TOXIC COMMENT CLASSIFICATION CHALLENGE

1

PRIVATE SCORE : 0.98705

総合順位：67位/4550チーム(※)

※ 期間が終わった後の参加のため、あくまで参考

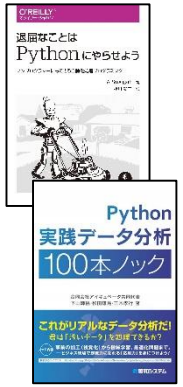
2020/11/11 中向

1. 背景

勉強内容の実践のために、Kaggleの言語コンペに挑戦してみました。
期間が終わっているコンペなのであくまで参考ですが、 67位/4550チームまで精度を上げることができたので、内容共有します。

勉強内容

2019/10 業務効率化のためPython勉強開始



2020/2 機械学習の勉強開始

2020/8 Deep Learningの勉強開始

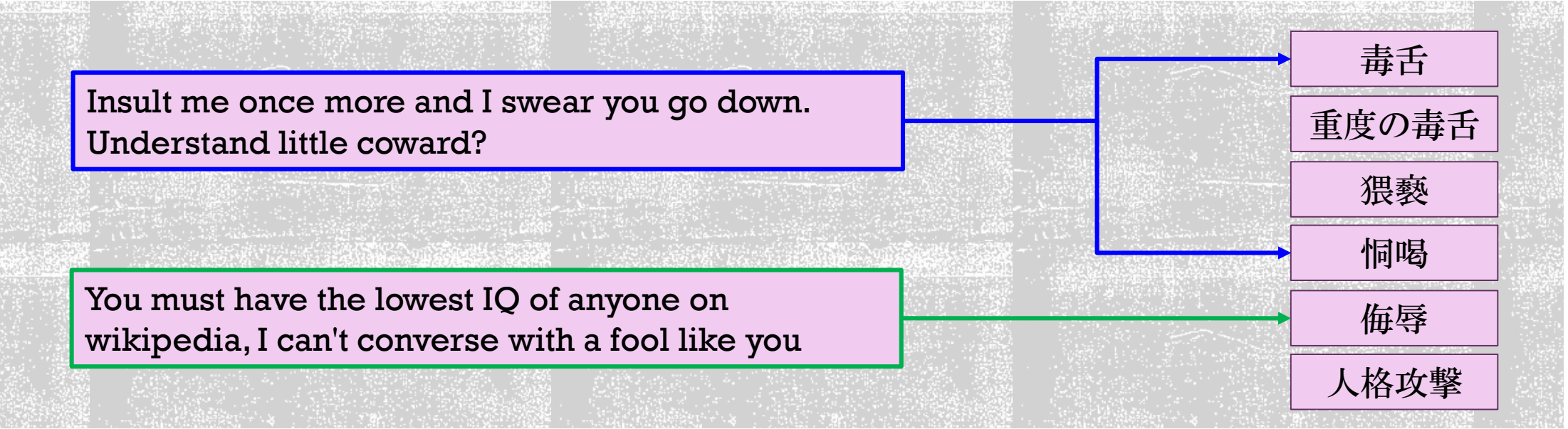
2020/9 実践のため、Kaggleの画像コンペに挑戦

2020/10 実践のため、Kaggleの言語コンペに挑戦

kaggle 統計家、データ分析家が分析手法を競い合う場所

2. テーマ

英語の文章を6種類の毒舌を発見してくれるAIを作ります。



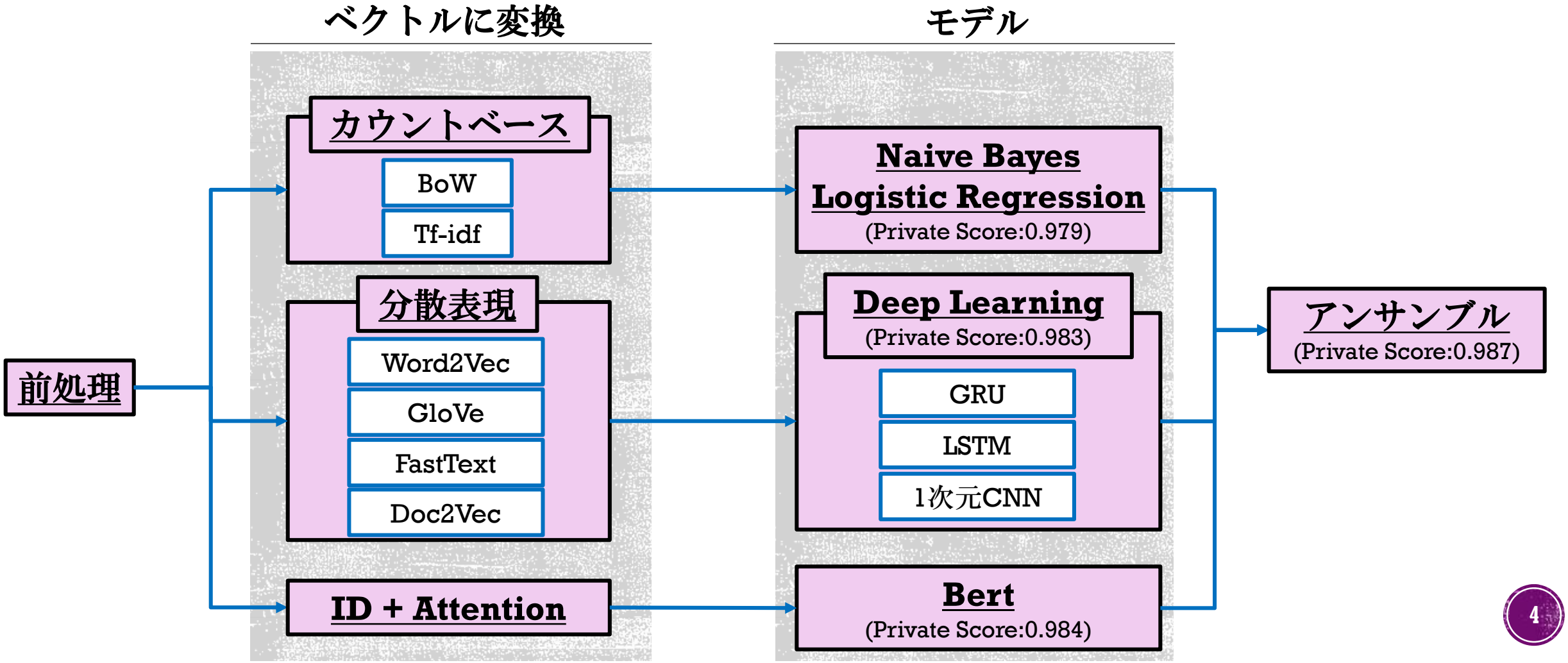
Train Data
159,571文章
(ラベル付き)

Test Dataを予測した時の
AUC(Area Under the Curve)で競う

Test Data
153,164文章
(ラベル無し)

3.全体像

やった内容の全体像はこちらになります。



4.前処理

色々な前処理をしています。本当は再翻訳を使ったデータ拡張まで行きたかったですが、そこまでは行けませんでした。

記号前後に空白を追加
!?以外の記号除去

You know?	you know ?
D'aww!	d aww !

単語の中に一つだけ大文字がある場合は、小文字に統一

He is selfish	he is selfish
HE IS SELFISH	HE IS SELFISH

表記の揺れの統一

:-)	good
:-(	bad
didn't	did not
asap	as soon as possible

データ拡張（時間がなく間に合わず）

英語：What's taking so long?

翻訳

日本語：何がそんなに時間がかかっているのですか？

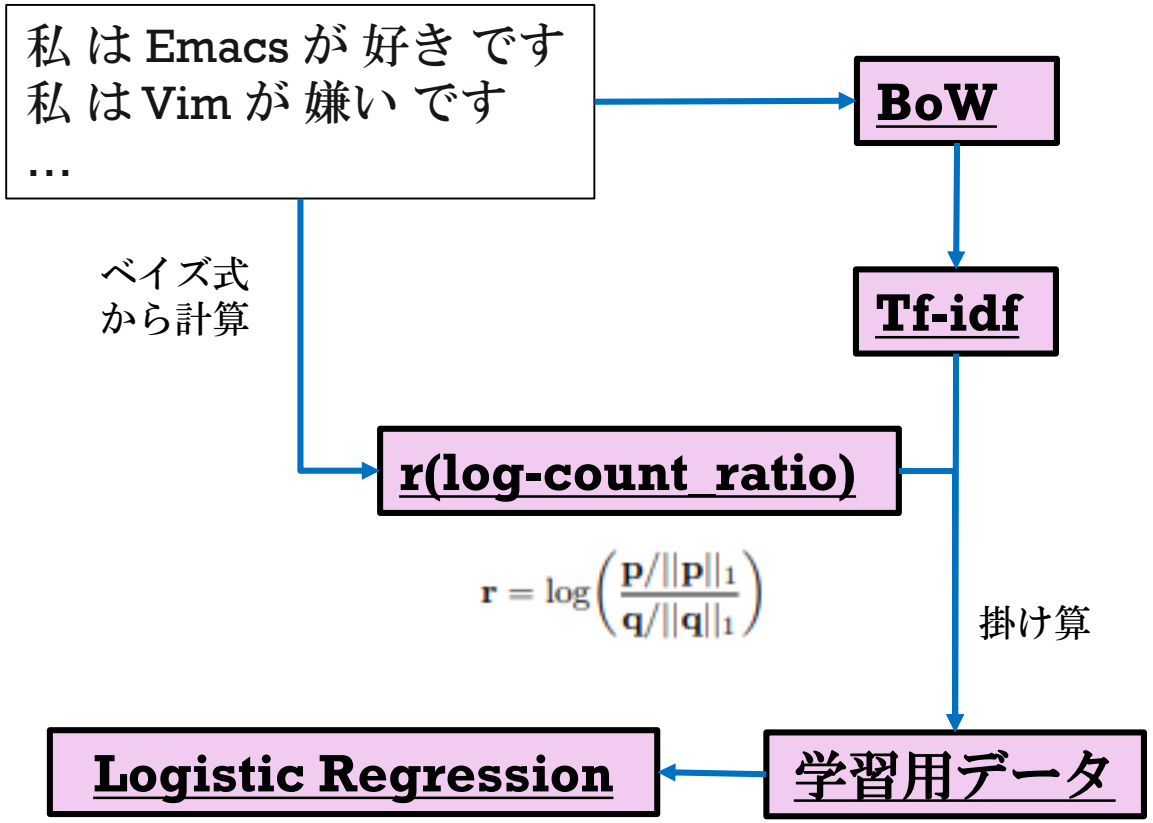
再翻訳

英語：What is taking so long?

5. NAIVE BAYES

Private Score ： 0.976

まずはNaïve Bayesを使った基本的な手法を試しましたが、この時点で Score:0.976 とかなり良いです。



BoW：単語をカウントしただけ

私	は	が	好き	嫌い	です	毒舌
1	1	1	1	0	1	0
1	1	1	0	1	1	1
…						…

Tf-idf : 単語の重要度を考慮（珍しい単語の重要度が上がる）

私	は	が	好き	嫌い	です	毒舌
0.01	0.01	0.01	**0.5**	0	0.01	0
0.01	0.01	0.01	0	**0.5**	0.01	1
…						…

r : 毒舌の文章への出てきやすさを表す

私	は	が	好き	嫌い	です
0.01	0.01	0.01	0.001	**0.5**	0.01

Tf-idf と r を掛け算することで、学習データを作成する

私	は	が	好き	嫌い	です	毒舌
0.0001	0.0001	0.0001	0.005	0	0.0001	0
0.0001	0.0001	0.0001	0	0.25	0.0001	1

5. NAIVE BAYES

Private Score ： 0.976

Private Score ： 0.979

グリッドサーチすることで、わずかにスコアが向上しました。

公開されているコード

参考：https://www.kaggle.com/jhoward/nb-svm-strong-linear-baseline

Tf-idf

r(log-count ratio)

学習用データ

Logistic Regression
(C=4)
Private Score:0.976

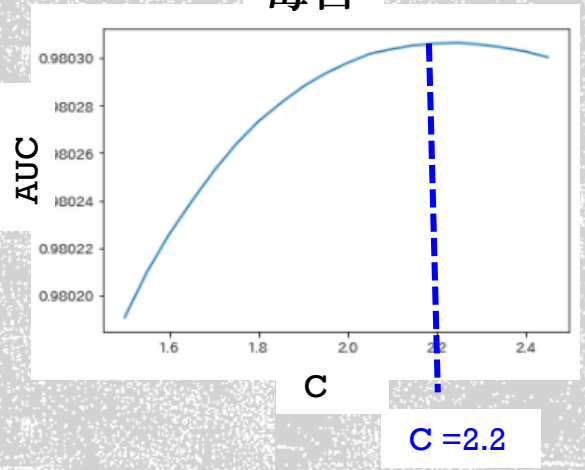
①前処理を追加
②AUCを指標とした
グリッドサーチを追加

Private Score:0.979

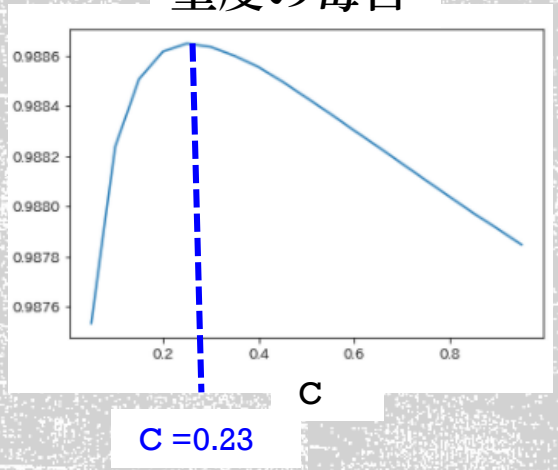
グリッドサーチ結果

6個のモデルを作成し、それぞれでグリッドサーチ

毒舌



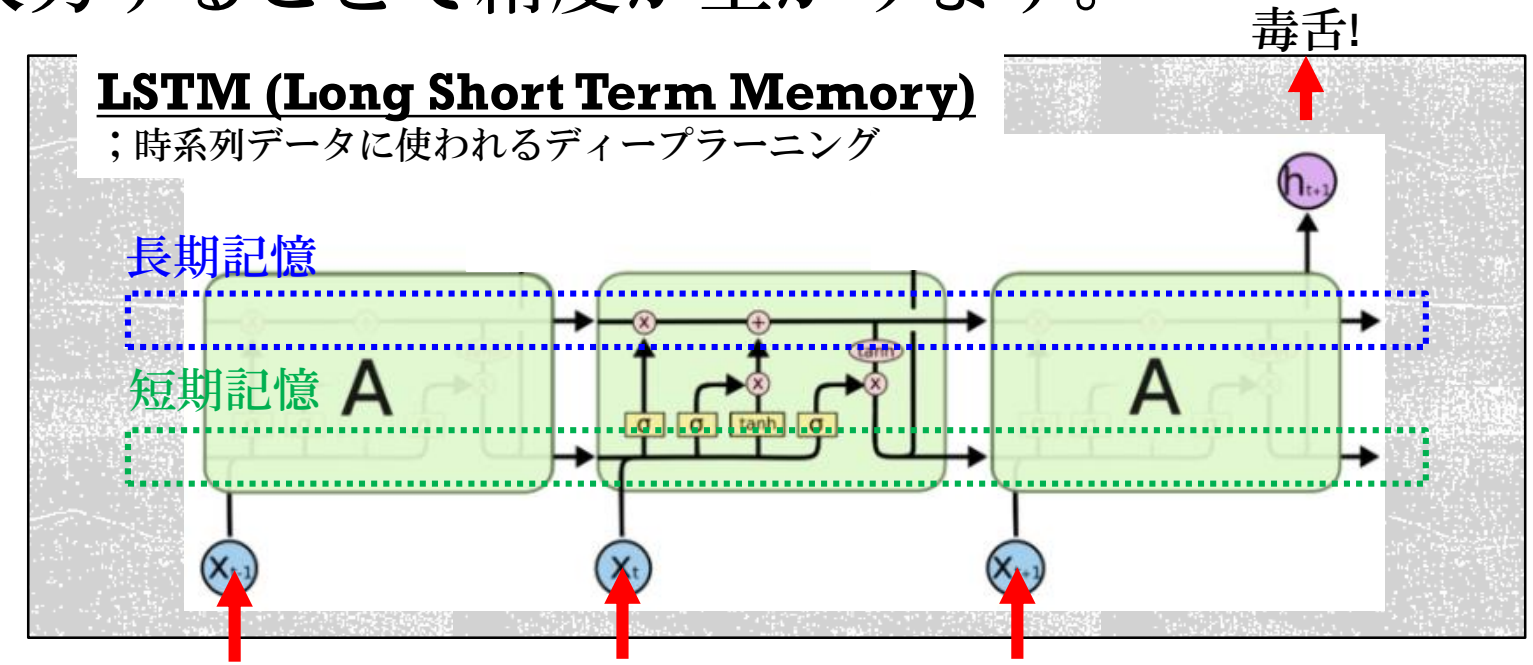
重度の毒舌




```
#logistic回帰をインスタンス化
logreg=LogisticRegression(dual=True, penalty='l2',solver='liblinear')
#GridSearchCVをインスタンス化
logreg_cv=GridSearchCV(logreg,grid,cv=3,scoring="roc_auc",verbose=10)
```

6.分散表現 + DEEP LEARNING

LSTMも試してみました。文章を分散表現と呼ばれるベクトルに変換して入力することで精度が上がります。



分散表現

単語の意味を考慮したベクトル。

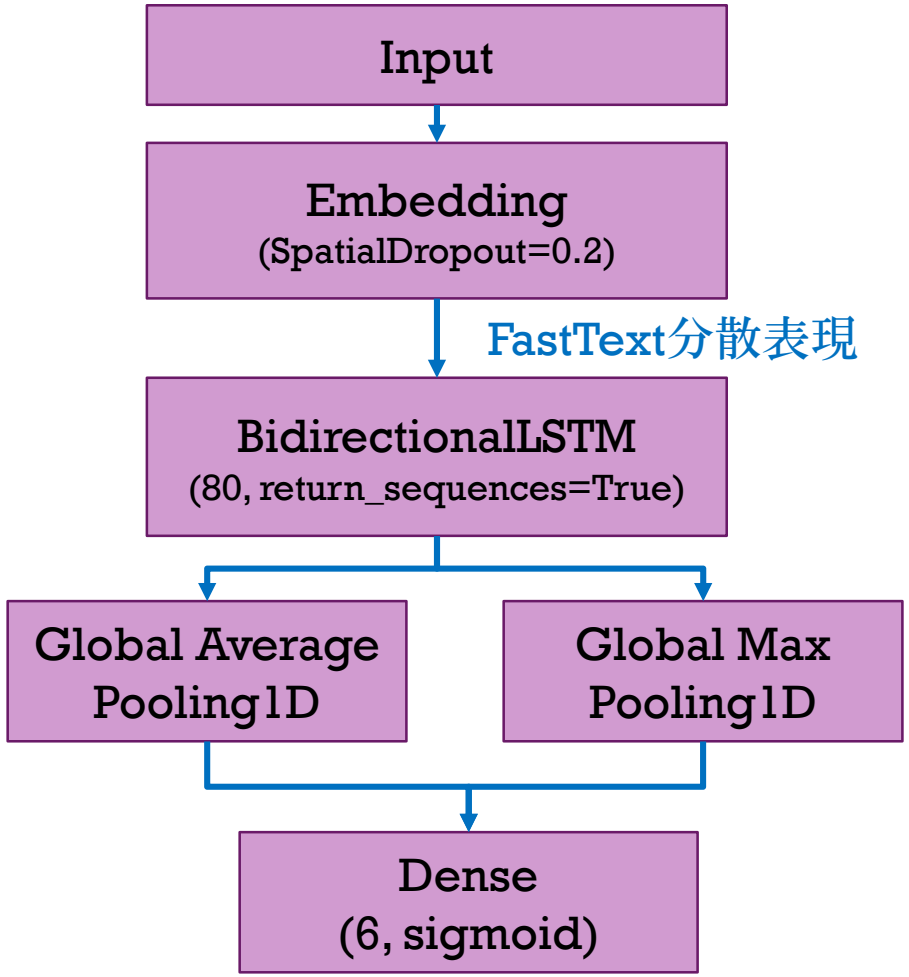
パリ－フランス＋日本＝東京

(1 ,0 ,0 ,…)	(0 ,1 ,0 ,…)	(0 ,0 ,1 ,…)	△ One-Hotベクトルを入力
(0.2, 0.3, 0.1,…)	(0.5, 0.5, 0.8, …)	(0.1, 0.1, 0.6, …)	◎ 分散表現を入力

ベクトル同士の関係に意味がない

6.分散表現 + DEEP LEARNING(実装)

分散表現を入力するためには、LSTMの前にEmbedding層を作ります。



Embedding層の作り方

参考：https://www.kaggle.com/yekenot/pooled-gru-fasttext

①crawl-300d-2M.vec　をダウンロード

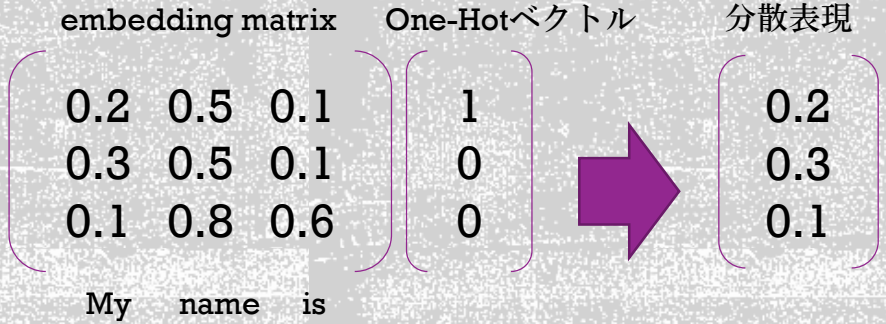
Wikipedia 学習済み FastTextモデルによる分散表現

②embedding matrixを作成

①のファイルの分散表現を並べて行列を作る

③embedding layerの重みに②を渡す

```
def get_model():
    inp = Input(shape=(maxlen, ))
    x = Embedding(max_features, embed_size, weights=[embedding_matrix],trainable=True)(inp)
```

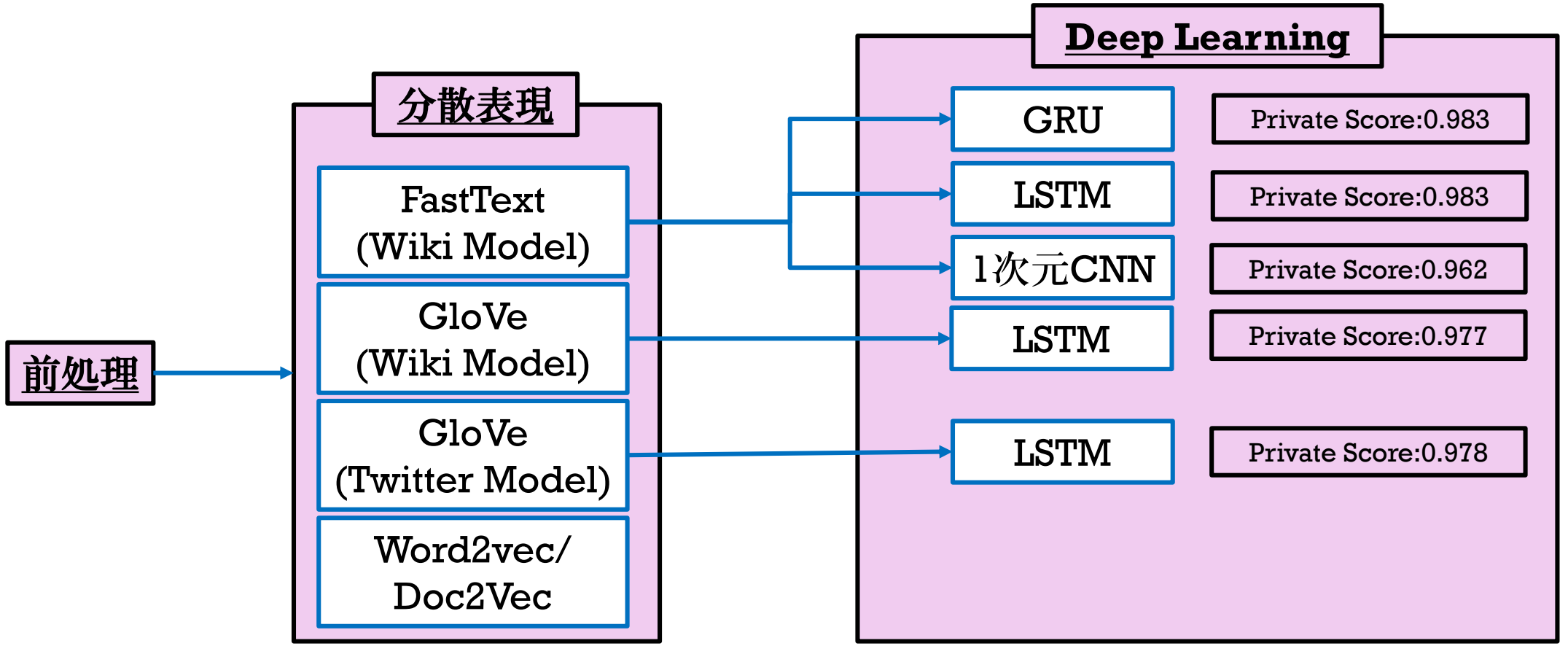


6.分散表現 + DEEP LEARNING

Private Score ： 0.979

Private Score ： 0.983

分散表現とDeepLearningの組み合わせを変えてみました。一番良いものでスコアが0.983になりました。



6.DOC2VECによる文章の可視化(おまけ)

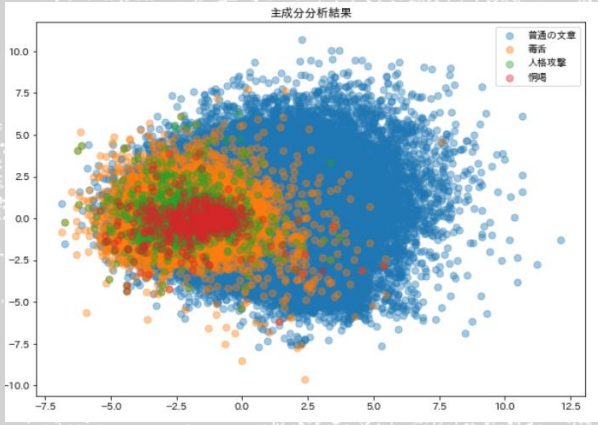
スコアには関係ないですが、分散表現と主成分分析を組み合わせることで文章を可視化することができることもわかりました。

文章データ → **Doc2Vec(教師無し学習)** → 100次元ベクトル → **主成分分析(教師無し学習)** → 2次元ベクトル → **可視化**

```
from gensim.models.doc2vec import Doc2Vec
from gensim.models.doc2vec import TaggedDocument

#Doc2vecのtrainデータを格納
trainDoc = []

y=0
for i in tqdm(train['comment_text']):
    #タグはindexとする
    tag = y
    #分かち書きをリスト化
    pl = i.split(' ')
    #タグをリスト化
    tl=[tag]

#     print(pl)
#     print(tag)
    tagDoc = TaggedDocument(words=pl, tags=tl)
    trainDoc.append(tagDoc)
    y=y+1

model = Doc2Vec(documents=trainDoc, epochs=100, min_count=1)
```

```
#Doc2vecをデータ全体に適用
import numpy as np

doc2_list=[]
for i in range(len(train)):
    doc2_list.append(model.docvecs[i])

#結果をデータフレーム化(100次元のベクトルに変換される)
df_doc2vec=pd.DataFrame(np.array(doc2_list))
```

```
#PCA
from sklearn.decomposition import PCA
pca=PCA(n_components=2)
pca_result=pca.fit_transform(df_doc2vec)
```



7. BERT

Private Score ： 0.983

Private Score ： 0.984

Bertも試してみたところ、一番精度が良かったです。

BERT；転移学習みたいなことができる言語モデル

1 epoch 回すだけで、かなり高精度

```
6822s 2s/step - loss: 0.0453 - acc: 0.8905 - val_loss: 0.0375 - val_acc: 0.9907
ROC-AUC - epoch: 1 - score: 0.989757
```

グーグルが挑む最先端AI

グーグルの最新AI「BERT」のインパクト、ついに読解力も人間超え

中田 敦＝日経 xTECH／日経コンピュータ　2020/03/24　日経XTECH ACTIVE

出典：日経コンピュータ、2019年12月12日号pp.30-33
（記事は執筆時の情報に基づいており、現在では異なる場合があります）　記事一覧

モデル

```
Layer (type)                 Output Shape              Param #     Connected to
==================================================================================================
input_4 (InputLayer)         [(None, 128)]             0
__________________________________________________________________________________________________
input_5 (InputLayer)         [(None, 128)]             0
__________________________________________________________________________________________________
input_6 (InputLayer)         [(None, 128)]             0
__________________________________________________________________________________________________
tf_bert_model_1 (TFBertModel)  ((None, 128, 768), ( 109482240   input_4[0][0]
                                                                 input_5[0][0]
                                                                 input_6[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)              (None, 6)                 4614        tf_bert_model_1[0][1]
==================================================================================================
Total params: 109,486,854
Trainable params: 109,486,854
Non-trainable params: 0
```

Inputには3種類渡す
（詳細は次ページ）

```
import transformers
bert_model = transformers.TFBertModel.from_pretrained(model_name)
```

参考：https://www.kaggle.com/nkaenzig/bert-tensorflow-2-huggingface-transformers
上記を読み解くための参考：https://www.tensorflow.org/tutorials/quickstart/advanced?hl=ja

7. BERT(学習データの作り方)

Bertに渡す学習データはこんな感じで作りました。

```
text='''Explanation¥nWhy the edits made under my username Hardcore Metallica Fan were reverted?
```

IDを付与 (tokenizer.encode)

```
[101, 7526, 2339, 1996, 10086, 2015, 2081, 2104, 2026, 5310, 18442, 13076, 12392, 2050, 5470, 2020, 16407, 1029,
```

```
101⇒[CLS]
7526⇒explanation
2339⇒why
1996⇒the
10086⇒edit
2015⇒##s
2081⇒made
```

0でpadding (pad_sequences)

Input_ids

```
[   101  7526  2339  1996 10086  2015  2081  2104  2026  5310 18442 13076
  12392  2050  5470  2020 16407  1029  2027  4694  1005  1056  3158  9305
  22556  1010  2074  8503  2006  2070  3806  2044  1045  5444  2012  2047
   2259 14421  6904  2278  1012  1998  3531  2123  1005  1056  6366  1996
  23561  2013  1996  2831  3931  2144  1045  1005  1049  3394  2085  1012
   6486  1012 16327  1012  4229  1012  1016   102     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0     0     0     0     0
      0     0     0     0     0     0     0     0]
```

attention_masks

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

0じゃない箇所を全部1にする

token_type_ids

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0.]
```

ファインチューニングの場合は
すべて0にする

8.纏め

Private Score ： 0.984

Private Score ： 0.987

最後にこれまでやったことをアンサンブルすることで精度が上がりました。