)[ 、・:，ーー]?', r'[col07_0]'],
    [r'（7(続き|続|つづき)）', r'[col07_0]'],
    # （8）（9）死亡した人の夫または妻
    [r'（8）(欄|続き)?', r'[col08_0]'],
    [r'(?<!1)8(続き|続|つづき)[ 、・:，ーー]?', r'[col08_0]'],
    [r'（8(続き|続|つづき)）', r'[col08_0]'],
    [r'（9）(欄|続き)?', r'[col09_0]'],
    [r'(?<!1)9(続き|続|つづき)[ 、・:，ーー]?', r'[col09_0]'],
    [r'（9(続き|続|つづき)）', r'[col09_0]'],
    # （10）（11）死亡したときの世帯の主な仕事と死亡した人の職業・産業
    [r'（10）(欄|続き)?', r'[col10_0]'],
    [r'10(続き|続|つづき)[ 、・:，ーー]?', r'[col10_0]'],
    [r'（10(続き|続|つづき)）', r'[col10_0]'],
    [r'（11）(欄|続き)?', r'[col11_0]'],
    [r'11(続き|続|つづき)[ 、・:，ーー]?', r'[col11_0]'],
    [r'（11(続き|続|つづき)）', r'[col11_0]'],
    # （12）（13）死亡したところ、及びその種別
    [r'（?12）?[ーー]施設名称?(続き|続|つづき)[ 、・:，ーー]?', r'[col12_0]'],
    [r'（12）死亡したところ[ 、・:，ーー]?', r'[col12_0]'],
    [r'（12）死亡したとき[ 、・:，ーー]?', r'[col12_0]'],
    [r'（12）(欄|続き)?', r'[col12_0]'],
    [r'12(続き|続|つづき)[ 、・:，ーー]?', r'[col12_0]'],
    [r'（12(続き|続|つづき)）', r'[col12_0]'],
    [r'（13）死亡したところ[ 、・:，ーー]?', r'[col12_0]'],
    [r'（13）死亡したとき[ 、・:，ーー]?', r'[col12_0]'],
    [r'（13）(欄|続き)?', r'[col13_0]'],
    [r'13(続き|続|つづき)[ 、・:，ーー]?', r'[col13_0]'],
    [r'（13(続き|続|つづき)）', r'[col13_0]'],
    #（17）生後1年未満で病死した場合の追加事項
    [r'（17）欄?欄妊娠・分娩時における母体の病態又は異状は?', r'[col17_0]'],
    [r'（17）(欄|続き)?', r'[col17_0]'],
    [r'17(続き|続|つづき)[ 、・:，ーー]?', r'[col17_0]'],
    [r'（17(続き|続|つづき)）', r'[col17_0]'],
  ]

def _assert(r, s):
  import re
  # [print(re.subn(rr[0], rr[1], s)[1]) for rr in r]
  print('開始[' + s + ']')
  result = ''
  for rr in r:
    t = re.subn(rr[0], rr[1], s)
    if t[1]:
      if len(result.replace('0', '')): print('経過[' + s + ']')
      s = t[0]
      result += '1'
    else:
      result += '0'
  # l = list(filter(lambda result: result[1] > 0, [re.subn(rr[0], rr[1], s)
for rr in r]))
  print('結果[' + s + ']\n')
  # print([result[0] + '\n' for result in l])
```

```
  return len(result.replace('0', ''))


if __name__ == '__main__':
  r = get();

  print('開始 ----------==========')

  print('==========---------- 終了')
```

機械学習用データセット作成プログラム (TF・IDF)

```
from datetime import datetime
from gensim import corpora
from gensim import models
from gensim import matutils
import MeCab
import math
import re
import sys
import os

def new_idf(docfreq, totaldocs, log_base=2.0, add=1.0):
    return add + math.log(1.0 * totaldocs / docfreq, log_base)

def arg_below():
    import argparse
    p = argparse.ArgumentParser()
    p.add_argument('-a', '--anybelow', help='単語出現回数下限の指定、未入力の場合
Noneで引数を取り、100で処理')
    a = p.parse_args()
    return a.anybelow

def main(any_below = 100, mode = 0):
    max_cnt = 0
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']開始しま
す。')
    #単語の出現回数の下限、Noneの場合は100
    below = int(100 if any_below == None else any_below)
    #print(below)

    #疎行列ベクトル出力先ディレクトリ作成
    dirname = ''
    if any_below != None:
        dirname = 'TFIDF'+ ( '' if below <= 0 else '_' + str(below) ) + '/'
        #os.makedirs(dirname,mode=511,exist_ok=True)

    #読み込みファイルと出力ファイル
    fis = open('./new_shibo_join_concat.tsv','r')
    fos = open('./' + dirname + '05' + ( '' if max_cnt <= 0 else '_' +
str(max_cnt) ) + '_sogyouretu.csv','w')
    fos2 = open('./' + dirname + '05' + ( '' if max_cnt <= 0 else '_' +
str(max_cnt) ) + '_total_wordnum.txt','w')

    #作成済み辞書の読み込み先リンク
    gdic_fname = './' + dirname + 'new_shibo_join.dict'

    #トーカナイザを取得
    tokenizer = get_tokenizer()

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']分かち書き開
始')
    #トークンリスト（複数行の文章、単語の二次元配列）の読み込み
    rownum_list,tokens_list = fetch_tokenslist(fis, tokenizer, max_cnt)
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']分かち書き終
了')

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']dictionary
作成開始')
    #トークンリストを辞書とするか、既存の辞書を読み込むか、モードで切り替え
    if mode == 0:
        #トークンリストを辞書変換
        tokens_gdic = tokenslist2dic(tokens_list)
        #辞書情報を保存
        #save_gdic(tokens_gdic, gdic_fname)
    elif mode == 1:
        #辞書情報の読み込み
        tokens_gdic = load_gdic(gdic_fname)

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']dictionary
作成終了')

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']corpus作成
開始')
    #辞書をbag-of-words形式のリスト（corpus）に変換
```

```
    tokens_corpus = doc2bow(tokens_gdic, tokens_list)
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']corpus作成終了')

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']model作成開始')
    test_model = models.TfidfModel(tokens_corpus,wglobal=new_idf)

    corpus_tfidf = test_model[tokens_corpus]
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']model作成終了')

    #0回出現の単語についても重要度0を出力する
    #new_dic = [{wordid:0 for word,wordid in dictionary.token2id.items()} for c in id_list]
    #辞書の総単語数を取得
    total_wordnum = 0
    for word,wordid in tokens_gdic.token2id.items():
        total_wordnum += 1
    fos2.write(str(total_wordnum))
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']ベクトル出力開始')
    for rowid,doc in zip(rownum_list,corpus_tfidf):
        lil =[]
        for word in doc:
            lil.append((word[0],word[1]))

        fos.write(data_make(rowid,lil))
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']ベクトル出力終了')

    fis.close()
    fos.close()
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']終了します。')

#出力の形式を整えるメソッド
def data_make(rowid,lil):
    return rowid + '\t['+','.join([sogyouretu(kv) for kv in lil])+']\n'

#tfidf値を疎行列の形にして返すメソッド
def sogyouretu(kv):
    return str(kv[0])+':'+str(kv[1])

    #for rowdic,rowid in zip(new_dic,id_list):
        #print(rowid+'\t',','.join([str(k)+','+str(v) for k,v in sorted(rowdic.items())]))
        #print(rowid+'\t',','.join([str(v) for k,v in rowdic.items()]))
        #fos.write(rowid+'\t'+','.join([str(v) for k,v in sorted(rowdic.items())])+'\n')

    """
    #出現0回の文字について重要度0を入れない出力
    texts_tfidf = []
    for rowid,doc in zip(id_list,corpus_tfidf):
        text_tfidf = []
        for word in doc:
            text_tfidf.append(word[1])
        texts_tfidf.append(text_tfidf)
        print(rowid+'\t',','.join([str(i) for i in text_tfidf]))
    """

#作成済みdictionaryを読み込むメソッド
def load_gdic(fname):
    return corpora.Dictionary.load(fname)

#作成したdictionaryを保存するメソッド
def save_gdic(tokens_gdic, fname):
    tokens_gdic.save(fname)

#形態素解析にMeCabを使用し、トーカナイザーを取得するメソッド
def get_tokenizer():
    return MeCab.Tagger('-r /etc/mecabrc -Owakati')
```

```
129
130 #解析データを読み込み分かち書きしてlistを作成するメソッド
131 def fetch_tokenslist(fis, tokenizer, max_cnt = 0):
132     cnt = 1
133     l = fis.readline();
134     rownum_list = []
135     tokens_list = []
136     while (cnt != max_cnt and l != ''):
137         rownum, sentences = remove_tagwords(l).split('\t')
138         rownum_list.append(rownum)
139         #トーカナイズ（1文章を単語ごとに分割した一次元配列）
140         tokens = tokenizer.parse(sentences).split()
141         #トークンリスト（複数行の文章、単語の二次元配列）
142         tokens_list.append(tokens)
143         cnt += 1
144         l = fis.readline();
145
146     return (rownum_list, tokens_list)
147
148 re_tagwords = re.compile(r'(【手術】)|(【解剖】)|(【状況】)|(【母体】)|(【付言】)|
(【備考】)|\n')
149 #解析データの項目名を単語から除去
150 def remove_tagwords(sentences):
151     return re_tagwords.sub('', sentences)
152
153 #トークンリストを辞書変換するメソッド
154 def tokenslist2dic(tokens_list):
155     return corpora.Dictionary(tokens_list)
156
157 #辞書をbag-of-words形式のリストに変換するメソッド
158 def doc2bow(tokens_gdic, tokens_list):
159     return list(map(tokens_gdic.doc2bow, tokens_list))
160
161 if __name__ == '__main__':
162     any_below = arg_below()
163     main(any_below, 1)
164
165
166
```

機械学習用データセット作成プログラム (LSI)

```
from datetime import datetime
from gensim import corpora
from gensim import models
from gensim import matutils
import MeCab
import math
import re
import os

def new_idf(docfreq, totaldocs, log_base=2.0, add=1.0):
    return add + math.log(1.0 * totaldocs / docfreq, log_base)

def main(mode = 0):
    max_cnt = 0
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']開始しま
す。')

    #疎行列ベクトル出力先ディレクトリ作成
    dirname = 'LSI/'
    os.makedirs(dirname,mode=511,exist_ok=True)

    #読み込みファイルと出力ファイル
    fis = open('./new_shibo_join_concat.tsv','r')
    fos = open('./'+ dirname + ( '' if max_cnt <= 0 else '_' + str(max_cnt) )
+ '05_sogyouretu.csv','w')
    fos2 = open('./'+ dirname + ( '' if max_cnt <= 0 else '_' + str(max_cnt)
) + '05_total_wordnum.txt','w')

    #作成済み辞書の読み込み先リンク
    gdic_fname = './new_shibo_join.dict'

    #トーカナイザを取得
    tokenizer = get_tokenizer()

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']分かち書き開
始')
    #トークンリスト（複数行の文章、単語の二次元配列）の読み込み
    rownum_list,tokens_list = fetch_tokenslist(fis, tokenizer, max_cnt)
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']分かち書き終
了')

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']dictionary
作成開始')
    #トークンリストを辞書とするか、既存の辞書を読み込むか、モードで切り替え
    if mode == 0:
        #トークンリストを辞書変換
        tokens_gdic = tokenslist2dic(tokens_list)
        #辞書情報を保存
        #save_gdic(tokens_gdic, gdic_fname)
    elif mode == 1:
        #辞書情報の読み込み
        tokens_gdic = load_gdic(gdic_fname)

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']dictionary
作成終了')

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']corpus作成
開始')
    #辞書をbag-of-words形式のリスト（corpus）に変換
    tokens_corpus = doc2bow(tokens_gdic, tokens_list)
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']corpus作成
終了')

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']model作成開
始')
    test_model = models.TfidfModel(tokens_corpus,wglobal=new_idf)

    corpus_tfidf = test_model[tokens_corpus]
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']model作成終
了')

    #0回出現の単語についても重要度0を出力する
    #new_dic = [{wordid:0 for word,wordid in dictionary.token2id.items()} for
```

```
c in id_list]
    #辞書の総単語数を取得
    """
    total_wordnum = 0
    for word,wordid in tokens_gdic.token2id.items():
        total_wordnum += 1
    fos2.write(str(total_wordnum))
    """
    #LSI modelによる次元圧縮したcorpusの作成
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']LSI開始')
    lsi_model = models.LsiModel(corpus_tfidf, id2word=tokens_gdic,
num_topics=200)
    lsi_model.save('lsi_topics200.model')
    #lsi_model = models.LsiModel.load('lsi_topics200.model')
    corpus_lsi = lsi_model[corpus_tfidf]
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']LSI終了')

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']ベクトル出力
開始')
    vec_size = 0
    for rowid,doc in zip(rownum_list,corpus_lsi):
        lil =[]
        for word in doc:
            lil.append((word[0],word[1]))

        if len(lil) != 0:
            vec_size = len(lil)
        fos.write(data_make(rowid,lil))
    fos2.write(str(vec_size))
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']ベクトル出力
終了')

    fis.close()
    fos.close()
    fos2.close()
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']終了しま
す。')

#出力の形式を整えるメソッド
def data_make(rowid,lil):
    return rowid + '\t['+','.join([sogyouretu(kv) for kv in lil])+']\n'

#tfidf値を疎行列の形にして返すメソッド
def sogyouretu(kv):
    return str(kv[0])+':'+str('{:f}'.format((kv[1]+1)/2))

    #for rowdic,rowid in zip(new_dic,id_list):
        #print(rowid+'\t',','.join([str(k)+','+str(v) for k,v in
sorted(rowdic.items())]))
        #print(rowid+'\t',','.join([str(v) for k,v in rowdic.items()]))
        #fos.write(rowid+'\t'+','.join([str(v) for k,v in
sorted(rowdic.items())])+'\n')

    """
    #出現０回の文字について重要度０を入れない出力
    texts_tfidf = []
    for rowid,doc in zip(id_list,corpus_tfidf):
        text_tfidf = []
        for word in doc:
            text_tfidf.append(word[1])
        texts_tfidf.append(text_tfidf)
        print(rowid+'\t',','.join([str(i) for i in text_tfidf]))
    """

#作成済みdictionaryを読み込むメソッド
def load_gdic(fname):
    return corpora.Dictionary.load(fname)

#作成したdictionaryを保存するメソッド
def save_gdic(tokens_gdic, fname):
    tokens_gdic.save(fname)

#形態素解析にMeCabを使用し、トーカナイザーを取得するメソッド
```

```
def get_tokenizer():
    return MeCab.Tagger('-r /etc/mecabrc -Owakati')

#解析データを読み込み分かち書きしてlistを作成するメソッド
def fetch_tokenslist(fis, tokenizer, max_cnt = 0):
    cnt = 1
    l = fis.readline();
    rownum_list = []
    tokens_list = []
    while (cnt != max_cnt and l != ''):
        rownum, sentences = remove_tagwords(l).split('\t')
        rownum_list.append(rownum)
        #トーカナイズ（1文章を単語ごとに分割した一次元配列）
        tokens = tokenizer.parse(sentences).split()
        #トークンリスト（複数行の文章、単語の二次元配列）
        tokens_list.append(tokens)
        cnt += 1
        l = fis.readline();

    return (rownum_list, tokens_list)

re_tagwords = re.compile(r'(【手術】)|(【解剖】)|(【状況】)|(【母体】)|(【付言】)|
(【備考】)|\n')
#解析データの項目名を単語から除去
def remove_tagwords(sentences):
    return re_tagwords.sub('', sentences)

#トークンリストを辞書変換するメソッド
def tokenslist2dic(tokens_list):
    return corpora.Dictionary(tokens_list)

#辞書をbag-of-words形式のリストに変換するメソッド
def doc2bow(tokens_gdic, tokens_list):
    return list(map(tokens_gdic.doc2bow, tokens_list))

if __name__ == '__main__':
    main(1)
```

機械学習用データセット作成プログラム (Word2Vec)

```
import sys
#sys.path.append('/home/bmiwork/ITEC')
from my_utils import exec_time

import MeCab
import re
from gensim.models import word2vec
from gensim import corpora
#from gensim import matutils
import numpy as np
import os

@exec_time.printer
def main(mode = 0):
    max_cnt = 0

    #疎行列ベクトル出力先ディレクトリ作成
    dirname = 'WORD2VEC/'
    os.makedirs(dirname,mode=511,exist_ok=True)

    #トーカナイザを取得
    tokenizer = get_tokenizer()

    #読み込みファイルと出力ファイル
    fis = open('./new_shibo_join_concat.tsv','r')
    fos = open('./'+ dirname + ( '' if max_cnt <= 0 else '_' + str(max_cnt) )
+ '05_sogyouretu.csv','w')
    fos2 = open('./'+ dirname + ( '' if max_cnt <= 0 else '_' + str(max_cnt)
) + '05_total_wordnum.txt','w')

    #作成済みmodelの出力先リンク
    #vec_fname = './new_shibo_join' + ( '' if max_cnt <= 0 else '_' +
str(max_cnt) ) + '.vec.pt'
    vec_fname = "./WORD2VEC/entity_vector.model.bin"

    #対象文書の形態素を格納
    rownum_list, tokens_list = fetch_tokenslist(fis, tokenizer, max_cnt)
    #print(tokens_list[0])

    if mode == 0:
        #word2vecのmodelに形態素を学習
        model = fit_word2vec(tokens_list)
        #modelを保存
        save_model(model, vec_fname)
    elif mode == 1:
        model = load_model(vec_fname)

    #各行の文書のベクトルを計算して出力
    write_vector(fos, fos2, model, rownum_list, tokens_list)

    fis.close()
    fos.close()
    fos2.close()

#word2vec学習モデルの読み込み
def load_model(fname):
    return word2vec.KeyedVectors.load_word2vec_format(fname, binary=True)

#word2vec学習モデルの保存
def save_model(model, fname):
    model.wv.save_word2vec_format(fname, binary=True)

@exec_time.printer
def write_vector(fos, fos2, model, rownum_list, tokens_list):
    #単語ベクトルを平均した値を文書ベクトルとして扱った場合の出力
    num_features = 200
    for rowid, doc in zip(rownum_list,tokens_list):
        #print(rowid)
        #print(doc[0])
        #単語を200次元のベクトルとして出力したもの
        """
        vec_list = []
        for word in doc:
```

```
            if word in model.key_to_index:
                vec = model[word]
                vec_list.append(vec)
        fos.write(data_make(rowid, doc, vec_list))
        """
        #docが空でない場合のみベクトルの平均を作成、空の場合は空のリストを作成
        if doc != []:
            feature_vec = np.zeros((num_features), dtype= "float32")
            for word in doc:
                if word in model.key_to_index:
                    feature_vec = np.add(feature_vec, model[word])
                    #feature_vec = np.add(feature_vec, model.get_vector(word,
norm=True))
            if len(doc) > 0:
                feature_vec = np.divide(feature_vec, len(doc))
        else:
             feature_vec = []
        #print(feature_vec)
        fos.write(data_make2(rowid, feature_vec, num_features))
    fos2.write(str(num_features))

#出力の形式を整えるメソッド
def data_make(rowid, doc, vec_list):
    return rowid + '\t['+ ','.join([sogyouretu(wordid,vec) for wordid, vec in
zip(doc, vec_list)])+']\n'

#出力の形式を整えるメソッド
def data_make2(rowid, feature_vec, vec_cnt):
    cnt_list = []
    for num in range(vec_cnt):
        cnt_list.append(num)
    if feature_vec != []:
        return rowid + '\t['+ ','.join([sogyouretu2(v,i) for v,i in
zip(feature_vec, cnt_list)])+']\n'
    else:
        return rowid + '\t[]\n'

#ベクトルを疎行列形式に整えるメソッド
def sogyouretu(wordid, vec):
    return str(wordid)+':'+str(vec)

#ベクトルを疎行列形式に整えるメソッド
def sogyouretu2(vec, cnt):
    return str(cnt)+':'+str('{:f}'.format((vec+10)/20))
    #return str(cnt)+':'+str(vec)

@exec_time.printer
def get_tokenizer():
    #形態素解析にMeCabを使用する
    return MeCab.Tagger('-r /etc/mecabrc -Owakati')

@exec_time.printer
def fetch_tokenslist(fis, tokenizer, max_cnt = 0):
    cnt = 1
    l = fis.readline()
    rownum_list = []
    tokens_list = []
    while (cnt != max_cnt and l != ''):
        rownum, sentences = remove_tagwords(l).split('\t')
        #if(sentences != ''):
        rownum_list.append(rownum)
        #トーカナイズ（1文章を単語ごとに分割した一次元配列）
        tokens = tokenizer.parse(sentences).split()
        #ストップワードの除去
        tokens = filter_stopwords(tokens)
        #トークンリスト（複数の文章、単語の二次元配列）
        tokens_list.append(tokens)
        cnt += 1
        l = fis.readline();
    return (rownum_list, tokens_list)

re_tagwords = re.compile(r'(【手術】)|(【解剖】)|(【状況】)|(【母体】)|(【付言】)|
(【備考】)|\n')
```

```
def remove_tagwords(sentences):
    return re_tagwords.sub('', sentences)

stopword_list = None
def filter_stopwords(tokens):
    global stopword_list
    if stopword_list is None:
        stopword_list = []
        #ストップワードの読み込み
        fis = open('./stopwords.txt','r')
        re_comment = re.compile(r'^\s*(#.*|)$')
        l = fis.readline()
        while l:
            if not re_comment.match(l):
                stopword_list.append(l.rstrip('\n'))
            l = fis.readline()
        fis.close()
    return [word for word in tokens if word not in stopword_list]

@exec_time.printer
def tokenslist2dic(tokens_list):
    return corpora.Dictionary(tokens_list)

@exec_time.printer
def fit_word2vec(tokens_list):
    #word2vecによる単語の学習
    return word2vec.Word2Vec(tokens_list)

if __name__ == '__main__':
    main(1)


```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

#=================================================#
#==                                             ==#
#==                 [embedding.py]              ==#
#==                   version.1.1               ==#
#==                   2022/03/28                ==#
#==                                             ==#
#=================================================#

import pandas as pd
import numpy as np
import argparse
from argparse import RawTextHelpFormatter
import ast
import MeCab
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
import tensorflow_hub as hub
import tensorflow_text
import re
import os

import ssl
ssl._create_default_https_context = ssl._create_unverified_context

# Embeddingクラス作成
class Embedding(object):
  def __init__(self, rawdata_path, delete_symbol, params, mode, output_path):
    self.rawdata_path = rawdata_path      # 入力ファイルのパス
    self.delete_symbol = delete_symbol    # 形態素解析時に削除する記号
    self.params = params                        # doc2vecを使用する際の引数(辞書
型)
    self.mode = mode
    self.output_path = output_path

  def check_mode(self):
    '''
    modeの確認。doc2vecを使用する際は、modeとparams['dm']が一致していないとエラーとな
る。
    params['dm']を設定していない場合は、modeがparams['dm']の引数となる
    '''
    if self.mode==2:
      print('mode: Universal Sentence Encoder')

    elif self.mode==0 or self.mode==1:
      if 'dm' not in self.params:
        self.params['dm']=self.mode
        if self.mode==0:
          print('mode: doc2vec PV-DBOW')
        elif self.mode==1:
          print('mode: doc2vec PV-DM')

      elif 'dm' in self.params:
        if self.params['dm'] == self.mode:
          if self.mode==0:
            print('mode: doc2vec PV-DBOW')
          elif self.mode==1:
            print('mode: doc2vec PV-DM')
        else:
          raise KeyError('modeとparams["dm"]が一致していません. mode: {},
params["dm"]: {}'.format(self.mode, self.params['dm']))

    else:
      raise ValueError('--mode 0~2から選んでください {0: doc2vec PV-DBOW, 1:
doc2vec PV-DM, 3: Universal Sentence Encoder}')


  def get_data(self):
    '''
    rawdataの読み込み。入力ファイルは一列目にid、二列目にテキストデータを含む、hedderおよ
びindex_colの無い.tsvファイル。
    '''
```

```
    # 入力ファイル形式が異なる場合は、read_csv()の引数設定を変更する
    src = pd.read_csv('{}'.format(self.rawdata_path), delimiter='\t',
index_col=None, header=None, names=['id', 'data'])
    # テキストデータがNaNのサンプル行を削除
    src_data = src.dropna(how='any')

    sentences = []
    for text in src_data['data']:
        # テキスト内の任意の記号を削除
        text = re.sub(r'{}'.format(self.delete_symbol), '', text)

        # 各行のテキストを１要素(リスト)としてsetencesに二次元リストとして格納
        text_list = text.split(' ')
        sentences.append(text_list)

    return src, sentences

  def doc2vec(self, sentences):
    '''
    doc2vecによるベクトル抽出。Mecab を用いて形態素解析した後、モデルを作成する。
    '''
    token = []
    tokernizer = MeCab.Tagger('-Owakati')     # Mecabの形態素解析の引数設定

    # setences内の各要素(テキスト)を形態素解析
    for s in sentences:
      token.append(tokernizer.parse(s[0]).split())

    # 以下処理過程の参考
    # 例 src_data.iloc[0,1] : 【付言】ＤＮＡ検査にて本人確認。
    # 例 sentences[0] : ['付言ＤＮＡ検査にて本人確認']
    # 例 token[0] : ['付言', 'ＤＮＡ', '検査', 'にて', '本人', '確認']

    # tokenをdoc2vecに読み込ませる形式に変換
    documents = [TaggedDocument(doc, [i]) for i, doc in enumerate(token)]

    #hash関数の指定 モデルの再現性を得るためにハッシュ値を固定
    import hashlib
    hashfxn = lambda x: int(hashlib.md5(str(x).encode()).hexdigest(), 16)

    # doc2vecモデル構築
    model = Doc2Vec(documents,hashfxn=hashfxn, **self.params)
    vectors = [model.docvecs[i] for i in range(len(sentences))]

    return vectors

  def UniversalSentenceEncoder(self, sentences):
    '''
    Universal Sentence Encoderによるベクトル抽出。形態素解析は行わずモデルを構築し、
512次元ベクトルを出力する。
    '''
    # tensorflow hubから学習済みモデルの読み込み
    USE_model = hub.load(
    "https://tfhub.dev/google/universal-sentence-encoder-multilingual/3")  #
2022/01時点, latest version:はver.3
    vectors_all = USE_model(sentences)
    vectors = [i for i in vectors_all]

    return vectors

  def output(self, src, vectors):
    '''
    id とベクトルを結合し、出力用データフレームを作成。output_pathが"None"の場合はファイ
ル出力されない。
    '''
    src_data = src.dropna(how='any')
    src_data_vec = src_data.assign( vecs = vectors)
    df = pd.merge(src, src_data_vec, how="left", on = "id")
    df = df.loc[:, ['id','vecs']]

    # output_pathを任意指定した時のみファイルを出力する
    if self.output_path=='None':
        pass
```

```
    else:
        output_path = self.output_path
        df.to_csv(output_path, sep='\t', header=False, index=False)

    return df

def main():
  parser = argparse.ArgumentParser(prog='embedding',
                       description='----------------------------------------
----------------------------------\n'
                       'PROGRAM NAME:下流工程を考慮したテキストの特徴抽出プログラム\n'
                       '概要: 自然言語(日本語)で記載されたテキストが持つ特徴を抽出するプ
ログラム。\n'
                       'doc2vecまたはuniversal setence encoderを用いてベクトル化す
る。\n'
                       '_____________________________________________________
----------------------\n',
                       usage='$ python3 embedding.py [-m] [-rp] [-ds] [-
params] [-op]\n',
                       formatter_class=RawTextHelpFormatter,
                       add_help=True)
  parser.add_argument('--mode', '-m', type=int, required=True,
                          help='モデルの選択\n'
                          'mode:{0: doc2vec PV-DBOW, 1: doc2vec PV-DM, 2:
Universal Sentence Encoder}')
  parser.add_argument('--rawdata_path', '-rp', type=str, required=True,
                          help='入力ファイルのパス. \n'
                          '入力ファイルは一列目にid、二列目にテキストデータを含む、
hedderおよびindex_colの無い.tsvファイル')
  parser.add_argument('--delete_symbol', '-ds', type=str, default='[【】、。
「」（）]',
                          help='形態素解析時の削除したい記号\n'
                          '削除したい記号をカッコ内に入れ[ ]、str型で渡す\n'
                          'e.g. [PROGRAM] --delete_symbol "[【】、。「」（）]"')
  parser.add_argument('--doc2vec_params', '-params', type=str, default="
{'vector_size':2}",
                          help='doc2vecに与える引数をstr型で入力する. 引数はモジュー
ル内で辞書型に変換される.\n'
                          '''e.g. [PROGRAM] -params "{'vector_size' : 2,
'window': 2, 'min_count': 1, 'epochs': 10}"''')
  parser.add_argument('--output_path', '-op', type=str, default='None',
                        help='出力ファイルのパス. \n'
                        '出力ファイルは一列目にid、二列目にベクトルデータを含む、
hedderおよびindex_colの無い.tsvファイル\n'
                        '引数を与えない、または"None"を与えるとファイルは出力されな
い')
  args = parser.parse_args()

  # 引数受け取り
  mode = args.mode
  rawdata_path = args.rawdata_path
  delete_symbol =args.delete_symbol
  params = ast.literal_eval(args.doc2vec_params)
  output_path = args.output_path

  # インスタンス作成
  E = Embedding(rawdata_path, delete_symbol, params, mode, output_path)
  # modeとparamsの確認
  E.check_mode()

  # 入力データ(rawdata)読み込み
  src, sentences = E.get_data()

  # doc2vecによる実行
  if mode==0 or mode==1:
    vectors = E.doc2vec(sentences)
  # USEによる実行
  elif mode==2:
    vectors = E.UniversalSentenceEncoder(sentences)

  # 出力用データフレーム作成
  df = E.output(src, vectors)
```

```
198 if __name__ == '__main__':
199   main()
```

分類器学習プログラム (fit_and_predict_xgboost.py)

```
import pandas as pd
import xgboost as xgb
import numpy as np
import pickle
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from matplotlib import pyplot as plt
from sklearn.metrics import confusion_matrix, roc_curve,
precision_recall_curve, auc
from sklearn.model_selection import GridSearchCV
from scipy.sparse import lil_matrix
from datetime import datetime
import sys
import os

def arg_parse():
    import argparse
    p = argparse.ArgumentParser()
    p.add_argument('-a', '--anyfolder', help='疎行列ベクトル参照先フォルダ')
    p.add_argument('-g', '--gpuid', help='使用するGPUのID（デフォルトは0')
    p.add_argument('-y', '--year', help='学習用データ年')
    a = p.parse_args()
    return (a.anyfolder, a.gpuid, a.year)

def main(any_folder='.', gpuid=0, year=2020):
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] ' +
__file__  + ' start')

    # 引数の検証
    if not any_folder: any_folder = '.'
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] 入力元フォ
ルダ[../07_付帯情報Embedding/' + any_folder + ']')
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] 出力先フォ
ルダ[./' + any_folder + '/RESULT/' + year + '/]')
    #os.makedirs('./RESULT/' + any_folder, exist_ok=True)
    # 疎行列行列数ファイル読み込み
    # rows = 81688   # 行数
    # cols = 1579 + 2736  # 次元数
    fis = open('../07_付帯情報Embedding/' + any_folder +
'/learningData_smatrix_rowcolnum_' + year + '.txt', 'r')
    rows = int(fis.readline())
    cols = int(fis.readline())
    fis.close()
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] 対象行数
[' + str(rows) + '] 対象ベクトル数[' + str(cols) + ']')

    # 疎行列ファイルを読み込み
    smatrix_fname = '../07_付帯情報Embedding/' + any_folder +
'/learningData_smatrix_' + year + '.txt'
    X, y = load_sparse_matrix(smatrix_fname, rows, cols)
    # test_size=0.2 全体のうち0.8を学習用、0.2をテスト用に分割する
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
shuffle=True, random_state=42, stratify=y)

    # 学習用の0.8で学習
    # NOTE: tree_method、gpu_idの設定をすることでGPUが使用されるようになる。
    # ----- Best Estimator
    clf = xgb.XGBClassifier(base_score=0.5, booster='gbtree',
colsample_bylevel=1,
                colsample_bynode=1, colsample_bytree=1, eta=0.25, gamma=0,
                gpu_id=gpuid, importance_type='gain',
interaction_constraints='',
                learning_rate=0.25, max_delta_step=0, max_depth=14,
                min_child_weight=0.05, monotone_constraints='()',
                n_estimators=2000, n_jobs=1, num_parallel_tree=1,
random_state=0,
                reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
                tree_method='gpu_hist', use_label_encoder=False,
                validate_parameters=1, verbosity=None, eval_metric='logloss')
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] fit
start')
    clf.fit(X_train, y_train)
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] fit
```

```
end')

    # モデルを保存
    pickle.dump(clf, open('./' + any_folder + '/' + year + '/model.pkl',
'wb'))
    clf = pickle.load(open('./' + any_folder + '/' + year + '/model.pkl',
'rb'))

    # テスト用の0.2で検証
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] predict
start')
    # pred = clf.predict(X_test)
    pred_p = clf.predict_proba(X_test)
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] predict
end')

    # 結果出力
    report = []
    for i in range(21):
        t = (i/20)
        pred = (pred_p[:, 1] > t).astype(int)
        print('-----===== 閾値[' + str(t) + '] ==========')
        accuracy = print_accuracy(y_test, pred)
        tp, fp, fn, tn, precision, recall = print_confusion_matrix(y_test,
pred)
        report.append([t, accuracy, tn, fp, fn, tp, precision, recall, tp+fp,
(tp+fp)/(tp+fp+fn+tn)])
        write_id_tf(X[:, 0], y_test, pred, t, any_folder, year)

    if any_folder == 'TFIDF_100':
        print_importances(clf, any_folder, year)

    y_score = pred_p[:,1]
    make_pr_curve_pict(y_test, y_score, any_folder, year)
    make_roc_curve_pict(y_test, y_score, any_folder, year)

    write_report(report, any_folder, year)
    write_pred_score(X[:, 0], y_test, pred_p[:, 1], any_folder, year)

    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] ' +
__file__  + ' end')

def load_sparse_matrix(fname, rows, cols):
    # 疎行列ファイルをオープン
    fis = open(fname, 'r')

    X_lil = lil_matrix((rows, cols - 1), dtype=float)
    # y_lil = lil_matrix((rows, 1), dtype=float)
    y = [0] * rows

    # 疎行列ファイルの読み込み
    l = fis.readline()
    rownum = 0
    while l != '':
        l_smatrix = sogyoretu2smatrix(l.rstrip('\n'))
        for k, v in l_smatrix.items():
            if k != cols - 1:
                if len(v) != 0:      # DOC2VEC 疎行列データエラー対策
                    X_lil[rownum, k] = float(v)
                else:
                    print("[ERROR]: '' can not be converted to float @" +
str(rownum))
                    X_lil[rownum, k] = 0
            else:
                ## y_lil[rownum, 0] = float(v)
                y[rownum] = float(v)

        l = fis.readline()
        rownum += 1

    fis.close()

    # sparseからnumpy.ndarrayに変換
```

```
    X = X_lil.toarray()
    # y = y_lil.toarray()
    return X, y

def sogyoretu2smatrix(sogyoretu):
    """
    01_sogyouretu.csvの疎行列表記文字列を疎行列のdictionaryに変換する。
    """
    # 先頭末尾の[]を外す
    # ,でスプリットしkvセットに切り分る
    # :でスプリットしてkeyとvalueを分ける
    return {int(kv.split(':')[0]): kv.split(':')[1] for kv in
sogyoretu[1:-1].split(',') if kv != ''}

def print_accuracy(act, pred):
    """
    精度を出力する。
    """
    acc = accuracy_score(act, pred)
    print('----- Accuracy')
    print(acc)
    return acc

def print_confusion_matrix(act, pred):
    """
    混合配列を出力する。
    """
    cm = {}
    cm[(0,0)] = 0
    cm[(0,1)] = 0
    cm[(1,0)] = 0
    cm[(1,1)] = 0
    for t, p in zip(act, pred):
        cm[(t, p)] += 1
    print('----- Confusion Matrix')
    print('(act, pred): ', cm)
    print('----- Confusion Matrix')
    print(pd.DataFrame(confusion_matrix(act, pred, labels=[0, 1]), columns=
["pred_0", "pred_1"], index=["act_0", "act_1"]))

    p, r = print_pr(cm[(1,1)], cm[(0,1)], cm[(1,0)], cm[(0,0)])
    return (cm[(1,1)], cm[(0,1)], cm[(1,0)], cm[(0,0)], p, r)

def print_pr(tp, fp, fn, tn):
    p = tp/(tp+fp) if (tp+fp)!=0 else 1
    print('----- Precision')
    print(str(p))
    r = tp/(tp+fn) if (tp+fn)!=0 else 1
    print('----- Recall')
    print(str(r))
    return (p, r)

def print_importances(clf, any_folder, year):
    """
    重要度を出力する。
    """
    import os
    from gensim import corpora

    dict_fname = '../07_付帯情報Embedding/' + any_folder +
'/new_shibo_join.dict'
    tokens_dict = None
    if os.path.exists(dict_fname) and os.path.getsize(dict_fname):
        tokens_dict = corpora.Dictionary.load(dict_fname)

    icdcode_fname = '../06_機械学習用基本データ/ICDList/acme_kari_code_sort_' +
year + '.txt'
    icdcodes = ['certificateKey', '年齢', '性別']
    fis = open(icdcode_fname, 'r')
    lines = 0
    l = fis.readline()
    while(l != ''):
        icdcodes.append(l.rstrip('\n'))
```

```
        l = fis.readline()
        lines += 1

    fis.close()

    icdcodes.append('手術フラグ(1)')
    icdcodes.append('手術の部位及び所見(116)')
    icdcodes.append('（手術）備考欄への記載(1)')
    icdcodes.append('手術日(8)')
    icdcodes.append('解剖フラグ(1)')
    icdcodes.append('解剖の部位及び所見(116)')
    icdcodes.append('（解剖）備考欄への記載(1)')
    icdcodes.append('死因の種類(2)')
    icdcodes.append('傷害が発生したとき(8)')
    icdcodes.append('傷害が発生したとき(5)')
    icdcodes.append('傷害が発生したところの種別(1)')
    icdcodes.append('傷害が発生したところその他の記述(40)')
    icdcodes.append('傷害発生場所(8)')
    icdcodes.append('傷害発生場所(12)')
    icdcodes.append('傷害発生場所(18)')
    icdcodes.append('手段及び状況(120)')
    icdcodes.append('（傷害）備考欄への記載(1)')
    icdcodes.append('「生後1年未満での病死」の病態・異状の詳細(84)')
    icdcodes.append('備考欄への記載(1)')
    icdcodes.append('その他付言すべき事柄(60)')
    icdcodes.append('備考欄外字有無(1)')
    icdcodes.append('備考欄(1024)')

    lines += 25
    imps = sorted(enumerate(clf.feature_importances_), key=lambda kv: -kv[1])
    print('----- Feature Importances')
    for k, v in imps:
        print(str(k) + ',' + (icdcodes[k] if k<lines else tokens_dict[k-
lines] if tokens_dict else '-')  + ',' + str(v))

def make_pr_curve_pict(y_test, y_score, any_folder, year):
    # PR取得
    precision, recall, thresholds = precision_recall_curve(y_true=y_test,
probas_pred=y_score)
    # 結果をプロット
    plt.figure(1, figsize=[12.8, 9.6], dpi=100)
    plt.figure(1)
    plt.plot(recall, precision, label=(any_folder if any_folder != '.' else
'precision_recall curve') + ' (AUC = %0.3f)' % auc(recall, precision))
    for i in range(21):
        close_point = np.argmin(np.abs(thresholds - (i * 0.05)))
        plt.plot(recall[close_point], precision[close_point], 'o')
     # ラベルなどを追加しファイル出力
    plt.plot([0,1], [1,1], linestyle='--', label='ideal line')
    plt.legend()
    plt.xlabel('recall')
    plt.ylabel('precision')
    plt.set_title=('P-R Curve')
    fname = './' + any_folder + '/' + year + '/PR-Curve.png'
    plt.savefig(fname)
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] 結果を出力
しました。[' + fname + ']')

def make_roc_curve_pict(y_test, y_score, any_folder, year):
    # ROC取得
    fpr, tpr, thresholds = roc_curve(y_true=y_test, y_score=y_score)
    # 結果をプロット
    plt.figure(2, figsize=[12.8, 9.6], dpi=100)
    plt.figure(2)
    plt.plot(fpr, tpr, label=(any_folder if any_folder != '.' else 'roc
curve') + ' (AUC = %0.3f)' % auc(fpr, tpr))
    for i in range(21):
        close_point = np.argmin(np.abs(thresholds - (i * 0.05)))
        plt.plot(fpr[close_point], tpr[close_point], 'o')
    # ラベルなどを追加しファイル出力
    plt.plot([0,0,1], [0,1,1], linestyle='--', label='ideal line')
    plt.plot([0, 1], [0, 1], linestyle='--', label='random prediction')
    plt.legend()
    plt.ylabel('false positive rate(FPR)')
```

```
    plt.xlabel('false positive rate(FPR)')
    plt.ylabel('true positive rate(TPR)')
    plt.set_title=('ROC Curve')
    fname = './' + any_folder + '/' + year + '/ROC-Curve.png'
    plt.savefig(fname)
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] 結果を出力
しました。[' + fname + ']')

def write_report(report, any_folder, year):
    fname = './' + any_folder + '/' + year +
'/fit_and_predict_xgboost_report.csv'
    fos = open(fname, 'w')
    for l in list(zip(*report)):
        fos.write(','.join([str(f) for f in l]) + '\n')
    fos.close()
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] 結果を出力
しました。[' + fname + ']')

def write_id_tf(ids, act, pred, ikichi, any_folder, year):
    fname = './' + any_folder + '/' + year +
'/fit_and_predict_xgboost_id_tf_ikichi_' + str(ikichi) + '.csv'
    fos = open(fname, 'w')
    for i, a, p in list(zip(ids, act, pred)):
        fos.write( '{:.0f},{:.0f},{:.0f},{}'.format(i, a, p, a==p) + '\n')
    fos.close()
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] 結果を出力
しました。[' + fname + ']')

def write_pred_score(ids, act, pred_score, any_folder, year):
    fname = './' + any_folder + '/' + year +
'/fit_and_predict_xgboost_pred_score.csv'
    fos = open(fname, 'w')
    for i, a, p in list(zip(ids, act, pred_score)):
        fos.write( '{:.0f},{:.0f},{:.5f}'.format(i, a, p) + '\n')
    fos.close()
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] 結果を出力
しました。[' + fname + ']')

if __name__ == '__main__':
    any_folder, gpuid, year = arg_parse()
    main(any_folder, gpuid, year)


```

令和４年４月１９日

厚生労働大臣
殿

機関名　国立大学法人東京大学

所属研究機関長　職　名　学長

氏　名　藤井　輝夫

次の職員の令和３年度厚生労働科学研究費の調査研究における、倫理審査状況及び利益相反等の管理については以下のとおりです。

１．研究事業名　政策科学総合研究事業（統計情報総合研究事業）

２．研究課題名　死因統計の精度及び効率性の向上に資する機械学習の検討に関する研究

３．研究者名　（所属部署・職名）　大学院医学系研究科・准教授

（氏名・フリガナ）　今井　健・イマイタケシ

４．倫理審査の状況

	該当性の有無 有　無	左記で該当がある場合のみ記入（※1）		
		審査済み	審査した機関	未審査（※2）
人を対象とする生命科学・医学系研究に関する倫理指針（※3）	□　■	□		□
遺伝子治療等臨床研究に関する指針	□　■	□		□
厚生労働省の所管する実施機関における動物実験等の実施に関する基本指針	□　■	□		□
その他、該当する倫理指針があれば記入すること （指針の名称：　）	□　■	□		□

（※1）当該研究者が当該研究を実施するに当たり遵守すべき倫理指針に関する倫理委員会の審査が済んでいる場合は、「審査済み」にチェックし一部若しくは全部の審査が完了していない場合は、「未審査」にチェックすること。

その他（特記事項）

（※2）未審査に場合は、その理由を記載すること。
（※3）廃止前の「疫学研究に関する倫理指針」、「臨床研究に関する倫理指針」、「ヒトゲノム・遺伝子解析研究に関する倫理指針」、「人を対象とする医学系研究に関する倫理指針」に準拠する場合は、当該項目に記入すること。

５．厚生労働分野の研究活動における不正行為への対応について

研究倫理教育の受講状況	受講 ■　未受講 □

６．利益相反の管理

当研究機関におけるＣＯＩの管理に関する規定の策定	有 ■　無 □（無の場合はその理由：　）
当研究機関におけるＣＯＩ委員会設置の有無	有 ■　無 □（無の場合は委託先機関：　）
当研究に係るＣＯＩについての報告・審査の有無	有 ■　無 □（無の場合はその理由：　）
当研究に係るＣＯＩについての指導・管理の有無	有 □　無 ■（有の場合はその内容：　）

（留意事項）　・該当する□にチェックを入れること。
・分担研究者の所属する機関の長も作成すること。

令和４年３月３１日

厚生労働大臣　　殿

機関名　国立大学法人筑波大学

所属研究機関長　職　名　国立大学法人筑波大学長

氏　名　永田　恭介

次の職員の令和３年度厚生労働科学研究費の調査研究における、倫理審査状況及び利益相反等の管理については以下のとおりです。

１．研究事業名　政策科学総合研究事業（統計情報総合研究事業）

２．研究課題名　死因統計の精度及び効率性の向上に資する機械学習の検討に関する研究

３．研究者名　（所属部署・職名）　医学医療系　講師

（氏名・フリガナ）　香川　璃奈　・　カガワ　リナ

４．倫理審査の状況

	該当性の有無	左記で該当がある場合のみ記入（※1）		
	有　無	審査済み	審査した機関	未審査（※2）
人を対象とする生命科学・医学系研究に関する倫理指針（※3）	□　■	□		□
遺伝子治療等臨床研究に関する指針	□　■	□		□
厚生労働省の所管する実施機関における動物実験等の実施に関する基本指針	□　■	□		□
その他、該当する倫理指針があれば記入すること（指針の名称：　　）	□　■	□		□

（※1）当該研究者が当該研究を実施するに当たり遵守すべき倫理指針に関する倫理委員会の審査が済んでいる場合は、「審査済み」にチェックし一部若しくは全部の審査が完了していない場合は、「未審査」にチェックすること。

その他（特記事項）

（※2）未審査に場合は、その理由を記載すること。

（※3）廃止前の「疫学研究に関する倫理指針」、「臨床研究に関する倫理指針」、「ヒトゲノム・遺伝子解析研究に関する倫理指針」、「人を対象とする医学系研究に関する倫理指針」に準拠する場合は、当該項目に記入すること。

５．厚生労働分野の研究活動における不正行為への対応について

研究倫理教育の受講状況	受講 ■　未受講 □

６．利益相反の管理

当研究機関におけるＣＯＩの管理に関する規定の策定	有 ■　無 □（無の場合はその理由：　　）
当研究機関におけるＣＯＩ委員会設置の有無	有 ■　無 □（無の場合は委託先機関：　　）
当研究に係るＣＯＩについての報告・審査の有無	有 ■　無 □（無の場合はその理由：　　）
当研究に係るＣＯＩについての指導・管理の有無	有 □　無 ■（有の場合はその内容：　　）

（留意事項）　・該当する□にチェックを入れること。
・分担研究者の所属する機関の長も作成すること。