機械学習プログラムからのドメイン知識の自動抽出

立石 孝彰[1,a)]　高橋 俊博[1,b)]　中村 祐一[1,c)]

概要：機械学習では，元の特徴量から新しい特徴量を追加する Feature Engineering と呼ばれるデータの前処理工程が重要である．一般的に，Feature Engineering は，与えられたデータセットに対するドメイン知識を利用して試行を繰り返すことが多く，時間のかかる工程である．また，ドメイン知識を活用するという点から自動化が難しい．本論文では，プログラムの実行履歴からドメイン知識を収集し，Feature Engineering のために自動的に再利用することを提案する．また，提案した手法を実装し，その実現性・有用性を確認するための実験を行った．実験では，ローンの予測問題に関わる github.com 上の 30 リポジトリを対象とし，機械学習に有用な数式の抽出とその再利用を行った．

1. はじめに

機械学習では，元の特徴量から新しく特徴量を作成する Feature Engineering と呼ばれるデータの前処理工程があり，機械学習を用いたプロジェクトの成否に関わるほど重要である．一般的に，機械学習のアルゴリズムは汎用的であるが，Feature Engineering はドメインや与えられたデータに特化した作業やドメイン知識が必要 [1] とされており，自動化が難しい工程である．例えば，過去の借入履歴データから返済不履行のリスクを予測する場合，毎月の収入に対する返済額の比 (debt-to-income ratio) が重要な特徴量となる．なぜなら，返済と収入の意味を考えると，これらの比が返済の容易さを表す数値となることが説明となり得るためである．このようなドメイン知識を用いて作成される特徴量を，本論文ではドメイン特徴量と呼ぶ．しかし，与えられたデータセットから，それぞれの特徴量の意味を自動的に把握し，さらに，それらの比に意味があることを自動的に判断するような技術は，我々が知る限りでは存在しない．そこで我々は，ドメイン知識を収集し再利用するによって，ドメイン特徴量を自動的に作成することを目指している．

このための最初の試みとして，本論文では，テーブルデータを対象とした機械学習において，カラム同士の計算を表す数式をドメイン特徴量として抽出し再利用する方法を提案する．例えば，先の例と同じく借入履歴のテーブルデータがあり，借入金額 (Debt) カラムと，年収 (Income) カラムがあるとする．このテーブルデータに対してドメイン知識を持つエンジニアが機械学習に取り組む場合，debt-to-income 比をテーブルに加えるために「Debt/Income」の値を計算した新しいカラム (DTI) を追加するようなプログラムを作成する．我々は，そのようなプログラム中の数式をドメイン知識として抽出する．ここで，プログラムから抽出する数式は，カラム同士の計算を表す数式となっており，そのままでは他のデータセットに対する再利用が困難である．そのため，類似するカラムの場合に再利用ができるように，数式中のカラム名をオントロジーにおけるコンセプトに置き換える．ここで，コンセプトとは，そのカラムの意味を表す名前である．また，本論文においては，カラムとコンセプト間の関係は事前に与えられているものとする．そして，このように抽出された (コンセプト上の) 数式を，同様の異なるデータセットに対して再利用する．例えば，Debt と Income をコンセプトとし，別のデータセットにおける年収と借入金額のカラム名が ANNUAL_INC(Annual Income) と LOAN_AMT(Loan Amount) になっている場合，これらのカラムに対するコンセプトとして Income と Debt を割り当てることによって「Debt/Income」の再利用を行うことができる．

論文の貢献

本論文の貢献は次の通りである．

- Pandas ライブラリ [*1] を用いた機械学習を行う Python プログラムからコンセプト上の数式を抽出するために，動的解析において追跡すべき実行時情報を明らかにする．

[1] IBM Research - Tokyo, Japan
[a)] tate@jp.ibm.com
[b)] e30137@jp.ibm.com
[c)] nakamury@jp.ibm.com

[*1] https://pandas.pydata.org/

- 動的解析および数式抽出を実装し，それらをオープンソースプログラムに適用することによって，自動化の範囲と抽出される数式の確認を行う．
- 抽出した数式に基づく新しい特徴量 (ドメイン特徴量) が，機械学習に貢献できる場合があることを実験によって示す．

論文の構成

本論文の構成は次の通りである．2 節において関連研究を述べる．プログラムからの数式の抽出と再利用方法を 3 節で述べ，その後の 4 節において，我々が行った実験とその結果を述べる．最後の 5 節で本論文のまとめを行う．

2. 関連研究

近年では，自動機械学習の取り組みがあり，そのうちのいくつかのものでは Feature Engineering の自動化 [2], [3], [4], [5] を提案している．これらは元の特徴量を組合せることによって新しい特徴量を求めており，すべての組合せの中から学習・予測に有用な組合せを統計的手法あるいは機械学習を用いて効率よく探索する仕組みを利用している．しかしながら，カラムの意味を把握して組合せの効率化を行っていないので，複数カラムを伴う複雑な数式を発見するには効率が悪い．また，人がドメイン知識を用いて行う Feature Engineering とは異なり，理解容易かつ合理的な組合せによる新しい特徴量が作られないこともあるため，新しい特徴量の意味を理解することが難しい場合がある．場合によっては，新しく作られた特徴量のために，過剰に訓練データに適合してしまうこともある．一方で，我々が提案する手法は，カラムの意味に基づいて既知の数式を利用するため，この点で既存手法よりも効率は良い．また，数式の理解や説明も容易となり，人による調整によって不合理な特徴量を特定することにも役立つと考えている．一方で，再利用という性質上，全く新しい特徴量を発見するということはできない．

プログラムからドメイン知識を抽出する取り組みは少ない．文献 [6] では，Web アプリケーションのソースコード (あるいはソースコード中から判定できる入力フィールド名など) と，関係データベースのカラムの対応付けを利用することによって，コンセプト間の関係を発見し，オントロジーの開発に役立てることを述べている．文献 [7] では，ソースコード中の識別子と，対応するドキュメント中の語句の対応付けを学習することによって，is-a 関係や has-a 関係を含めてドメイン固有の用語集の作成を行っている．また，文献 [8] では，保険分野のドメイン知識としてビジネスルールを抽出する対話的な手法を提案している．

カラムに対応するコンセプトの特定には, Ontology Learning[9] の技術を利用できると考えている．Ontology Learning では，主に文章からコンセプトとその間の関係を抽出する．多くのデータセットでは，カラムに対して短い説明が付いており，その説明文章を利用してカラムに対するコンセプトを特定することができると考えている．また，コンセプトへの対応付けは，数式再利用時のカラム同士の類似性を判定するために必要であった．このカラム同士の類似性という観点からは，スキーママッチング [10], [11] の手法があり，再利用のために応用できる可能性があると考えている．

3. ドメイン知識の抽出と再利用

プログラムからドメイン知識としての数式を抽出し再利用するまでの手続きは次の通りである．

依存グラフの作成 プログラムを解析し，カラム間の依存関係を表す依存グラフを作成する．

数式データベースの作成 依存グラフに基づいてコンセプト上の数式を構成し数式データベースに格納する．数式データベースとは，コンセプト上の数式を集めたデータベースであり，コンセプトの集合をクエリーとして，関連する数式の集合をクエリー結果として返す．

数式の再利用 与えられたカラムに対応するコンセプトの集合を入力として，数式データベースから関連する数式を検索し，その数式に対応するカラムを追加するような前処理プログラムを生成する．

以降では，それぞれの手続きの詳細を述べる．

3.1 依存グラフの作成

プログラム中におけるメソッド呼び出しとカラムの依存関係を解析するために，我々は動的解析を用いる．一般的に，プログラム中の関係 (例えば，変数間の関係や関数呼び出し関係) を解析する場合，大きく分けて動的解析と静的解析という二つのアプローチがある．動的解析では，プログラムを実際に実行することによって，実行時に起こった関係を抽出する．一方で，静的解析では，プログラムを実行せずに解析を行い，実行時に起こり得る可能性がある関係を近似または抽象化された関係として抽出する．特に，配列とループを用いた処理や，再利用可能なモジュールに分割されたプログラムに対して，実行時に起こる関係を精度良く特定することは困難である．機械学習における Feature Engineering で役立てるという観点では，実際に生成される特徴量が重要となる．このため，我々は動的解析のアプローチを用いる．

我々の動的解析は，テーブルデータの操作およびカラム同士の演算の実行時情報のリストから成る実行履歴 Trace を入力として，カラム名とメソッドの依存関係を表す依存グラフと呼ぶラベル付き有効グラフを生成する．以降では，動的解析の対象とする実行履歴，解析結果となる依存グラフ，実行履歴からの依存グラフの生成方法について説明する．

```
1  import pandas as pd
2  import numpy as np
3  df1 = pd.read_csv(...)
4  df1['ratio'] = df1['loan']/df1['income']
5  df2 = pd.get_dummies(df1['gender'])
6  df3 = pd.concat([df1, df2], axis=1)
```

図 1 Python プログラムの例

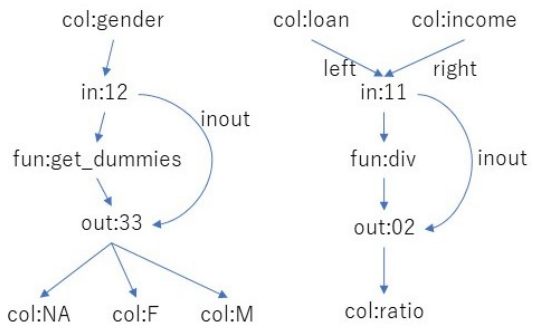

図 2 依存グラフの例

```
1  get('loan', obj01)
2  get('income', obj02)
3  call('div', [], [],
4       [('left',obj01),('right',obj02)],
5       [obj03])
6  set('ratio', obj04, obj03)
7  get('gender', obj11)
8  call('get_dummies',
9       [('columns',['gender'])],
10      ['F','M','NA'],
11      [], [])
```

図 3 実行履歴の例

実行履歴

入力となる実行履歴 Trace は，カラム間の関係を解析するために必要な次の 4 種類の実行時に得られる情報の列である．

- ini($names, tbl$): カラム名のリスト $names$ から成るテーブルオブジェクト tbl を作る．
- get($name, obj$): カラム名 $name$ によってあるテーブルからカラムオブジェクト obj を取り出す．[*2]
- set($name, tbl, obj$): テーブルオブジェクト tbl にカラム名 $name$ としてカラムオブジェクト obj を挿入する．
- call(f, ics, ocs, ios, oos): パラメータ名とカラム名の組のリスト ics と，パラメータ名とオブジェクトの組のリスト ios を入力として関数 f を呼び出し，その結果として新しく生成される複数のカラムに対して，その名前のリストが ocs，または，対応するカラムオブジェクトのリストが oos である．

[*2] 現在の我々の手法では，取り出したオブジェクトに対するカラム名のみを追跡するため，テーブルの情報を利用していない．

- ini($names, obj$) の場合，以下のエッジを生成する．
  - すべての $name \in names$ に対して，
    $(\texttt{col}, name) \rightarrow (\texttt{obj}, obj)$
- get($name, obj$) の場合，以下のエッジを生成する．
  - $(\texttt{col}, name) \rightarrow (\texttt{obj}, obj)$)
- set($name, tbl, obj$) の場合，以下のエッジを生成する．
  - $(\texttt{obj}, obj) \rightarrow (\texttt{col}, name)$
  - $(\texttt{obj}, tbl) \rightarrow (\texttt{col}, name)$
- call(f, ics, ocs, ios, oos) の場合，以下のエッジを生成する．
  - すべての $(name, icol) \in ics$ に対して，
    $(\texttt{col}, icol) \xrightarrow{name} (\texttt{in}, \langle ics, ocs \rangle)$
  - すべての $(name, iobj) \in ios$ に対して，
    $(\texttt{obj}, iobj) \xrightarrow{name} (\texttt{in}, \langle ics, ocs \rangle)$
  - $(\texttt{in}, \langle ics, ios \rangle) \rightarrow (\texttt{fun}, f)$
  - $(\texttt{fun}, f) \rightarrow (\texttt{out}, \langle ocs, oos \rangle)$
  - すべての $ocol \in ocs$ に対して，
    $(\texttt{out}, \langle ocs, oos \rangle) \rightarrow (\texttt{col}, ocol)$
  - すべての $oobj \in oos$ に対して，
    $(\texttt{out}, \langle ocs, oos \rangle) \rightarrow (\texttt{obj}, oobj)$
  - $(\texttt{in}, \langle ics, ios \rangle) \xrightarrow{inout} (\texttt{out}, \langle ocs, oos \rangle)$

図 4 依存グラフ生成のルール

ここで，テーブルオブジェクトやカラムオブジェクトとは，テーブルやカラムを表すオブジェクトであり，例えば Pandas ライブラリにおける DataFrame クラスや Series クラスのオブジェクトである．カラム名とオブジェクトの扱いを明示的に分けることによって，同一オブジェクトであっても異なるカラム名が設定されること，または，同一のカラム名であっても異なるオブジェクトが生成されることを表すことができる．図 3 は，図 1 の Python プログラムに対する実行履歴の例である．このような実行履歴の取得は，それぞれのメソッドの仕様と実行時の動作を確認しながら行うものとする．

依存グラフ

依存グラフ (N, E, L) は，プログラム中の値に対応するノード N と，それらの依存関係を表すエッジ E，エッジへのラベル L から構成される．ノード $(k, v) \in N$ は，ノード種別を表すラベル k と値 (オブジェクトや関数など) の集合 v の組として表現され，ノード種別には col(カラム名)，fun(メソッド名)，in(メソッドへの入力に対する識別子)，out(メソッドの出力に対する識別子)，const(定数) を用いる．ここで，in ノードと out ノードは，メソッドの入力あるいは出力となる (複数の) カラムやオブジェクトを一つの集まりとして扱うために導入したノードである．E は依存関係 $(k_1, v_1) \xrightarrow{l} (k_2, v_2)$ の集合を表し，$(k_1, v_1), (k_2, v_2) \in N$ が依存元と依存先のノード，$l \in L$ が依存関係の種類表す．依存関係の種類には，メソッドの入出力関係を表す inout，メソッドが用いるパラメータ名 (あるいは引数の番号)，または ϵ を用いる．特に ϵ の場合には $(k_1, v_1) \rightarrow (k_2, v_2)$ と略記する．例えば，図 1 の Python プログラムに対して，図 2 の依存グラフを作成する．

実行履歴から依存グラフの作成

実行履歴 Trace から依存グラフ (N, E, L) を構成するた

めには，まず最初に各実行時情報に対して図 4 の通りに依存関係 $(k_1, v_1) \xrightarrow{l} (k_2, v_2)$ を生成する．直感的には，メソッドの入力から出力への関係と，それら入出力に対するカラム名の対応付けを行い，グラフ探索によって数式を構成可能なように依存関係を生成している．ここで，ノード種別として「その他のオブジェクト」を表す `obj` を新たに導入している．また，$\langle x, y \rangle$ は x と y の組に対する識別子を表す．

次に，得られた依存関係の集合で構成される有向グラフから `obj` ノードを取り除き，カラム名，関数名，定数だけからなる有向グラフを得ることによって，依存グラフを獲得することができる．ここで，`obj` ノードの除去は，有限オートマトンから正規表現を構成するアルゴリズムと同様に，連続するエッジのラベルを結合しながら状態の削除を繰り返すことによって達成する．

3.2 数式データベースの作成

数式データベース *ExpDB* は，コンセプトの集合をクエリーとして与えたときに, 数式の集合をクエリー結果として返すデータベースである．より形式的には, コンセプトの集合 *Cs* と数式の集合 *Exp* に対して, $\mathit{ExpDB} \subseteq \mathrm{pow}(\mathit{Cs}) \times \mathit{Exp}$ を満たし，かつ，各要素 $(s, e) \in \mathit{ExpDB}$ が $s = \mathrm{cs}(e)$ を満たす集合として定義する．ここで，$\mathrm{cs}(e)$ は数式 e に現れるコンセプトの集合を表し，数式 e は次の文法 G_e から構成されるものとする．

$$\begin{array}{rcl} G_e & \rightarrow & f(G_e, \cdots) \\ & | & c \end{array}$$

f は関数名 (`fillna` や $+$ など) であり，任意の個数の引数を持つ．2 引数の関数については，例えば，$+(e_1, e_2)$ は $e_1 + e_2$ と略記する．c はコンセプトである．そして，コンセプトの集合 q がクエリーとして与えられた場合には，$s \subseteq q \wedge (s, e) \in \mathit{ExpDB}$ を満たす e の集合をクエリー結果として返す．このため，数式の集合が与えられれば，数式データベースを作成することができる．

依存グラフ (N, E, L) から数式を生成するためには，`out` ノードから `inout` エッジを辿りながら関数 (演算子を含む) を適用する項を作成する．より具体的には，この再帰的な手続きは次の通りのノード n を引数にとり数式の集合を返す関数 M によって定義する．

$$M(n) = \begin{cases} [\![v]\!] & n = (\mathtt{col}, v) \text{ の場合} \\ \{[\![f]\!](a_1, a_2, \cdots) \mid (a_1, a_2, \cdots) \in S, & n = (\mathtt{out}, v) \text{ の場合} \\ \quad S = M(m_1) \times M(m_2) \times \cdots, & (1) \\ \quad m_i \xrightarrow{i} (\mathtt{in}, v') \in E, & (2) \\ \quad (\mathtt{fun}, f) \rightarrow (\mathtt{out}, v) \in E, & (3) \\ \quad (\mathtt{in}, v') \xrightarrow{\mathtt{inout}} (\mathtt{out}, v) \in E\} & (4) \end{cases}$$

ここで，$[\![v]\!]$ によって，カラム v に対応するコンセプトの集合を表す．これは，カラムには複数のコンセプトが対応付く場合があるためである．また，$[\![f]\!]$ によって，メソッド f に対応する数式における関数名を表す．例えば，依存グラフ上での “`div`” メソッドが，数式の関数名 “/ ” となる場合，$[\![\mathtt{div}]\!] = /$ である．条件 (1) において，$M(m_1) \times M(m_2) \times \cdots$ は数式の組合せを表し，その要素が $(a_1, a_2, \cdots)$ である．条件 (2) は，m_i が入力ノードへの依存元のノードであることを表す[*3]．条件 (3) は，出力ノードに対応するメソッドが f であることを表す．最後に条件 (4) は，出力ノードと入力ノードの対応を表している．

多くの場合，`in` ノードと `out` ノードの組合せは，実行時のメソッド呼び出し毎に異なる組合せとなる．このため，`out` ノードから `in` ノードへのエッジがあったとしても `inout` エッジの連鎖が循環することはない．しかし，メソッド呼び出しによって内容が変化しないなどの場合には依存グラフ中に循環が生じる．この場合，上記の条件だけではアルゴリズムは停止しない場合がある．そのため，n の履歴を保持しておき，2 度目に同じ n を取る場合には空集合を返すものとする．

この手続きをすべての出力ノードに対して適用することによって，プログラム中で利用されている数式と，その数式に関連するコンセプトを収集し，数式データベースを作成する．

3.3 数式の再利用

再利用する数式を数式データベースから集めるためには，次の 2 つのステップを行う．

- あるデータセットが与えられたときに，そのデータセットのカラムを表すコンセプトの集合 q をクエリーとして数式データベースへ問い合わせる．クエリー結果は，3.2 節で述べた通り，数式の集合となる．
- Feature Selection[12] によって得られた数式の集合の中から，効果が期待できる数式を選択する．Feature Selection とは，与えられた特徴量の集合から，その部分集合を求める手続きのことであり，機械学習分野では，この自動化について数多くの技術が提案されている．カラムに対応するコンセプトを求める手続きが保守的な場合 (間違ったコンセプトを返すことを許容する場合)，あるいは一つのカラムが複数の異なるコンセプトを持つ場合，本来は異なる意味を持つカラム同士であるのにもかかわらず，同じコンセプトを持つことがある．この結果，検索によって得られる数式も多くなる．この中には，実際には有用ではない数式も含まれることがある．そこで，数式の選択のステップ

[*3] 依存元のノードは，$(\mathtt{col}, v)$ または $(\mathtt{out}, v)$ となり，プログラム中において関数呼び出し結果を使う関数がある場合に，依存元ノードが $(\mathtt{out}, v)$ となる．また，Python を想定して依存グラフでは名前付きの引数を考慮しているが，ここでは簡単のため，引数は自然数によって番号付けれらえているものとする．

において再利用により効果が見込める数式だけを選択する．

尚，数式から前処理プログラムを生成する方法は，簡単なプログラム変換と見なすことができ，その詳細は 3.4 節で述べる．

3.4 実装

前節までに述べた一連の手続きのうち，カラムからのコンセプトの推定と Feature Selection 以外の手続きを Python を用いて実装した．そして，カラムとコンセプトの対応関係を表 1 の通り事前に与えておくことによって，次の二つの手続きを自動化した．

(1) プログラムから動的解析によって数式を収集し数式データベースを作成

(2) データセットをクエリーとして数式データベースから関連する数式を検索し，前処理プログラムを生成

以降では，(1) における動的解析と (2) における前処理プログラムの生成について述べる．

動的解析の実装

動的解析の対象は，Python プログラム，または，Jupyter Notebook [*4]のノートブック中の Python プログラムである．対象となる Python プログラムに対して動的解析を行い依存グラフを生成するまでのプロセスは次の通りである．

1. `pipenv`[*5]環境の構築

2. Python プログラムに対する instrumentation

カラム名	コンセプト
Loan_ID	id
Gender	gender
Married	married
Dependents	number
Education	degree, education
Self_employed	self_employed
ApplicantIncome	income
CoapplicantIncome	income
LoanAmount	loan
Loan_Amount_Term	term
Credit_History	history
Property_Area	area
Loan_Status	loan_status
Creditability	creditability
Account Balance	balance, income
Duration of Credit (month)	term
Payment Status of Previous Credit	history
Purpose	purpose
Credit Amount	credit, loan
Value Savings/Stocks	saving, stock
Length of current employment	length, years
Instalment per cent	instalment
Sex_and_Marital_Status	gender, married
Guarantors	guarantor
Duration in Current address	duration
Most valuable available asset	asset
Age (years)	age, years
Concurrent Credits	loan, credit
Type of apartment	apartment
No of Credits at this Bank	credit, loan
Occupation	occupation
No of dependents	dependents
Telephone	telephone
Foreign Worker	foreign_worker

表 1 カラムとコンセプトの対応

[*4] https://jupyter.org/
[*5] https://github.com/pypa/pipenv

```
1  np._orig_log = np.log
2  def _instr_numpy_log(x):
3    return np._orig_log(x)
4  np.log = _instr_numpy_log
5  ...
6  pd._orig_read_csv = pd.read_csv
7  def _instr_read_csv(csv,*args,**kw):
8    csv=find_file(basename(csv))
9    df=pd._orig_read_csv(csv,*args,**kw)
10   return df.sample(n=100)
11 pd.read_csv = _instr_read_csv
```

図 5 前処理後の Python プログラム

3. `pipenv` 環境下でプログラムを実行し実行履歴を取得

4. 得られた実行履歴から依存グラフを作成

まず初めに，`pipenv` 環境を作るために，その設定ファイルである `Pipfile` を自動的に作成する．`pipenv` とは，仮想環境上で python プログラムを実行するツールである．また，`Pipfile` 中には，実行に必要なパッケージ一覧を記述するが，この一覧を自動的に得るために `pipreqs`[*6] を用いた．

instrumentation ステップにおいては，次の二つの観点で元のソースコードに対して自動で改変を行った．

- C 言語で実装されたメソッド (native メソッド) について，あたかも Python 言語で実装された場合と同様に値の追跡が可能となるように再定義を行う．これは，native メソッド呼び出しの引数や返り値を取得することが困難であったためである．例えば，Numpy ライブラリの `log()` は native メソッドであるため，図 5 の 1–4 行目の通りに `log()` の再定義を行った．
- 多くの機械学習のプログラムでは，データセット (CSV ファイル) へのファイルパスが各著者の環境に合わせてハードコードされていることが多い．このため，ベースファイルを手掛かりに，事前に与えられた検索パスから対象とする CSV ファイルを探索するように Pandas ライブラリの `read_csv()` 関数の再定義を行う (図 5 の 6-11 行目)．
- 上記に加えて，`read_csv()` メソッドに対しては，プログラムの実行時間や必要なメモリの削減のために，事前に与えらえた数でランダムサンプリングしたデータセットを返すようにする (図 5 の 10 行目)．

3.1 節で述べた実行履歴を取得するために，hunter[*7] と呼ぶ実行を監視するツールを拡張して利用した．ここで，Pandas ライブラリおよび Numpy ライブラリによるカラム同士の数式による演算と，いくつかのテーブルデータの操作のみを監視対象とした．拡張によって追加した機能は以下の通りである．

- 監視対象のテーブルやカラム中のデータを操作するメ

[*6] https://pypi.org/project/pipreqs/
[*7] https://pypi.org/project/hunter/

ソッド呼び出しを事前に登録しておき，そのメソッド呼び出し時の引数及び返り値を JSON 形式で取得可能とする．また，引数や返り値がテーブルオブジェクトやカラムオブジェクトの場合には，カラム名の情報も JSON 形式のデータの中に含める．

最後に，実行履歴から依存グラフの作成は，3.1 節で述べた方法を実装した．

前処理プログラムの生成

前処理プログラムの生成では，カラムの集合を入力として，テーブルデータを操作する Python プログラムを生成する．この Python プログラムは，入力のカラム集合に対応するデータセットから新しい特徴量 (ドメイン特徴量と呼ぶ) を生成するものである．入力カラムの集合は，事前に与えられたカラム-コンセプトの対応関係に基づいて，コンセプトの集合に変換される．このコンセプトの集合をクエリーとして，数式データベースから関連する数式を取得し，数式中のコンセプトをカラム名に置換して，前処理プログラムを生成する．複数のコンセプトが数式中にあり，それぞれのコンセプトが複数のカラムと対応する場合には，すべてのカラムの組合せによる置換結果を生成する．例えば，debt/income という数式があり，コンセプト debt に対してカラム Loan と Credit が対応し，コンセプト income に対してカラム AppIncome と CoappIncome が対応する場合，次のプログラムを生成する[*8]．

```
df['F1'] = df['Loan']/df['AppIncome']
df['F2'] = df['Loan']/df['CoappIncome']
df['F3'] = df['Credit']/df['AppIncome']
df['F4'] = df['Credit']/df['CoappIncome']
```

df は Pandas ライブラリの DataFrame オブジェクトである．F1 から F4 は，新しく生成されるカラムの名前であり，既存のカラム名と重複しない名前を用いる．また，数式データベースとして，RDF TTL フォーマット [13] 形式のファイルを用いた．このため，SPARQL クエリー言語 [14] を用いてコンセプトの集合に基づき数式の探索を行った．

4. 実験と調査

本論文では以下の 3 つの観点から実験・調査を行った．

- 数式自動抽出のための動的解析の実用性
- 抽出した数式の有用性と再利用の可能性

なお，今回の実験においては，2.3GHz(4 コア) CPU と 16GB メモリの仮想マシンを用いた．また，全プログラムの実行が数時間内で完了でき，サンプル数が少ないために起こるエラーをなるべく減らすことを考えて，read_csv() メソッドでは，ランダムサンプリングによって 100 個のサンプルを含む DataFrame オブジェクトを返すように再定義した．

4.1 動的解析の実用性

静的解析とは異なり，動的解析では実際の実行が必要となる．そのため，3.4 節の実装によって，どの程度自動実行が可能であるのかを調べるために実験を行った．実験では，github.com において，次のキーワードの組み合わせにより検索を行い，得られた 497 リポジトリのうちデータセットとして CSV ファイルをリポジトリ内に持つ 30 リポジトリ (表 2) を数式自動抽出の対象とした．

- ”loan”, ”default”, ”prediction”
- ”loan”, ”lightgbm”
- ”loan”, ”prediction”, ”income”
- ”loan”, ”xgboost”

これらのリポジトリ中には，Python プログラム (ノートブック含む) は合計で 45 個あり，各レポジトリ毎に最小で 1 個，最大で 4 個の Python プログラムが存在した．また，29 リポジトリは同一のカラム名を持つ CSV ファイルを対象としており，残りの 1 リポジトリ[*9] のみに異なるカラム名も持つ CSV ファイルがあった．

これらのリポジトリから我々の実装を用いて自動的に数式を抽出して数式データベース (TTL ファイル) を作成することを試みた．この結果，45 個の Python プログラム中の 34 個において依存グラフの作成を行い，そこから数式を

表 2 実験対象のリポジトリ一覧
それぞれ先頭の https://github.com/users/ は省略している．

Aishwarya4823/Loan-Prediction-Problem
Architectshwet/Loan-prediction-using-Machine-Learning-and-Python
DVSimon/Loan_Granting_Predictions
RatnamDubey/Loan_Prediction_Analytics_vidhya
Shriya29/LoanPrediction
NaeemHasan23/loan-prediction
adityajn105/Loan-Prediction
akshaykapoor347/Loan-prediction
UTKARSHPRAJAPTI/Practice-Problem-ML_Loan-Prediction-III
alonsopg/loan_prediction
aman1002/Loan-Status-Prediction-Analytics-Vidya-
balancy/Loan_prediction
anshulporwal26/Loan-approval-prediction
behroozeslami/Predicting_Mortgage_Rates
blmali/Loan-Prediction
datascience110/Loan-Prediction
dwivedi1997/Loan-Prediction
gagicha/Loan_prediction_xgboost
harsh-kr/Loan-Prediction-Problem
ganevniko/LendingClub-Credit-Risk-Analysis
maulik2222/Loan-Prediction-Problem-
nanyamshukla/Loan_Approval_Prediction
narwaljyoti/loan_prediction-
rakshithvasudev/Home-Credit-Default-Risk
neuralsnig1999/AV--Hackathon---Loan-Prediction
preeyonujb1/EDA-and-Basic-ML-Practice-on-Loan-Defaulter-Dataset
ritusri/Loan-Prediction-Problem
shivanireddy/Loan-Prediction
shriya2909/Loan-Prediction
suyashgupta02/LoanPrediction-KernelSVM-RandomForest-NeuralNetwork-XGBoost

[*8] このように生成されるプログラムでは，与えられたデータセットによっては，実行時にエラーが起こる場合があるため，例外処理を行うためのコードも生成した．
[*9] 表 2 中の alonsopg/loan_prediction

抽出することができた．表 3 は，中間成果物として生成された依存グラフの情報をまとめたものである．inout ノード数は，我々の動的解析が監視対象とするメソッド呼び出しの回数と等しい．また，表 4 は，抽出した数式のパターンと，その数式の導出元となったプログラム数とレポジトリ数を示したものである．ここで，“{}” で囲まれた部分 (“{left}” など) は，仮引数であり，数式データベース中においてコンセプト名と対応付けられている．

プログラムの中にはインデント等の文法エラー，ノートブックから Python プログラムへの変換の不完全さから実行時エラーとなるものがあったが，軽微なものについては手作業で修正を行った．それ以外の実行時エラーのために依存グラフの作成が行えなかった理由は以下の通りである．

- 未定義メソッドの呼び出し: ライブラリのバージョンが異なることが原因のエラーである．また，複数のライブラリの依存関係も考慮して，正常に動作するためのライブラリのバージョンの組合せを探すことが困難であった．
- インデックスやテーブルサイズに関するエラー: one-hot encoding(Pandas では get_dummies() 関数) などメソッドは，データの内容に応じて新規に作成されるカラムが決定される．そのため，我々が実行のために用いた削減されたデータでは想定されているカラムが生成できずにエラーとなってしまったと推測している．

4.2 抽出した数式の有用性

抽出した数式の有用性を調査するために，図 6 の通り，その数式に基づく特徴量 (ドメイン特徴量) が追加された

表 3 依存グラフ中の各要素数

	合計	最小	最大
ノード数	4335	25	385
エッジ数	38605	140	3725
inout 関係	1236	5	171

表 4 抽出した数式の一覧
N_P はプログラム数，N_R はレポジトリ数である．数式中の fillna は，Pandas ライブラリの Series.fillna() を表す．ただし，数式を伴わず Series.fillna() などが単独で現れるものはカウントしていない．

数式のパターン	N_P	N_R
{left}+{right}	14	12
log({x})	12	10
log({left}+{right})	8	6
{left}/({left_1}+{right})	4	2
(({left}+{right})/fillna(...))*100	1	1
log(fillna(...))	1	1
log(log({x}))	1	1
({left}+{right})-(({left}/{right})*1000)	1	1
{left}/{right}	1	1
log({left}-({left_1}*1000))	1	1
log(({left}+{right}) -(({left}/{right})*1000))	1	1

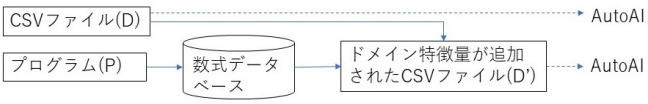

図 6 抽出した数式の有用性調査のための実験

CSV ファイル D' を生成する．そして，CSV ファイル D' と元の CSV ファイル D に対して Watson AutoAI [*10](以降では AutoAI) を適用し，予測スコアの比較と予測への寄与度 (Feature Importance) が高い特徴量の比較を行う．ここで，数式抽出に利用した Python プログラム P は，比較対象である CSV ファイル D を処理するように作成されているものである．

実験では，前節で集めたリポジトリのうち https://github.com/Shriya29/LoanPrediction にあるノートブック PredictiveModel.ipynb を Python プログラム P として用いた．これは，ローン申請に対する承認の可否を予測するプログラムである．また，同リポジトリには学習用データとして過去の承認可否データである train.csv ファイルが存在する．この学習用データをデータセット D として用いて，特徴量が追加されたデータセット D' を生成した．

AutoAI は，学習アルゴリズムの選択，ハイパーパラメータの最適化，Feature Engineering を自動的に行い，ROC AUC や Accuracy,F1 値など与えられた評価指標に最適な予測モデルを作成する．ここで，承認可否のような 2 値分類問題に対しては，AutoAI では 7 つのアルゴリズム[*11] を利用可能である．我々の実験では，ROC AUC を評価指標として，これら 7 つのアルゴリズムから 2 つ (デフォルト値) を自動選択することとした．

表 5 は，実験によって得た予測スコアをまとめたものである．元の CSV ファイル D に対しては，Extra Trees Classifier, Logic Regression の二つのアルゴリズが選択され，交差検証における ROC AUC の最高スコアは Extra Trees Classifier を用いた場合の 0.787 であった．

次に，ドメイン特徴量が追加された CSV ファイル D' に対しては，LGBM Classifier, XGB Classifier の二つのアルゴリズムが選択され, ROC AUC の最高スコアは LGBM Clas-

表 5 ドメイン特徴量による予測スコアの変化
括弧内の数字は，アルゴリズムを手動で選択し適用した場合のスコアである．NA は自動選択によって選ばれなかったことを表す．

	D	D'
Extra Trees	0.787	NA
Logistic Regression	0.770	NA
LGBM	(0.800)	0.806
XGB	(0.778)	0.789

[*10] https://www.ibm.com/jp-ja/cloud/watson-studio/autoai
[*11] https://dataplatform.cloud.ibm.com/docs/content/wsj/analyze-data/autoai-details.html

sifier の 0.806 であった．このときの，Feature Importance が高い特徴量は表 6 の通りであり，我々が追加したドメイン特徴量である RatioLoantoIncome_0 と TotalIncome_0 がそれぞれ上位 1,2 位であった．特に RatioLoantoIncome_0 は，3 つのカラムと 2 つの演算子から構成される式である．カラムと演算子の組合せを総当たりで発見するような手法では，このような数式を導出ことは難しいと考えている．

この実験では，D と D' に対して，それぞれ異なるアルゴリズムが自動選択された．そこで，参考のために LGBM と XGB を手動で選択し，それぞれを D に適用した．その結果，ROC AUC スコアはそれぞれ 0.800, 0.778 であった．

4.3 数式の再利用

数式を再利用することによって，異なるデータセットに対してスコア改善に効果があること，また，スコアが改善する場合にはどのような特徴があるのかを調査するために図 7 の手続きによる実験を行う．ある CSV ファイル D に対して，D とは異なる CSV ファイルに対する n 個の Python プログラム $P_1, \cdots, P_n$ から数式データベースを作成し，この数式データベースを用いて CSV ファイル D にドメイン特徴量を追加した CSV ファイル D' を作成する．そして，D と D' のそれぞれに対して，Watson AutoAI を適用し，得られた予測スコアの比較と，予測への寄与度 (Feature Importance) の比較を行う．また，この実験に加えて，次の 2 つの観点についての実験も行う．

- 3.3 節で述べた通り，本手法はスコア改善の見込みのある特徴量をできるだけ多く生成することが目的であり，場合によっては生成したドメイン特徴量がノイズとなってしまうこともある．そこで，D' において Feature Importance の高いドメイン特徴量のみを D に追加した D'' というデータセットを作成してスコアの変化を観察する．

表 6 Feature Importance
CSV ファイル D' に LGBM Classifier を適用した場合の上位 10 個である．先頭に “*” が付いた特徴量が，我々の実装で生成されたドメイン特徴量またはそれを含む特徴量である．

特徴量	Importance
*RatioLoanToIncome_0	0.164
*ToalIncome_0	0.155
LoanAmount	0.124
NewFeature_0	0.119
Credit_History	0.101
ApplicantIncome	0.088
*NewFeature_3	0.039
Property_Area	0.026
Loan_Amount_Term	0.026
NewFeature_2	0.025

RatioLoantoIncome_0 = LoanAmount/(ApplicantIncome + CoapplicantIncome)
TotalIncome_0 = ApplicantIncome + CoapplicantIncome
NewFeature_0 = max(ApplicantIncome, CoapplicantIncome)
NewFeature_3 = max(ApplicantIncome, RatioLoantoIncome_0)
NewFeature_2 = max(CoapplicantIncome, LoanAmount)

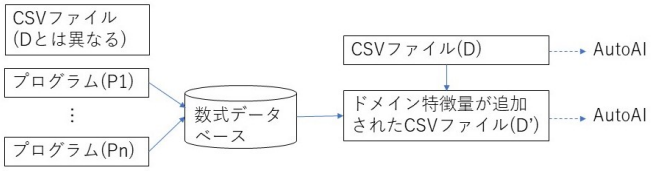

図 7 数式再利用の実験

- ドメイン特徴量が機械学習のモデル作成に効果があることを確認するために，データが少ない場合のスコアの変化を確認する．このために，D から半分のデータをランダムに抽出した場合についても実験を行う．

実験では，`alonsopg/loan_prediction` を除く表 2 のリポジトリにあるプログラムを $P_1, \cdots, P_n$ とする．そして，`alonsopg/loan_prediction` 内にある `german_credit.csv` を D として用いた．この CSV ファイルは，予測対象のカラム (Creditability) を含めて 21 カラムから構成され，1000 個のクレジットデータを含む．

D と数式データベースに基づいて D' を作成したところ，D' のカラム数は 50 個となった．D と D' に対して AutoAI を適用したところ，どちらの場合も XGB Classifier と Random Forest Classifier が自動選択され，ROC AUC スコアは表 7 の通りである．

このときの，Feature Importance が 0.25 以上の特徴量は表 8 の通りである．この中で以下の特徴量が我々の実装によって作られたドメイン特徴量である．

- RatioLoantoIncome_0
- TotalIncome_0

次にこれら 2 つの特徴量のみを CSV ファイル D に追加して D'' を作成して，同様に AutoAI を適用した場合のスコアが図 7 の D'' の欄である．自動的に選択されたアルゴリズムが D の場合と同じである．

次に，D から半分の 500 個のデータをランダムに抽出したデータセットを D_{500} とし，これに基づいて D'_{500}，D''_{500} を作成する．D'_{500} はドメイン特徴量を追加したデータセットであり，D''_{500} は 2 つのドメイン特徴量 RatioLoantoIncome_0 と TotalIncome_0 だけを追加したデータセットである．そして，これら 3 つのデータセットに対して，XGB Classifier と Random Forest Classifier を適用した．表 7 の D_{500},D'_{500},D''_{500} は，これら二つのアルゴリズムを適用した

表 7 数式再利用による予測スコアの比較
データ数 1000(オリジナル) の場合:

	D	D'	D''
XGB	0.792	0.795	0.793
Random Forest	0.797	0.784	0.793

データ数 500 の場合:

	D_{500}	D'_{500}	D''_{500}
XGB	0.757	0.772	0.759
Random Forest	0.759	0.748	0.765

結果である．

以降では、XGB Classifier と Random Forest Classifier の結果についてそれぞれ考察する．XGB Classifier に関しては，D よりも D', D'' のスコアがわずかであるが高く，また，Feature Importance を見ても上位にドメイン特徴量が表れている．このため，スコアの改善が小さいものの，抽出したドメイン特徴量の中には，予測スコアに貢献するものが含まれており，機械学習によって有用であったと考えている．また，D' よりも D'' のスコアが低いことから，重要な特徴量を十分には選択できなかったと考えている．

半分のデータセットに対して XGB Classifier を適用した結果では，データ数が減ったために元のデータセットに比べるとスコアが下がっているが，D_{500} が 0.757 であるのに対して，D'_{500} は 0.772 となり，大きくスコアが向上している．このとき，以下の二つのドメイン特徴量の Feature Importance がそれぞれ 3,4 位となっていた．

- RatioLoantoIncome_2 = Concurrent Credits/ (Account Balance + Account Balance)
- RatioLoantoIncome_1 = No of Credits at this Bank/ (Account Balance + Account Balance)

表 8 Feature Importance
CSV ファイル D' に XGB Classifier(上), Random Forest Classifier(下) を適用した場合に Importance が 0.25 以上の特徴量を抽出した．先頭に “*” が付いた特徴量が，我々の実装で生成されたドメイン特徴量またはそれを含む特徴量である．

XGB Classifier の場合:

特徴量	Importance
Account Balance	0.187
Guarantors	0.045
Payment Status of Previous Credit	0.041
Value Savings/Stocks	0.036
Duration of Credit (month)	0.032
*NewFeature_2	0.030
*NewFeature_7	0.030
Most valuable available asset	0.030
Length of current payment	0.028
Instalment per cent	0.028
Concurrent Credits	0.028
*RatioLoantoIncome_0	0.028
Purpose	0.028

Random Forest Classifier の場合:

特徴量	Importance
Account Balance	0.126
*TotalIncome_0	0.113
Length of current employment	0.050
Age(years)	0.047
Value Savings/Stocks	0.036
Purpose	0.035
*NewFeature_24	0.034
*NewFeature_13	0.031
Credit Amount	0.030
Payment Status of Previous Credit	0.029
Most valuable asset	0.025

NewFeature_2 = sum(RatioLoantoIncome_0, Age(years))
NewFeature_7 = sum(RatioLoantoIncome_0, tan(Age(years)))
NewFeature_13 = sum(RatioLoantoIncome_0, square(Age(years)))
NewFeature_24 = sum(square(Age(years)), RatioLoantoIncome_0)
RatioLoantoIncome_0 = Credit Amount/(Account Balance + Account Balance)
TotalIncome_0 = Account Balance + Account Balance

これら二つの特徴量が D''_{500} に含まれないため，D''_{500} のスコアは下がったと考えている．また，D''_{500} については，RatioLoantoIncome_0 の Feature Importance が 2 位となっていた．

Random Forest Classifier に関しては，D に適用した場合が最もスコアが高くなり，逆に D' を適用した場合が最もスコアが悪くなった．D'' は D' よりもスコアが良いことを考慮すると，追加されたドメイン特徴量がノイズになってしまいスコアの悪化を招いてしまったと考えている．このため，3.3 節で述べた通り，本論文で抽出したドメイン特徴量に対して，Feature Selection によってより効果のある特徴量のみ選択する必要がある．また，D における Feature Importance を調査すると，上位 10 個中の 2 位から 10 位は PCA(主成分分析) によって得られた特徴量であった．これらは AutoAI が自動で追加した特徴量である．ここで，1 位の特徴量は Account Balance であった．一方で，ドメイン特徴量を追加した場合には，表 8 に記載した通り PCA による特徴量は一つも追加されていなかった．このような PCA 適用の判断が AutoAI によって行われ，特徴量の追加に顕著な差がでた詳細は調査中である．

半分のデータセットに対して Random Forest Classifier を適用した結果においても，XGB Classifier の場合と同様に，データ数が減ったために元のデータセットに比べるとスコアが下がった．しかし，D_{500} が 0.759 であるのに対して，D''_{500} は 0.765 となりスコアの改善があった．そして，D''_{500} については，RatioLoantoIncome_0 の Feature Importance が 2 位となっていた．

以上の結果から，今回の実験の範囲においては，自動生成されたドメイン特徴量はデータ数が少ないほど，または，適切にドメイン特徴量を選択することによってスコア改善に役立つものと考えている．そして，既存の Automated Feature Engineering や Feature Selection の手法と組み合わせることによって，機械学習に有効な特徴量をより効率よく作成することが期待できると考えている．

5. まとめ

本論文では，既存の機械学習プログラムからドメイン知識として数式を抽出・収集し，収集した数式を別の機械学習の問題へ適用する方法を提案し，その実現性・有用性を確認するための予備実験を行った．この実験を通して，限定的ではあるが，動的解析を用いることによって，いくつかの Python プログラムから，機械学習に有用な数式の抽出を行うことができた．また，この数式を別の機械学習の問題へ再利用する実験を行った．この実験結果より，自動生成されたドメイン特徴量はデータ数が少ないほど，または，適切にドメイン特徴量を選択することによってスコア改善に役立つ可能性があり，既存の Automated Feature Engineering の手法の改善に貢献できるものと考えている．

参考文献

[1] Domingos, P.: A Few Useful Things to Know about Machine Learning, *Commun. ACM*, Vol. 55, No. 10 (online), DOI: 10.1145/2347736.2347755 (2012).
[2] Kanter, J. M. and Veeramachaneni, K.: Deep feature synthesis: Towards automating data science endeavors, *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 1–10 (2015).
[3] Katz, G., Shin, E. and Song, D.: ExploreKit: Automatic Feature Generation and Selection, pp. 979–984 (online), DOI: 10.1109/ICDM.2016.0123 (2016).
[4] Khurana, U., Turaga, D., Samulowitz, H. and Parthasrathy, S.: Cognito: Automated Feature Engineering for Supervised Learning, pp. 1304–1307 (online), DOI: 10.1109/ICDMW.2016.0190 (2016).
[5] Lam, H. T., Thiebaut, J.-M., Sinn, M., Chen, B., Mai, T. and Alkan, O.: One button machine for automating feature engineering in relational databases (2017).
[6] Azanzi, F. J. and Camara, G.: Knowledge extraction from source code based on Hidden Markov Model: application to EPICAM, *2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA)*, IEEE, pp. 1478–1485 (2017).
[7] Wang, C., Peng, X., Liu, M., Xing, Z., Bai, X., Xie, B. and Wang, T.: A learning-based approach for automatic construction of domain glossary from source code and documentation, *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 97–108 (2019).
[8] 立石孝彰，北山文彦，藤井邦和：レガシー・トランスフォーメーションのためのビジネスルール・リファクタリング，ソフトウェア工学の基礎: 日本ソフトウェア科学会 FOSE2004，JSSST FOSE (2004).
[9] Wong, W., Liu, W. and Bennamoun, M.: Ontology learning from text: A look back and into the future, *ACM Computing Surveys (CSUR)*, Vol. 44, No. 4, pp. 1–36 (2012).
[10] Rahm, E. and Bernstein, P. A.: A survey of approaches to automatic schema matching, *the VLDB Journal*, Vol. 10, No. 4, pp. 334–350 (2001).
[11] Bell, G. B. and Sethi, A.: Matching Records in a National Medical Patient Index, *Commun. ACM*, Vol. 44, No. 9, p. 83–88 (online), DOI: 10.1145/383694.383711 (2001).
[12] Guyon, I. and Elisseeff, A.: An introduction to variable and feature selection, *Journal of machine learning research*, Vol. 3, No. Mar, pp. 1157–1182 (2003).
[13] Beckett, D., Berners-Lee, T., Prud'hommeaux, E. and Carothers, G.: RDF 1.1 Turtle (2014).
[14] Harris, S. and Seaborne, A.: SPARQL 1.1 Query Language (2013).