令和二年度  
プロジェクトデザインⅢ プロジェクトレポート

YouTube動画の再生数予測モデルに効果的な指標の解析

金沢工業大学 工学部 情報工学科

4EP4-35 鶴瀬 和輝

指導教員 元木 光雄 准教授

令和2年12月25日提出

論文概要

近年，YouTubeに動画を投稿し，広告収益を得て生活を担うYouTuberという職業が存在する．この広告収益は，動画の再生数や評価の割合によって変化し，再生数や高評価の割合が多いほど収入も増えていく．これらの影響から動画の再生数を増やすためのアイデアが，日々多くのメディア(動画広告，セミナー，記事)で紹介されている．

しかし，実際にそれらの取り組みによってどれだけ試聴回数を増やすことができるのか定量データに基づいたノウハウ紹介は少なく，何が効果的なのかは分かっていない状況である．そこで，YouTube動画の再生数予測に効果的な指標を解析することによって，YouTube動画の再生数を増やすことができる指標を抽出できると考えた．

本論文では， YouTube動画の再生数予測モデルに効果的な指標の解析する方法を記載している．

プロジェクト実施記録

|  |  |  |
| --- | --- | --- |
| 期間 | 作業内容 | 作業時間 |
| 4月から7月 | 欠損データの補完と特徴量の作成 | 110時間 |
| 8月から10月 | テキストデータのベクトル化 | 50時間 |
| 11月から12月上旬 | LightGBMによる予測モデルの構築 | 130時間 |
| 12月下旬から1月 | 論文作成 | 40時間 |

目次

[第1章 序論 1](#_Toc59729679)

[第2章 準備 3](#_Toc59729680)

[2.1 YouTube動画視聴回数予測コンペティション 3](#_Toc59729681)

[2.2 機械学習フレームワークLightGBM 4](#_Toc59729682)

[第3章 基本的な解析手法 6](#_Toc59729683)

[3.1 欠損データの補完 6](#_Toc59729684)

[3.2 特徴量の作成 6](#_Toc59729685)

[3.3 LightGBMによる予測モデルの構築 8](#_Toc59729686)

[3.4 特徴量ごとの重要度の抽出 8](#_Toc59729687)

[第4章 実データへの適用 10](#_Toc59729688)

[4.1 予測モデルの評価 10](#_Toc59729689)

[4.1.1 評価方法 10](#_Toc59729690)

[4.1.2 評価結果 10](#_Toc59729691)

[4.2 特徴量の重要度 11](#_Toc59729692)

[第5章 結論 12](#_Toc59729693)

[参考文献 13](#_Toc59729694)

[謝辞 14](#_Toc59729695)

[付録A ソースコード 15](#_Toc59729696)

[A.1 データ読み込みと特徴量の作成 15](#_Toc59729697)

[A.1.1 create\_feature.py 15](#_Toc59729698)

[A.2 テキストデータのベクトル化 21](#_Toc59729699)

[A.2.1 morphological\_analysis.py 21](#_Toc59729700)

[A.2.2 tfidf\_svd.py 22](#_Toc59729701)

[A.3 LightGBMによる再生数を予測するモデルの構築 22](#_Toc59729702)

[A.3.1 predict\_lightgbm.py 22](#_Toc59729703)

1. 序論

近年，世界的なスマートフォンの普及とモバイル通信の高速化に伴い，様々な動画サービスが世界に広まった．代表的な動画サービスの一つにYouTubeがある．

最近では，YouTubeに動画を投稿し，動画の広告収益を得て生活を担うYouTuberと呼ばれる職業が存在する．この広告収益の値は，動画の再生数や評価の割合によって変化し，再生数や高評価の割合が多いほど増えていく．これらの影響から動画の再生数を増やすためのアイデアが，日々多くのメディア（動画広告，セミナー，記事）で紹介されている．

しかし，実際にそれらの取り組みによってどれだけ試聴回数を増やすことができるのか定量データに基づいたノウハウ紹介は少なく，何が効果的なのかは分かっていない状況である．そこで，YouTube動画の再生数予測に効果的な指標を解析することによって，YouTube動画の再生数を増やすことができる指標を抽出できると考えた．

本研究では，YouTube動画の再生数予測モデルに効果的な指標の解析を行う．この解析によって得られる情報は，今後YouTuberが動画を作成する際に予め任意のYouTuberの動画の月や年ごとの高評価数や低評価数の割合を知ることによって，どのような動画が長期的に，もしくは短期的に増やせるかを分かる．効果的な指標の抽出は，再生数と相関関係の高い特徴量から抽出できると考える．またそれらの特徴量を集計化し，それらの特徴量ごとの重要度を算出することによって，再生数を予測するのに効果的な指標が抽出できると考えた．

提案する手法としては，YouTube動画の再生数，高評価数，低評価数などのYouTube APIで取得できるメタデータを用意し，再生数と相関関係の高い特徴量を調べるため探索型データ解析を行う．次に使用するデータセットの前処理を行い，欠損値の補完をする．そしてテキストデータを機械学習モデルに学習できる形にするためテキストデータのベクトル化を行う．そのあと作成した特徴量から再生数を予測するために機械学習フレームワークLightGBMを使用し，再生数予測モデルを構築する．最後に構築したモデルから特徴量の重要度を算出することによって，再生数を予測するのに効果的な指標を抽出できる．

本論文の構成は以下の通りである．第２章では，YouTube動画視聴回数予測コンペティションと，機械学習フレームワークLightGBMについて説明を行う．第３章では，欠損データの補完と特徴量の作成，LightGBMによる予測モデルの構築，特徴量ごとの重要度の抽出について述べる．第４章では，実データへの適用，評価方法，評価結果，特徴量の重要度について述べる．第５章では，結論を述べる．

1. 準備

第２章では，YouTube動画視聴回数予測コンペティションと，機械学習フレームワークLightGBMの説明を行う．

* 1. YouTube動画視聴回数予測コンペティション

本節では，YouTube動画視聴回数予測コンペティションについて述べる．

データ分析のプラットフォームであるProbSpaceに「YouTube動画視聴回数予測コンペティション」が2020年4月から2020年6月の期間に開催された．本コンペでは，YouTube APIから取得できるメタデータを入力として，動画の視聴回数（再生数）を予測するアルゴリズムを開発する．表1 にYouTube APIから取得できるメタデータを示す．

1. YouTube APIから取得できるメタデータ

|  |  |
| --- | --- |
| 項目名 | 説明 |
| video\_id | 動画ごとに割り振られる一意のid |
| title | 動画のタイトル |
| publishedAt | 動画の投稿時間 |
| channelId | 動画を投稿したチャンネルのid |
| channelTitle | チャンネルのタイトル |
| categoryId | 動画カテゴリのid |
| collection\_date | データレコードの収集日 |
| tags | 動画に割り当てられたタグ |
| likes | 動画の高評価数 |
| dislikes | 動画の低評価数 |
| comment\_count | 動画のコメント数 |
| thumbnail\_link | 動画のサムネイルのリンク |
| comments\_disabled | コメントが許可されていない動画かどうか？ |
| ratings\_disabled | 評価が許可されていない動画かどうか？ |
| description | 動画の説明文 |
| y | 視聴回数(再生数) |

提供されるデータには，動画コンテンツの質的指標となる高評価数，低評価数，コメント数，視聴回数や，SEOとして重要とされているタイトル名，説明文，タグ，投稿時間といった情報が含まれている．訓練データのデータ数は19,720である．テストデータのデータ数は29,582であり視聴回数を除いてある．本研究では，このコンペティションで提供されたデータを用いる．

* 1. 機械学習フレームワークLightGBM

本節では，文献[1] [2] を参考にして，決定木とアンサンブル学習の説明を行う．

LightGBMは決定木アルゴリズムに基づいた勾配ブースティングフレームワークである．LightGBMは2016年にリリースされ，Kaggleなどのデータ分析コンペティションや機械学習実務において，これまでに多く利用されている．

LightGBMの仕組みとして「決定木」，「アンサンブル学習」，「勾配ブースティング」がある．

決定木とは，特定の特徴がよく現れるようなデータのかたまりを見つけ，その分類ルールを生成する機械学習の手法である．具体的には，目的変数と説明変数を設定し，目的変数の特徴的な領域が固まって存在するようなデータのグループを見つけていくのだが，複数の説明変数による条件でデータを分割していくことでそのデータ内における目的変数の特徴の濃度を高めることが．そうして得られた説明変数の条件で構成されるデータの分岐ルールを生成する手法が決定木である[1] ．

機械学習のモデルの性能を評価する指標としてバイアスとバリアンスの2種類がある．バイアスは実際値と予測値との誤差の平均であり，バリアンスは予測値がどれだけ散らばっているかを示す度合いである．よってバイアスとバリアンスの値が小さいほど，予測値と実際値の誤差が小さいということになる．そのため学習効率を上げるにはこの2つの指標の値を下げていく必要がある．

アンサンブル学習[2] は，複数のモデル（学習器）を組み合わせて一つの学習モデルを生成する手法であり，「バギング」，「ブースティング」，「スタッキング」と呼ばれる3つの手法に分類することが出来る．

バギングはバリアンスを下げるために行われ，学習データの情報を全て使うのでなく，ブートストラップ法で分割したデータを弱学習器で学習し，最終的な学習器に統合する．ブートストラップ法とは，復元抽出によってサブデータを作成する手法である．分類問題の場合は最大値，回帰問題の場合は平均をとる．並列処理が可能で，バギングを利用したアンサンブル学習にはランダムフォレストなどが知られている．

ブースティングはバイアスを下げるために行われ，前の学習に使用したデータを復元抽出して，逐次的に弱学習器を生成していく．よってバギングとは異なり，並列処理は出来ない．

スタッキングはモデルを積み上げていくことで，バイアスとバリアンスをバランスよく調整する手法である．まずブートストラップ法で得たデータセットをランダムに選んだ様々な学習器で学習させる．次に，この予測値を用いて，どの学習器を組み合わせると正解率が最大になるかを学習させる．より正解率の高い学習器同士を組み合わせることで，学習器をランダムで組み合わせるよりも精度が高くなる．

勾配ブースティングは前述のブースティングを拡張したものであり，直前の弱学習器までの予測結果と目的変数の損失関数の勾配を求め，損失関数の勾配に対して弱学習器を逐次的に学習していく手法である．

1. 基本的な解析手法

機械学習手法の一つである決定木の勾配ブースティングフレームワークにより，学習器を作成したのち，算出される重要度が高い特徴量を，再生数予測モデルに効果的な指標として考える．

第3章では，欠損データの補完と特徴量の作成，LightGBMによる予測モデルの構築，特徴量ごとの重要度の抽出について述べる．

* 1. 欠損データの補完

本節では，文献[3] を参考にして，探索型データ解析の説明を行う．

まず，本研究で使用する入力データを探索型データ解析により，再生数と相関関係の高い特徴量を見つける．探索的データ解析とは，統計学者J.W.Tukeyによって提唱されたもので，データの解釈にあたっては「まずモデルありき」ではなく，モデルを仮定する前に現実的な立場で，データの示唆する情報を多面的に捉えるという，解析初期のフェーズを重視したアプローチである[3] ．探索型データ解析により再生数と相関関係の高い特徴量は，高評価数，低評価数，コメント数であることが分かった．

次に，入力データには，高評価数，低評価数，コメント数の欠損値が全体の13%存在している，この部分が予測モデルの精度に影響していると考えられる．よって，欠損している箇所にLightGBMを使用し，高評価数，低評価数，コメント数それぞれの予測モデルを構築し，高評価数，低評価数，コメント数の予測値で欠損部分の補完を行った．

* 1. 特徴量の作成

本節では，文献[4] [5] を参考にして，TF-IDF，LSAの説明を行う．

まず，数値で表現される指標に対する特徴量の作成について述べる．特徴量はいくつかの指標を集計することで作成する．具体的に用いる指標は，カテゴリIDや動画を公開した年ごとの高評価数，低評価数，コメント数であり，集計方法にはトリム平均値，最小値，最大値，中央値などを用いる．また，期間ごと（１日あたり，１ヶ月あたり，1年あたり）の高評価数，低評価数，コメント数の特徴量，高評価数，低評価数，コメント数の実際値と予測値の差分などの特徴量を作成した．

次に，テキストで表現される指標に対する特徴量の作成について述べる．まず，チャンネル名，動画タイトル，説明文のテキスト特徴量に対し，MeCabによる形態素解析を行う．形態素解析とは，対象言語の文法や辞書と呼ばれる単語の品詞情報をもとに，文章を語に分け，それらの語の品詞を判別することをいう．また，品詞を指定して単語を取り出すことも可能である．

表2 に例文と形態素解析によって取り出される単語群を示す．本研究では，単語単体で意味のある名詞，動詞，形容詞を抽出する．

1. 例文と形態素解析によって取り出された単語群

|  |  |
| --- | --- |
| 例文 | 取り出された単語群 |
| 野球の試合観戦に行く | 野球，試合，観戦，行く |
| アーティストのライブが始まる | アーティスト，ライブ，始まる |

次に，形態素解析を行ったテキスト特徴量をベクトル化するために，TF-IDF手法を使用する．TF-IDFとは，2種類の重みづけ指標を組み合わせたものである．具体的には，TF (Term Frequency索引語頻度) は「より多く出現する語は，より重要な語である」という考え方に基づき，n 回出現した語は 1 回しか出現していない語よりも log10(n)＋1 倍重要であるとみなす．また，文書の特定力向上のためには，助詞などの文書集合の全体もしくは非常に多くの文書に出現する語よりも，一つもしくは少数の文書にしか出現しないような語を用いる場合がある．これが，IDF (Inverse Document Frequency逆文書頻度) であり，「ある語に対して，その語の出現回数が少なければ少ないほど，より大きな重みを与える」という考え方に基づいている[4] ．本研究ではTF-IDF手法を，Pythonの外部ライブラリであるscikit-learnのTfidfVectorizerによって行う．TF-IDFを行った結果，チャンネル名，動画タイトルでは3万次元，説明文では20万次元を超える単語の重要度の特徴量を作成した．

単語の重要度の特徴量を作成することは出来たが，次元数が多く，中には重複したデータなども入っており，これらのデータは学習に時間がかかるだけでなく，ノイズになり過学習を引き起こす可能性もある．よって次元圧縮(次元削除)を行う．本研究では，次元圧縮の方法としてLSA(潜在的意味解析)を用いる．LSAとは，情報検索の分野において，言葉の同義性や多様性に対処するために発展した統計的技法である[5] ．本研究では，SVDをPythonの外部ライブラリであるscikit-learnのTruncatedSVDによって行う．TruncatedSVDでは出力データの推奨されている次元数は100である．よってチャンネル名，動画タイトル，説明文のTF-IDFによって作成した単語の重要度の特徴量を次元圧縮し，次元数を100にした．本研究では，TruncatedSVDによって次元圧縮された後に作成された特徴量のことをSVD特徴量とする．

* 1. LightGBMによる予測モデルの構築

本節では，文献[6] を参考にして，ハイパーパラメータの説明を行う．

3.2で作成した特徴量，訓練データをLightGBMの入力データとして，再生数予測モデルを構築する．具体的に入力データ数は19,720，特徴量が616種類である．しかし，入力データだけでは，モデル訓練を行うのに不十分なのでLightGBMのハイパーパラメータを調整する．ハイパーパラメータとは，各アルゴリズムに付随して，アルゴリズムの挙動を制御するための値である[6] ．モデルの学習実行前にハイパーパラメータを調整することでモデルの性能向上や過学習の抑制，効率の良い学習などが期待できる．表3 に本研究で使用したLightGBMの過学習を防ぐハイパーパラメータの一部を示す．

1. 過学習を防ぐハイパーパラメータ

|  |  |
| --- | --- |
| パラメータ名 | パラメータの値 |
| min\_data\_in\_leaf | 10 |
| min\_gain\_to\_split | 0.01 |
| max\_depth | -1 |
| feature\_fraction | 0.6 |

* 1. 特徴量ごとの重要度の抽出

本節では，文献[7] [8] を参考にして，ジニ不純度と特徴量ごとの重要度を抽出する方法についての説明を行う．

3.3で構築した予測モデルから再生数を予測するのに効果的な特徴量の重要度を抽出する．特徴量の重要度とは，その特徴量の分類がターゲットの分類にどれぐらい寄与しているかを測る指標であり，ジニ不純度を基にして計算できる決定木の任意のノードのジニ不純度*IG*(*t*)は，  
で求められる．ただし，はカテゴリ数を表し， は決定木のノード総数に対して，カテゴリに属するデータの割合を表す[7] ．ジニ不純度は0から1をとり，小さいほど対応する特徴量が重要となる．このジニ不純度を利用した特徴量の重要度を算出するまでの計算式を以下に示す[8] ．

\

は，各ノードにおけるジニ不純度の減少を計算しており，は，あるノードiにおけるサンプル数，，は，あるノードの子ノードのうちの左右のノードにおけるサンプル数，はあるノードにおける不純度，，は，あるノードの子ノードのうち，左右のノードにおけるジニ不純度を表す．は，を利用し，分母が全ノードにおける不純度の減少の総和であり，分子が対象の特徴量による分岐ノードでのジニ不純度の減少の総和から求められる．

算出される特徴量の重要度の数値が高ければ高いほど，重要度として適切である．本研究では，この特徴量の重要度の計算をLightGBMのfeature\_importanceメソッドによって行い，正規化した特徴量ごとの重要度を抽出する．

1. 実データへの適用

第4章では，実データを用いて，予測モデルを評価する方法，特徴量の重要度について述べる．

* 1. 予測モデルの評価

機械学習を行う際，訓練データでの予測モデルの性能が良いにも関わらず，テストデータでの予測モデルの性能が悪くなってしまう過学習が起こることがある．そこで，予測モデルの汎化性能を実験的に評価する．

* + 1. 評価方法

予測モデルの汎化性能を評価するために交差検証法のK-分割交差検証を用いた．K-分割交差検証とは，データをK個に分割してそのうちの一つをテストデータにし，残りのデータを訓練データとして正解率の評価を行う．これをK個のデータすべてが1回ずつテストデータになるようにK回学習を行い，予測モデルの精度の平均をとる手法である．

また，予測モデルの評価指標は，回帰モデルの最も一般的な性能評価指標であるRMSE（二乗平均平方根誤差）とする．RMSEとは，実際値と予測値の誤差の二乗の総和の平均値のルートをとったものである．よってRMSEが0に近いほどモデルとしての性能が良いということになる．はデータの個数，は実際値，iは予測値を表すRMSEの計算式を以下の式に示す[9] ．

* + 1. 評価結果

本研究では，4.1.1で述べたK-分割交差検証のKの値を8とし，LightGBMによるモデル訓練を8回実施した．それぞれの予測モデルのRMSEを図1 に示す．また，モデル訓練を8回実施した中で精度が一番良い時(K=2)における上位10個の特徴量の重要度を表4 に示す．

1. 予測モデルのRMSE
2. K=2における上位10個の特徴量の重要度

|  |  |
| --- | --- |
| 特徴量名 | 特徴量の重要度 |
| year\_min\_diff\_dislikes | 0.2293 |
| Dislikes | 0.1337 |
| categoryId\_min\_diff\_dislikes | 0.0979 |
| likes\_by\_year | 0.0547 |
| year\_trim\_mean\_diff\_dislikes | 0.0522 |
| year\_min\_diff\_dislikes | 0.0407 |
| likes\_by\_month | 0.0229 |
| likes\_by\_day | 0.0145 |
| year\_trim\_mean\_diff\_likes | 0.0141 |
| ratings\_disabled\_mean\_ratio\_likes | 0.0127 |

図1 より，構築した全ての予測モデルのRMSEが0.7前後となっており，今回の研究で構築した予測モデルはモデルとしての性能があまり高くないという結果となった．

* 1. 特徴量の重要度

表4 からはlikes\_by\_year，likes\_by\_monthなどの期間ごとの評価数やyear\_trim\_mean\_diff\_dislikesなどのトリム平均値を使った集計特徴量が高く，その中でも低評価に関する特徴量が重要度として効果があることが分かった．

1. 結論

本研究では，YouTube APIのメタデータからLightGBMによる再生数を予測するモデルを構築することで，予測モデルから特徴量の重要度を抽出し，再生数を予測するのに効果的な指標を見つけることができた．しかし，モデルとしての性能は高くはないので，今後の課題として，LightGBMのモデルに入力する特徴量を高評価数，低評価数，コメント数などのエンゲージメントに関わる特徴量に厳選する．そして，画像データなどの利用がある．

参考文献

1. 株式会社アナリティクスデザインラボ，決定木，http://www.analyticsdlab.co.jp/column/decisiontree.html，2020年11月20日参照．
2. AI研究所，アンサンブル学習，https://ai-kenkyujo.com/term/ensemble-learning/，2020年11月20日参照．
3. 株式会社NTTデータ数理システム，探索型データ解析 (Exploratory Data Analysis)，https://www.msi.co.jp/splus/products/eda.html，2020年11月23日参照．
4. 木村美紀，TF-IDFを用いた文書分類の試み，文学研究論集，Vol. 48，pp. 2-3, 2018．
5. 尾﨑亮太，テキストマイニングによる文書の類似度計算に関する研究，http://www.gifu-nct.ac.jp/elec/deguchi/sotsuron/osaki/node14.html，  
   2020年12月19日参照．
6. キカガク，機械学習 実践 （ハイパーパラメータ調整），https://www.kikagaku.ai/tutorial/basic\_of\_machine\_learning/learn/machine\_learning\_hyperparameters，2020年11月24日参照．
7. 近藤悠介，教育データ分析入門2 （第4回），https://www.f.waseda.jp/yusukekondo/IEDA19/IEDA2\_05.html，2020年12月10日参照．
8. mlcourse.ai，Topic5．Ensembles and random forest Part3．Feature Importance，  
   https://mlcourse.ai/articles/topic5-part3-feature-importance/，2020年12月18日参照．
9. biostatistics，回帰分析の評価指標，https://stats.biopapyrus.jp/glm/lm-evaluation.html，2020年12月20日参照．

謝辞

本研究にあたり，ご指導をいただいた元木光雄准教授に深く感謝いたします．また，元木研究室の皆様にも様々なご意見をいただき感謝いたします．

ソースコード

データ読み込みと特徴量の作成

create\_feature.py

ProbSpaceから提供されているYouTube API メタデータ，高評価数，低評価数，コメント数の予測値のデータの読み込みと特徴量の作成を行うプログラムを図2 に示す．

|  |
| --- |
| import pandas as pd  import numpy as np  import statistics  import os  import random  import warnings  warnings.filterwarnings('ignore')  from sklearn.preprocessing import LabelEncoder  %matplotlib inline  pd.set\_option('display.max\_rows', 1000)  pd.set\_option('display.max\_columns', 1000)  pd.set\_option('display.width', 1000)  pd.set\_option('display.max\_colwidth', 1000)  pd.set\_option("display.precision", 8)  def seed\_everything(seed):  random.seed(seed)  os.environ['PYTHONHASHSEED'] = str(seed)  np.random.seed(seed)    # seed  seed = 817  seed\_everything(seed)  # load train test  train = pd.read\_csv('../../data/input/probspace/train\_data.csv')  train['y\_bin'] = pd.cut(train['y'], [0, 10, 100,1000,10000,100000,1000000,10000000000], labels=[1,2,3,4,5,6,7])  train['y\_bin'] = train['y\_bin'].astype(int)  test = pd.read\_csv('../../data/input/probspace/test\_data.csv')  df = pd.concat([train,test],axis=0).reset\_index(drop=True)  print ('train',train.shape)  print ('test',test.shape)  df['comments\_ratings'] = df['comments\_disabled'].astype(str)+df['ratings\_disabled'].astype(str)    for c in ['channelId','channelTitle','collection\_date','description','tags','comments\_disabled','ratings\_disabled','comments\_ratings']:  lbl = LabelEncoder()  df[c+'\_encoder'] = lbl.fit\_transform(df[c].astype(str))    df['length\_title'] = [len(d) for d in df['title']]    # use predicted dislikes,likes,comment\_out  dislikes\_pred = pd.read\_csv('../../data/input/probspace/dislikes\_pred\_0623.csv')  likes\_pred = pd.read\_csv('../../data/input/probspace/likes\_pred\_0623.csv')  comments\_pred = pd.read\_csv('../../data/input/probspace/comment\_count\_pred\_0623.csv')  df = df.merge(dislikes\_pred,on=['video\_id'],how='left')  df = df.merge(likes\_pred,on=['video\_id'],how='left')  df = df.merge(comments\_pred,on=['video\_id'],how='left')  df['diff\_dislikes'] = df['dislikes'] - df['dislikes\_pred']  df['diff\_likes'] = df['likes'] - df['likes\_pred']  df['diff\_comments'] = df['comment\_count'] - df['comment\_count\_pred']  df['original\_dislikes'] = df['dislikes']  df['original\_likes'] = df['likes']  df['original\_comment\_count'] = df['comment\_count']  # disableされている値を予測した値に入れ替える．  df.loc[df['ratings\_disabled']==True,'dislikes'] = df.loc[df['ratings\_disabled']==True,'dislikes\_pred']  df.loc[df['ratings\_disabled']==True,'likes'] = df.loc[df['ratings\_disabled']==True,'likes\_pred']  df.loc[df['comments\_disabled']==True,'comment\_count'] = df.loc[df['comments\_disabled']==True,'comment\_count\_pred']  # timestamp transformation  df["c\_date"] = "20" + df["collection\_date"]  df["c\_date"] = pd.to\_datetime(df["c\_date"], utc=True, format="%Y.%d.%m")  df["c\_year"] = df["c\_date"].dt.year  df["c\_month"] = df["c\_date"].dt.month  df["c\_day"] = df["c\_date"].dt.day  df["c\_dayofweek"] = df["c\_date"].dt.dayofweek  df["publishedAt"] = pd.to\_datetime(df["publishedAt"],utc=True, format="%Y-%m-%d")  df["past"] = (df["publishedAt"].dt.floor("D").max() - df["publishedAt"].dt.floor("D")).dt.days  df["year"] = df["publishedAt"].dt.year  df["month"] = df["publishedAt"].dt.month  df["weekofyear"] = df["publishedAt"].dt.weekofyear  df["day"] = df["publishedAt"].dt.day  df["dayofweek"] = df["publishedAt"].dt.dayofweek  df["hour"] = df["publishedAt"].dt.hour  df["minute"] = df["publishedAt"].dt.minute  # 公開されてからの収集するまで日数  df['seconds\_from\_publish'] = (df['c\_date'] - df['publishedAt']).dt.seconds  df['days\_from\_publish'] = (df['c\_date'] - df['publishedAt']).dt.days  df['months\_from\_publish'] = (df['c\_date'] - df['publishedAt']).dt.days // 30  df['years\_from\_publish'] = (df['c\_date'] - df['publishedAt']).dt.days // 365  # YouTubeが解説されてから公開までの日数  df['days\_from\_publish\_start'] = (df['publishedAt'] - df['publishedAt'].min()).dt.days  df['month\_from\_publish\_start'] = (df['publishedAt'] - df['publishedAt'].min()).dt.days // 30  df['year\_from\_publish\_start'] = (df['publishedAt'] - df['publishedAt'].min()).dt.days // 365  # データ収取を始めてからの日数  df['days\_from\_cdate\_start'] = (df['c\_date'] - df['c\_date'].min()).dt.days  # youtubeができてからの1日あたりのlike数，dislike数，コメント数  df['like\_per\_published\_day'] = (df['likes'] / df['days\_from\_publish\_start']).replace([np.inf, -np.inf], 0)  df['dislike\_per\_published\_day'] = (df['dislikes'] / df['days\_from\_publish\_start']).replace([np.inf, -np.inf], 0)  df['comment\_count\_per\_published\_day'] = (df['comment\_count'] / df['days\_from\_publish\_start']).replace([np.inf, -np.inf], 0)  # youtubeができてからの1ヶ月あたりのlike数，dislike数，コメント数  df['like\_per\_published\_month'] = (df['likes'] / df['month\_from\_publish\_start']).replace([np.inf, -np.inf], 0)  df['dislike\_per\_published\_month'] = (df['dislikes'] / df['month\_from\_publish\_start']).replace([np.inf, -np.inf], 0)  df['comment\_count\_per\_published\_month'] = (df['comment\_count'] / df['month\_from\_publish\_start']).replace([np.inf, -np.inf], 0)  # youtubeができてからの1年あたりのlike数，dislike数，コメント数  df['like\_per\_published\_year'] = (df['likes'] / df['year\_from\_publish\_start']).replace([np.inf, -np.inf], 0)  df['dislike\_per\_published\_year'] = (df['dislikes'] / df['year\_from\_publish\_start']).replace([np.inf, -np.inf], 0)  df['comment\_count\_per\_published\_year'] = (df['comment\_count'] / df['year\_from\_publish\_start']).replace([np.inf, -np.inf], 0)  # interaction  df['comments\_disabled'] = df['comments\_disabled'].map(lambda x:1 if x==True else 0)  df['ratings\_disabled'] = df['ratings\_disabled'].map(lambda x:1 if x==True else 0)  df['likes\_comments'] = df['likes'] \* df['comments\_disabled']  df['dislikes\_comments'] = df['dislikes'] \* df['comments\_disabled']  df['comment\_count\_ratings'] = df['comment\_count'] \* df['ratings\_disabled']  df['comments\_ratings\_disabled'] = df['comments\_disabled'] + df['ratings\_disabled']  df['diff\_likes\_dislikes'] = df['likes'] - df['dislikes']  df['ratio\_likes\_dislikes'] = df['likes'] / (df['dislikes'] + 1)  df['ratio\_likes\_comment\_count'] = df['likes'] / (df['comment\_count'] + 1)  df['ratio\_dislikes\_comment\_count'] = df['dislikes'] / (df['comment\_count'] + 1)  # 1日あたりのlike数，dislike数，コメント数  df['likes\_by\_day'] = df['likes'] / df['days\_from\_publish']  df['dislikes\_by\_day'] = df['dislikes'] / df['days\_from\_publish']  df['comments\_by\_day'] = df['comment\_count'] / df['days\_from\_publish']  # 1ヶ月あたりのlike数，dislike数，コメント数  df['likes\_by\_month'] = df['likes'] / df['months\_from\_publish']  df['dislikes\_by\_month'] = df['dislikes'] / df['months\_from\_publish']  df['comments\_by\_month'] = df['comment\_count'] / df['months\_from\_publish']  # 1年あたりのlike数，dislike数，コメント数  df['likes\_by\_year'] = df['likes'] / df['years\_from\_publish']  df['dislikes\_by\_year'] = df['dislikes'] / df['years\_from\_publish']  df['comments\_by\_year'] = df['comment\_count'] / df['years\_from\_publish']  # 標準得点  df['likes\_std\_score'] = (df['likes'] - df['likes'].mean()) / statistics.stdev(list(df['likes']))  df['dislikes\_std\_score'] = (df['dislikes'] - df['dislikes'].mean()) / statistics.stdev(list(df['dislikes']))  df['commentss\_std\_score'] = (df['comment\_count'] - df['comment\_count'].mean()) / statistics.stdev(list(df['comment\_count']))  def agg(df,agg\_cols):  for c in tqdm(agg\_cols):  new\_feature = '{}\_{}\_{}'.format('\_'.join(c['groupby']), c['agg'], c['target'])  if c['agg'] == 'mean\_diff':  df[new\_feature] = df.groupby(c['groupby'])[c['target']].transform('mean') - df[c['target']]  df[new\_feature+'\_rank'] = df.groupby(c['groupby'])[c['target']].rank()  elif c['agg'] == 'mean\_ratio':  df[new\_feature] = df.groupby(c['groupby'])[c['target']].transform('mean') / (1+df[c['target']])  df[new\_feature+'\_rank'] = df.groupby(c['groupby'])[c['target']].rank()  elif c['agg'] == 'trim\_mean':  df[new\_feature] = df.groupby(c['groupby'])[c['target']].transform(lambda x: stats.trim\_mean(x, 0.1))  df[new\_feature+'\_rank'] = df.groupby(c['groupby'])[c['target']].rank()  elif c['agg'] == 'trim\_mean\_diff':  df[new\_feature] = df.groupby(c['groupby'])[c['target']].transform(lambda x: stats.trim\_mean(x, 0.1)) - df[c['target']]  df[new\_feature+'\_rank'] = df.groupby(c['groupby'])[c['target']].rank()  elif c['agg'] == 'max\_diff':  df[new\_feature] = df.groupby(c['groupby'])[c['target']].transform('max') - df[c['target']]  df[new\_feature+'\_rank'] = df.groupby(c['groupby'])[c['target']].rank()  elif c['agg'] == 'max\_ratio':  df[new\_feature] = df.groupby(c['groupby'])[c['target']].transform('max') / (1+df[c['target']])  df[new\_feature+'\_rank'] = df.groupby(c['groupby'])[c['target']].rank()  elif c['agg'] == 'min\_diff':  df[new\_feature] = df.groupby(c['groupby'])[c['target']].transform('min')- df[c['target']]  df[new\_feature+'\_rank'] = df.groupby(c['groupby'])[c['target']].rank()  elif c['agg'] == 'min\_ratio':  df[new\_feature] = df.groupby(c['groupby'])[c['target']].transform('min')/ (1+df[c['target']])  df[new\_feature+'\_rank'] = df.groupby(c['groupby'])[c['target']].rank()  elif c['agg'] == 'max\_min\_diff':  df[new\_feature] = df.groupby(c['groupby'])[c['target']].transform('max') - df.groupby(c['groupby'])[c['target']].transform('min')  elif c['agg'] == 'max\_min\_ratio':  df[new\_feature] = df.groupby(c['groupby'])[c['target']].transform('max') / (1+df.groupby(c['groupby'])[c['target']].transform('min'))  elif c['agg'] == 'median\_diff':  df[new\_feature] = df.groupby(c['groupby'])[c['target']].transform('median') - df[c['target']]  elif c['agg'] == 'median\_ratio':  df[new\_feature] = df.groupby(c['groupby'])[c['target']].transform('median') / (1+df[c['target']])  return df  agg\_cols = [  {'groupby': ['categoryId'], 'target':'likes', 'agg':'trim\_mean'},  {'groupby': ['categoryId'], 'target':'likes', 'agg':'trim\_mean\_diff'},  {'groupby': ['categoryId'], 'target':'likes', 'agg':'max\_diff'},  {'groupby': ['categoryId'], 'target':'likes', 'agg':'max\_ratio'},  {'groupby': ['categoryId'], 'target':'likes', 'agg':'mean\_ratio'},  {'groupby': ['categoryId'], 'target':'likes', 'agg':'mean\_diff'},  {'groupby': ['categoryId'], 'target':'likes', 'agg':'min\_diff'},  {'groupby': ['categoryId'], 'target':'likes', 'agg':'min\_ratio'},  {'groupby': ['categoryId'], 'target':'likes', 'agg':'max\_min\_diff'},  {'groupby': ['categoryId'], 'target':'likes', 'agg':'max\_min\_ratio'},  {'groupby': ['categoryId'], 'target':'likes', 'agg':'median\_diff'},  {'groupby': ['categoryId'], 'target':'likes', 'agg':'median\_ratio'},    {'groupby': ['categoryId'], 'target':'dislikes', 'agg':'trim\_mean'},  {'groupby': ['categoryId'], 'target':'dislikes', 'agg':'trim\_mean\_diff'},  {'groupby': ['categoryId'], 'target':'dislikes', 'agg':'max\_diff'},  {'groupby': ['categoryId'], 'target':'dislikes', 'agg':'max\_ratio'},  {'groupby': ['categoryId'], 'target':'dislikes', 'agg':'mean\_ratio'},  {'groupby': ['categoryId'], 'target':'dislikes', 'agg':'mean\_diff'},  {'groupby': ['categoryId'], 'target':'dislikes', 'agg':'min\_diff'},  {'groupby': ['categoryId'], 'target':'dislikes', 'agg':'min\_ratio'},  {'groupby': ['categoryId'], 'target':'dislikes', 'agg':'max\_min\_diff'},  {'groupby': ['categoryId'], 'target':'dislikes', 'agg':'max\_min\_ratio'},  {'groupby': ['categoryId'], 'target':'dislikes', 'agg':'median\_diff'},  {'groupby': ['categoryId'], 'target':'dislikes', 'agg':'median\_ratio'},    {'groupby': ['categoryId'], 'target':'comment\_count', 'agg':'trim\_mean'},  {'groupby': ['categoryId'], 'target':'comment\_count', 'agg':'trim\_mean\_diff'},  {'groupby': ['categoryId'], 'target':'comment\_count', 'agg':'max\_diff'},  {'groupby': ['categoryId'], 'target':'comment\_count', 'agg':'max\_ratio'},  {'groupby': ['categoryId'], 'target':'comment\_count', 'agg':'mean\_ratio'},  {'groupby': ['categoryId'], 'target':'comment\_count', 'agg':'mean\_diff'},  {'groupby': ['categoryId'], 'target':'comment\_count', 'agg':'min\_diff'},  {'groupby': ['categoryId'], 'target':'comment\_count', 'agg':'min\_ratio'},  {'groupby': ['categoryId'], 'target':'comment\_count', 'agg':'max\_min\_diff'},  {'groupby': ['categoryId'], 'target':'comment\_count', 'agg':'max\_min\_ratio'},  {'groupby': ['categoryId'], 'target':'comment\_count', 'agg':'median\_diff'},  {'groupby': ['categoryId'], 'target':'comment\_count', 'agg':'median\_ratio'},  {'groupby': ['year'], 'target':'likes', 'agg':'trim\_mean'},  {'groupby': ['year'], 'target':'likes', 'agg':'trim\_mean\_diff'},  {'groupby': ['year'], 'target':'likes', 'agg':'max\_diff'},  {'groupby': ['year'], 'target':'likes', 'agg':'max\_ratio'},  {'groupby': ['year'], 'target':'likes', 'agg':'mean\_ratio'},  {'groupby': ['year'], 'target':'likes', 'agg':'mean\_diff'},  {'groupby': ['year'], 'target':'likes', 'agg':'min\_diff'},  {'groupby': ['year'], 'target':'likes', 'agg':'min\_ratio'},  {'groupby': ['year'], 'target':'likes', 'agg':'max\_min\_diff'},  {'groupby': ['year'], 'target':'likes', 'agg':'max\_min\_ratio'},  {'groupby': ['year'], 'target':'likes', 'agg':'median\_diff'},  {'groupby': ['year'], 'target':'likes', 'agg':'median\_ratio'},  {'groupby': ['year'], 'target':'dislikes', 'agg':'trim\_mean'},  {'groupby': ['year'], 'target':'dislikes', 'agg':'trim\_mean\_diff'},  {'groupby': ['year'], 'target':'dislikes', 'agg':'max\_diff'},  {'groupby': ['year'], 'target':'dislikes', 'agg':'max\_ratio'},  {'groupby': ['year'], 'target':'dislikes', 'agg':'mean\_ratio'},  {'groupby': ['year'], 'target':'dislikes', 'agg':'mean\_diff'},  {'groupby': ['year'], 'target':'dislikes', 'agg':'min\_diff'},  {'groupby': ['year'], 'target':'dislikes', 'agg':'min\_ratio'},  {'groupby': ['year'], 'target':'dislikes', 'agg':'max\_min\_diff'},  {'groupby': ['year'], 'target':'dislikes', 'agg':'max\_min\_ratio'},  {'groupby': ['year'], 'target':'dislikes', 'agg':'median\_diff'},  {'groupby': ['year'], 'target':'dislikes', 'agg':'median\_ratio'},    {'groupby': ['year'], 'target':'comment\_count', 'agg':'trim\_mean'},  {'groupby': ['year'], 'target':'comment\_count', 'agg':'trim\_mean\_diff'},  {'groupby': ['year'], 'target':'comment\_count', 'agg':'max\_diff'},  {'groupby': ['year'], 'target':'comment\_count', 'agg':'max\_ratio'},  {'groupby': ['year'], 'target':'comment\_count', 'agg':'mean\_ratio'},  {'groupby': ['year'], 'target':'comment\_count', 'agg':'mean\_diff'},  {'groupby': ['year'], 'target':'comment\_count', 'agg':'min\_diff'},  {'groupby': ['year'], 'target':'comment\_count', 'agg':'min\_ratio'},  {'groupby': ['year'], 'target':'comment\_count', 'agg':'max\_min\_diff'},  {'groupby': ['year'], 'target':'comment\_count', 'agg':'max\_min\_ratio'},  {'groupby': ['year'], 'target':'comment\_count', 'agg':'median\_diff'},  {'groupby': ['year'], 'target':'comment\_count', 'agg':'median\_ratio'},    {'groupby': ['channelTitle\_encoder'], 'target':'diff\_likes\_dislikes', 'agg':'trim\_mean'},  {'groupby': ['channelTitle\_encoder'], 'target':'diff\_likes\_dislikes', 'agg':'trim\_mean\_diff'},  {'groupby': ['channelTitle\_encoder'], 'target':'diff\_likes\_dislikes', 'agg':'max\_diff'},  {'groupby': ['channelTitle\_encoder'], 'target':'diff\_likes\_dislikes', 'agg':'max\_ratio'},  {'groupby': ['channelTitle\_encoder'], 'target':'diff\_likes\_dislikes', 'agg':'mean\_ratio'},  {'groupby': ['channelTitle\_encoder'], 'target':'diff\_likes\_dislikes', 'agg':'mean\_diff'},  {'groupby': ['channelTitle\_encoder'], 'target':'diff\_likes\_dislikes', 'agg':'min\_diff'},  {'groupby': ['channelTitle\_encoder'], 'target':'diff\_likes\_dislikes', 'agg':'min\_ratio'},  {'groupby': ['channelTitle\_encoder'], 'target':'diff\_likes\_dislikes', 'agg':'max\_min\_diff'},  {'groupby': ['channelTitle\_encoder'], 'target':'diff\_likes\_dislikes', 'agg':'max\_min\_ratio'},  {'groupby': ['channelTitle\_encoder'], 'target':'diff\_likes\_dislikes', 'agg':'median\_diff'},  {'groupby': ['channelTitle\_encoder'], 'target':'diff\_likes\_dislikes', 'agg':'median\_ratio'},  {'groupby': ['categoryId'], 'target':'diff\_likes\_dislikes', 'agg':'trim\_mean'},  {'groupby': ['categoryId'], 'target':'diff\_likes\_dislikes', 'agg':'trim\_mean\_diff'},  {'groupby': ['categoryId'], 'target':'diff\_likes\_dislikes', 'agg':'max\_diff'},  {'groupby': ['categoryId'], 'target':'diff\_likes\_dislikes', 'agg':'max\_ratio'},  {'groupby': ['categoryId'], 'target':'diff\_likes\_dislikes', 'agg':'mean\_ratio'},  {'groupby': ['categoryId'], 'target':'diff\_likes\_dislikes', 'agg':'mean\_diff'},  {'groupby': ['categoryId'], 'target':'diff\_likes\_dislikes', 'agg':'min\_diff'},  {'groupby': ['categoryId'], 'target':'diff\_likes\_dislikes', 'agg':'min\_ratio'},  {'groupby': ['categoryId'], 'target':'diff\_likes\_dislikes', 'agg':'max\_min\_diff'},  {'groupby': ['categoryId'], 'target':'diff\_likes\_dislikes', 'agg':'max\_min\_ratio'},  {'groupby': ['categoryId'], 'target':'diff\_likes\_dislikes', 'agg':'median\_diff'},  {'groupby': ['categoryId'], 'target':'diff\_likes\_dislikes', 'agg':'median\_ratio'},  {'groupby': ['year'], 'target':'diff\_likes\_dislikes', 'agg':'trim\_mean'},  {'groupby': ['year'], 'target':'diff\_likes\_dislikes', 'agg':'trim\_mean\_diff'},  {'groupby': ['year'], 'target':'diff\_likes\_dislikes', 'agg':'max\_diff'},  {'groupby': ['year'], 'target':'diff\_likes\_dislikes', 'agg':'max\_ratio'},  {'groupby': ['year'], 'target':'diff\_likes\_dislikes', 'agg':'mean\_ratio'},  {'groupby': ['year'], 'target':'diff\_likes\_dislikes', 'agg':'mean\_diff'},  {'groupby': ['year'], 'target':'diff\_likes\_dislikes', 'agg':'min\_diff'},  {'groupby': ['year'], 'target':'diff\_likes\_dislikes', 'agg':'min\_ratio'},  {'groupby': ['year'], 'target':'diff\_likes\_dislikes', 'agg':'max\_min\_diff'},  {'groupby': ['year'], 'target':'diff\_likes\_dislikes', 'agg':'max\_min\_ratio'},  {'groupby': ['year'], 'target':'diff\_likes\_dislikes', 'agg':'median\_diff'},  {'groupby': ['year'], 'target':'diff\_likes\_dislikes', 'agg':'median\_ratio'},    {'groupby': ['ratings\_disabled'], 'target':'likes', 'agg':'mean\_diff'},  {'groupby': ['ratings\_disabled'], 'target':'likes', 'agg':'mean\_ratio'},    {'groupby': ['ratings\_disabled'], 'target':'dislikes', 'agg':'mean\_diff'},  {'groupby': ['ratings\_disabled'], 'target':'dislikes', 'agg':'mean\_ratio'},    {'groupby': ['ratings\_disabled'], 'target':'comment\_count', 'agg':'mean\_diff'},  {'groupby': ['ratings\_disabled'], 'target':'comment\_count', 'agg':'mean\_ratio'},    {'groupby': ['comments\_disabled'], 'target':'likes', 'agg':'mean\_diff'},  {'groupby': ['comments\_disabled'], 'target':'likes', 'agg':'mean\_ratio'},  {'groupby': ['comments\_disabled'], 'target':'dislikes', 'agg':'mean\_diff'},  {'groupby': ['comments\_disabled'], 'target':'dislikes', 'agg':'mean\_ratio'},    {'groupby': ['comments\_disabled'], 'target':'comment\_count', 'agg':'mean\_diff'},  {'groupby': ['comments\_disabled'], 'target':'comment\_count', 'agg':'mean\_ratio'},    ]  df = agg(df,agg\_cols) |

1. データの読み込みと特徴量を作成するプログラム

テキストデータのベクトル化

morphological\_analysis.py

形態素解析を行うプログラムを図3 に示す．

形態素解析を行い，テキストデータから名詞，動詞，形容詞の抽出を行う．

|  |
| --- |
| import MeCab  # 分かち書きオブジェクト  tagger = MeCab.Tagger('')  tagger.parse('')  def morophological\_analysis(text):  node = tagger.parseToNode(text)  terms = []  select\_conditions = ['動詞', '形容詞', '名詞', '形容動詞']  while node:  # 単語  term = node.surface  # 品詞  pos = node.feature.split(',')[0]  # もし品詞が条件と一致してたら  if pos in select\_conditions:  terms.append(term)  node = node.next  # 連結  text\_result = ' '.join(terms)  return text\_result    for i in ["channelTitle", "title"]:  split\_words = []  for text in list(df[i]):  split\_words.append(morophological\_analysis(text))  df[i] = split\_words    df['new\_tags'] = df['tags'].astype(str).apply(lambda x: x.replace('|',' '))  df["description"] = df["description"].astype(str) |

1. 形態素解析を行うプログラム

tfidf\_svd.py

TF-IDFとLSAを行うプログラムを図4 に示す．tfidf\_vec.fit\_transformメソッドによってテキストデータを単語の重要度に変換する．TruncatedSVDメソッドによって単語の重要度があるテキストデータの次元圧縮を行う．

|  |
| --- |
| from sklearn.feature\_extraction.text import TfidfVectorizer  from sklearn.decomposition import TruncatedSVD  n\_comp = 100  for i in ['channelTitle','title']:#,'new\_title','new\_description',  print (i)  tfidf\_vec = TfidfVectorizer(analyzer='word',ngram\_range=(1,2))  text\_tfidf = tfidf\_vec.fit\_transform(df[i].values.tolist() )  text\_svd = TruncatedSVD(n\_components=n\_comp, algorithm='arpack',random\_state=9999)  df\_svd = pd.DataFrame(text\_svd.fit\_transform(text\_tfidf))  df\_svd.columns = ['svd\_'+str(i)+str(j+1) for j in range(n\_comp)]  df = pd.concat([df,df\_svd],axis=1)    for i in ['new\_tags',]:  print (i)  tfidf\_vec = TfidfVectorizer(analyzer='word',ngram\_range=(1,1))  text\_tfidf = tfidf\_vec.fit\_transform(df[i].values.tolist() )  text\_svd = TruncatedSVD(n\_components=n\_comp, algorithm='arpack',random\_state=9999)  df\_svd = pd.DataFrame(text\_svd.fit\_transform(text\_tfidf))  df\_svd.columns = ['svd\_char\_'+str(i)+str(j+1) for j in range(n\_comp)]  df = pd.concat([df,df\_svd],axis=1) |

1. TF-IDFとLSAを行うプログラム

LightGBMによる再生数を予測するモデルの構築

predict\_lightgbm.py

　LightGBMによる再生数を予測するモデルの構築を行うプログラムを図5 に示す．

|  |
| --- |
| import lightgbm as lgb  import datetime  from sklearn.preprocessing import StandardScaler  from sklearn.metrics import mean\_squared\_error  def rmse(y\_true, y\_pred):  return (mean\_squared\_error(y\_true, y\_pred))\*\* .5  def target\_encoder\_kfold(train\_df,test\_df,col,target,folds,method):  oof\_preds = np.zeros(train\_df.shape[0])  sub\_preds = np.zeros(test\_df.shape[0])  for n\_fold, (train\_idx, valid\_idx) in enumerate(folds.split(train\_df, train\_df['y\_bin'])):  print ('FOLD:' + str(n\_fold))  train\_x = train\_df.iloc[train\_idx]  valid\_x = train\_df.iloc[valid\_idx]  if method == 'mean':  oof\_preds[valid\_idx] = valid\_x[col].map(train\_x.groupby(col)[target].apply(lambda x: stats.trim\_mean(x, 0.01)))  if method == 'median':  oof\_preds[valid\_idx] = valid\_x[col].map(train\_x.groupby(col)[target].median())  if method == 'max':  oof\_preds[valid\_idx] = valid\_x[col].map(train\_x.groupby(col)[target].max())  if method == 'min':  oof\_preds[valid\_idx] = valid\_x[col].map(train\_x.groupby(col)[target].min())  if method == 'mean':  sub\_preds = test\_df[col].map(train\_df.groupby(col)[target].apply(lambda x: stats.trim\_mean(x, 0.01)))  if method == 'median':  sub\_preds = test\_df[col].map(train\_df.groupby(col)[target].median())  if method == 'max':  sub\_preds = test\_df[col].map(train\_df.groupby(col)[target].max())  if method == 'min':  sub\_preds = test\_df[col].map(train\_df.groupby(col)[target].min())  return oof\_preds,sub\_preds  def lgb\_k\_fold(train\_df, test\_df, features, target, cat\_features, folds, params, use\_pseudo=False, sampling=False):  oof\_preds = np.zeros(train\_df.shape[0])  sub\_preds = np.zeros(test\_df.shape[0])  cv\_list = []  for n\_fold, (train\_idx, valid\_idx) in enumerate(folds.split(train\_df[features], train\_df['y\_bin'])):  print ('FOLD:' + str(n\_fold))  print("train idx is {} and valid idx is {}".format(train\_idx, valid\_idx))  print("train idx length is {} and valid idx length is {}".format(len(train\_idx), len(valid\_idx)))    train\_x, train\_y = train\_df[features].iloc[train\_idx], train\_df[target].iloc[train\_idx]  valid\_x, valid\_y = train\_df[features].iloc[valid\_idx], train\_df[target].iloc[valid\_idx]    # remove outliers  if sampling is True:  valid\_x, valid\_y = train\_df[features].iloc[valid\_idx], train\_df[target].iloc[valid\_idx]  train\_df\_new = train\_df.iloc[train\_idx]  train\_df\_new = train\_df\_new[train\_df\_new['y\_residual']<3.2]#3  train\_x, train\_y = train\_df\_new[features], train\_df\_new[target]    # pseudo  if use\_pseudo is True:  train\_x = pd.concat([train\_x, pseudo[features]], axis=0)  train\_y = train\_y.append(pseudo[target])  for n, (pseudo\_train\_idx, pseudo\_valid\_idx) in enumerate(folds.split(pseudo[features], pseudo['y\_bin'])):  print ('PSEUDO FOLD:' + str(n))  if n\_fold == n:  train\_x = pd.concat([train\_x, pseudo[features].iloc[pseudo\_valid\_idx]],axis=0)  train\_y = train\_y.append(pseudo[target].iloc[pseudo\_valid\_idx] )  break    print ('train\_x shape:',train\_x.shape,train\_y.mean())  print ('valid\_x shape:',valid\_x.shape,valid\_y.mean())    dtrain = lgb.Dataset(train\_x, label=train\_y,categorical\_feature=cat\_features)  dval = lgb.Dataset(valid\_x, label=valid\_y, reference=dtrain,categorical\_feature=cat\_features)  bst = lgb.train(params, dtrain, num\_boost\_round=50000,  valid\_sets=[dval,dtrain], verbose\_eval=500,early\_stopping\_rounds=500, )  # output feature importance  feature\_importances = sorted(zip(features, bst.feature\_importance('gain')),key=lambda x: x[1], reverse=True)  sum\_fi = []  for fi in feature\_importances:  sum\_fi.append(fi[1])    sum\_fi = sum(sum\_fi)    for fi in feature\_importances[:30]:  print("効果的な特徴量", fi)  print(fi[1]/sum\_fi)    # 訓練データを利用  # 交差検証で得られたvalidのインデックスの場所に予測値をいれる  oof\_preds[valid\_idx] = bst.predict(valid\_x, num\_iteration=bst.best\_iteration)  # RMSE  oof\_cv = rmse(valid\_y, oof\_preds[valid\_idx])  cv\_list.append(oof\_cv)  print (cv\_list)  print(oof\_preds)    # テストデータに対する予測  sub\_preds += bst.predict(test\_df[features], num\_iteration=bst.best\_iteration) / folds.n\_splits  cv = rmse(train\_df[target], oof\_preds)  print('Full OOF RMSE %.6f' % cv)  train\_df['lgb\_y'] = oof\_preds  test\_df['lgb\_y'] = sub\_preds  return train\_df,test\_df,cv  params = {  "objective" : "regression",  "boosting" : "gbdt",  "metric" : "rmse",  "max\_depth": -1,  "min\_data\_in\_leaf": 10, #10  "min\_gain\_to\_split": 0.01,#0.01  "min\_child\_weight": 0.001,  "reg\_alpha": 0.1,  "reg\_lambda": 1, #1  "num\_leaves" : 35, #40  "max\_bin" : 300,#300  "learning\_rate" :0.01,  "bagging\_fraction" : 0.9,  "bagging\_freq" : 1,  "bagging\_seed" : 4590,  "feature\_fraction" : 0.6,#0.6  "verbosity": -1,  "boost\_from\_average": False,  }  # target encoder  train\_df['categoryId\_target\_mean'],test\_df['categoryId\_target\_mean'] = target\_encoder\_kfold(train\_df,test\_df,'categoryId',target,folds,'mean')  train\_df['ratings\_disabled\_target\_mean'],test\_df['ratings\_disabled\_target\_mean'] = target\_encoder\_kfold(train\_df,test\_df,'ratings\_disabled',target,folds,'mean')  train\_df['comments\_disabled\_target\_mean'],test\_df['comments\_disabled\_target\_mean'] = target\_encoder\_kfold(train\_df,test\_df,'comments\_disabled',target,folds,'mean')  train\_df['comments\_ratings\_disabled\_target\_mean'],test\_df['comments\_ratings\_disabled\_target\_mean'] = target\_encoder\_kfold(train\_df,test\_df,'comments\_ratings\_disabled',target,folds,'mean')  cat\_features = [  'categoryId',  'channelTitle\_encoder',  'comments\_ratings\_encoder',  'c\_month',  'year',  ]  print ('numerical features:', len(features),features)#  train\_lgb,test\_lgb,cv = lgb\_k\_fold(train\_df,test\_df,features,target,cat\_features,folds,params,use\_pseudo=False,sampling=False)  out\_dir = "out\_tmp"  !mkdir -p $out\_dir  day\_dir = str(datetime.date.today())  !mkidr -p $day\_dir  # submission  train\_lgb[['id','lgb\_y','y']].to\_csv(f'./{out\_dir}/{day\_dir}/train\_lgb.csv',index=False)  test\_lgb[['id','lgb\_y']].to\_csv(f'./{out\_dir}/{day\_dir}/test\_lgb.csv',index=False) |

1. LightGBMによる再生数予測モデルを構築するプログラム