機械学習を用いた特許文書分類

～文書ベクトルを使用した分類器の実装例～

2018年11月30日

アジア特許情報研究会
（ユポ・コーポレーション）
西尾　潤

AI（人工知能）×特許情報

- Fターム付与
- 変動予測
- 予防・治療
- 自動運転
- 異常検知
- クラスタリング
- Q&A
- 空間認識
- ロボティックス
- バイオデータ
- センサデータ
- IoT
- テキスト分類
- ビッグデータ
- 特許データ
- パテントマップ
- 自然言語処理
- テキスト
- 市場データ
- 文書類似度
- ?
- IP landscape 2.0
- 経営データ
- 画像
- 発明の内容理解・認定
- 音声生成
- 人工発話
- テキスト生成
- 画像生成
- スマートスピーカー
- 先行技術調査
- 画像類似度
- 電話応答システム
- 特許価値評価
- 人工知能による創作（著作）
- 機械翻訳
- 登録可否判断
- 人工知能による発明（考案）
- 商標調査
- 意匠調査
- 明細書作成

解析モデル

生成モデル

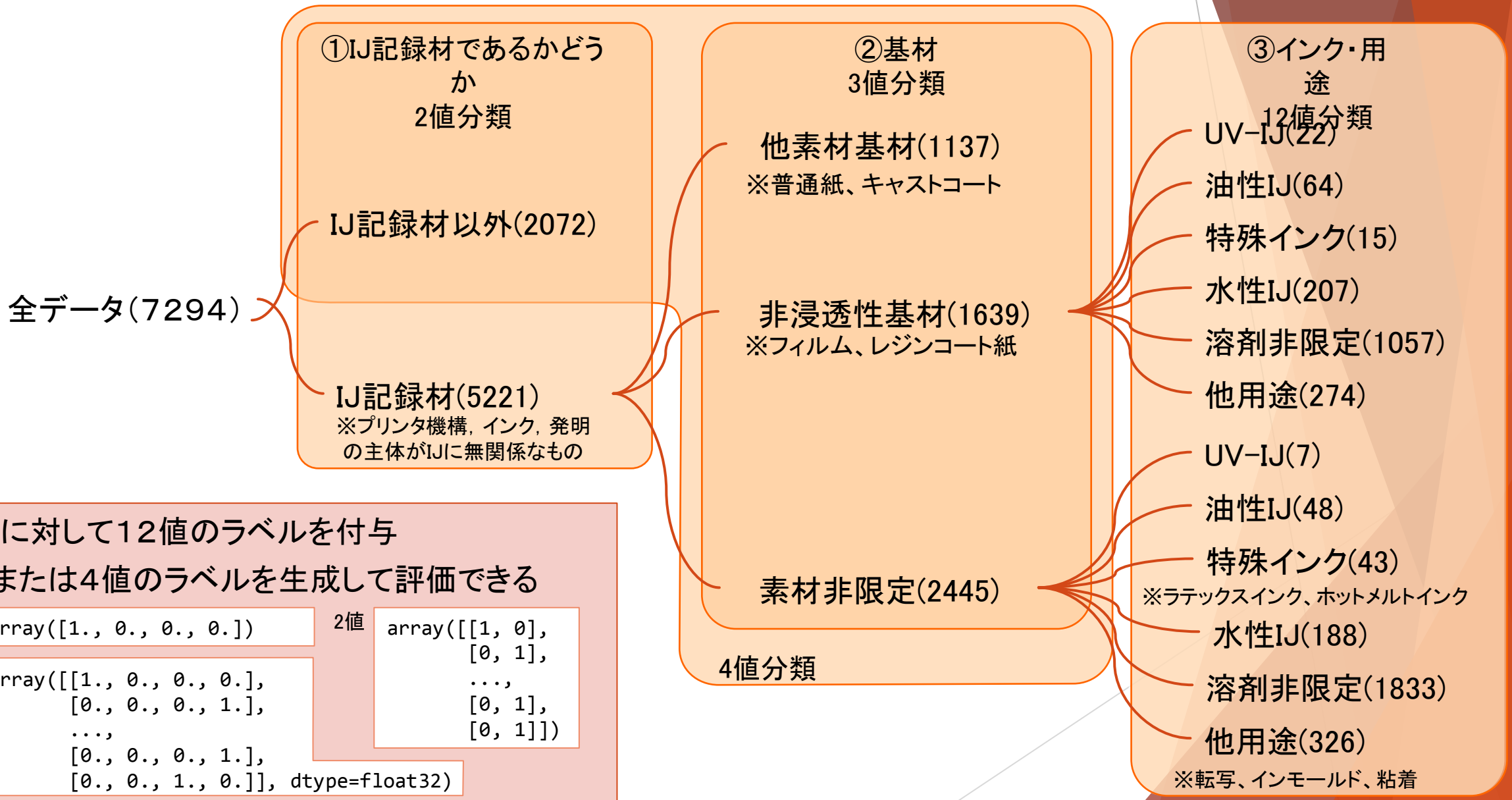
目的

無料のツール（＝python）の利用
GPUを持たない汎用PC
計算コストを抑えるモデル
教師データは査読によって与えられた分類 ― 2値分類 / 多値分類
可能な限り査読数を抑える

実行環境

PC：Core i7-6700, RAM 8MB, GPU なし
OS：Windows 10 64bit
テキスト処理：Text Mining Studio 6.1.2
機械学習：Python 3.6.6(Anaconda), IPython 6.5.0(Spyder 3.3.1)
特許データ：shareresearch, 特許請求の範囲（インクジェット）
データ数：7294件（教師1097件, 検証6197件, 教師データ率15%）

データ分類(ラベル付与)基準



全件に対して12値のラベルを付与

2値または4値のラベルを生成して評価できる

2値
```
array([1., 0., 0., 0.])
```

4値
```
array([[1., 0., 0., 0.],
       [0., 0., 0., 1.],
       ...,
       [0., 0., 0., 1.],
       [0., 0., 1., 0.]], dtype=float32)
```

2値
```
array([[1, 0],
       [0, 1],
       ...,
       [0, 1],
       [0, 1]])
```

テキスト前処理

Text Minig Studio で分かち書き＋再結合　連結語化

分かち書きのみ一形態素

ファイルID	行ID	文章ID	単語ID	見出し語	原形	置換語	品詞	品詞詳細	係り先
1	1	1	1	【	【	【	記号	括弧開	-1
1	1	1	2	特許	特許	特許	名詞	一般	-1
1	1	1	3	請求	請求	請求	名詞	サ変可能	-1
1	1	1	4	の	の	の	助詞	格助詞	-1
1	1	1	5	範囲	範囲	範囲	名詞	一般	-1
1	1	1	6	】	】	】	記号	括弧閉	-1
1	1	1	7				空白		-1
1	1	1	8	【	【	【	記号	括弧開	-1
1	1	1	9	請求	請求	請求	名詞	サ変可能	-1
1	1	1	10	項	項	項	名詞	一般	-1
1	1	1	11	1	1	1	名詞	数詞	-1
1	1	1	12	】	】	】	記号	括弧閉	-1

分かち書きと自動連結と係り受け

ファイルID	行ID	文章ID	単語ID	見出し語	原形	置換語	品詞	品詞詳細	係り先
1	1	1	1	【特許請求の	特許請求	特許請求	名詞	一般	2
1	1	1	2	範囲】	範囲	範囲	名詞	一般	3
1	1	1	3	【請求項1】	請求項1	請求項1	名詞	一般	4
1	1	1	4	水、	水	水	名詞	一般	6
1	1	1	5	カチオン性の	カチオン性	カチオン性	名詞	一般	6
1	1	1	6	水分散性樹脂、	水分散性樹脂	水分散性樹脂	名詞	一般	8
1	1	1	7	及び	及び	及び	接続詞		8
1	1	1	8	表面張力調整剤を	表面張力調整剤	表面張力調整剤	名詞	一般	9
1	1	1	9	含み、	含む	含む	動詞	一般	20
1	1	1	10	前記	前記	前記	名詞	サ変可能	11
1	1	1	11	カチオン性の	カチオン性	カチオン性	名詞	一般	12
1	1	1	12	水分散性樹脂は、	水分散性樹脂	水分散性樹脂	名詞	一般	20

係り先IDと単語IDから係り受け表現を求める

1	【特許請求の	→ 2	範囲】
2	範囲】	→ 3	【請求項1】
3	【請求項1】	→ 4	水、
4	水、	→ 6	水分散性樹脂、
5	カチオン性の	→ 6	水分散性樹脂、
6	水分散性樹脂、	→ 8	表面張力調整剤を
7	及び	→ 8	表面張力調整剤を
8	表面張力調整剤を	→ 9	含み、
9	含み、	→ 20	成型板用表面処理液。
10	前記	→ 11	カチオン性の
11	カチオン性の	→ 12	水分散性樹脂は、
12	水分散性樹脂は、	→ 20	成型板用表面処理液。
13	動的光散乱法により	→ 14	測定される
14	測定される	→ 15	メジアン径が
15	メジアン径が	→ 18	含む、
16	１μｍ以上１０μｍ以下の	→ 17	大粒子を
17	大粒子を	→ 18	含む、
18	含む、	→ 20	成型板用表面処理液。

テキスト数値化

学習器に投入するデータは数字として与える

・浮動小数点型のリストを作成

・重複する連結語を整理してユニークID辞書を作成

・存在しない連結語を指定されるとき(エラー回避)のために 0 を割り振る

・辞書に連結語を投入して得られるIDをリストにまとめる　以後「単語」と表現

	B	C	D	E
	行ID	文章ID	単語ID	見出し語
	1	1	1	【特許請求
	1	1	2	範囲】
	1	1	3	【請求項1】

	A	B	C	D	E	F	G	H	I	J	K
1	83591	128709	158465	81504	146317	28088	175802	152088	134703	24196	1463
2	83591	128709	158465	42225	35511	175802	146972	118533	6260	83156	2419
3	83591	128709	158465	127362	134888	49017	90530	125305	49441	29580	5186
4	83591	128709	158465	114562	173776	165148	160314	144007	30348	48522	6644
5	83591	128709	9129	105504	105023	83156	830	7114	142426	103829	8315
6	83591	128709	158465	157950	105023	105504	103829	83156	830	7114	1424
7	83591	128709	158465	35511	31760	175802	92592	117411	118533	922	8315
8	83591	128709	9129	87382	140458	97852	134269	62383	3451	81599	7076
9	35710	83591	128709	158465	35511	31760	175802	92592	117411	118533	922

行：公報数（7294）
列：各公報に含まれる単語数の最大値（4634）
※7294 × 4634 次元の行列

最小値：1（コンプライアンス部の）
最大値：179,200（１質量部）
※値同士は何の関連性もない

```python
import pandas as pd
...
df = pd.read_table(file, delimiter=',')
dict_data = list(set(df['見出し語']))
Dictionary = []
for i in range(len(dict_data)):
    # len(dict_data) = 179200
    Dictionary.append((dict_data[i], i + 1))
dictionary = dict(Dictionary)
dictionary[''] = 0

text_data = []
text_line = []
for i in range(len(df['行ID'])):
    # len(df['行ID']) = 1156219, max(df['行ID']) = 7294
    text_line.append(dictionary[df['見出し語'][i]])
    if df['行ID'][i] != df['行ID'][i + 1]:
        text_data.append(text_line)
        text_line = []
```

数値化テキストの特徴

公報中単語数は偏りが大

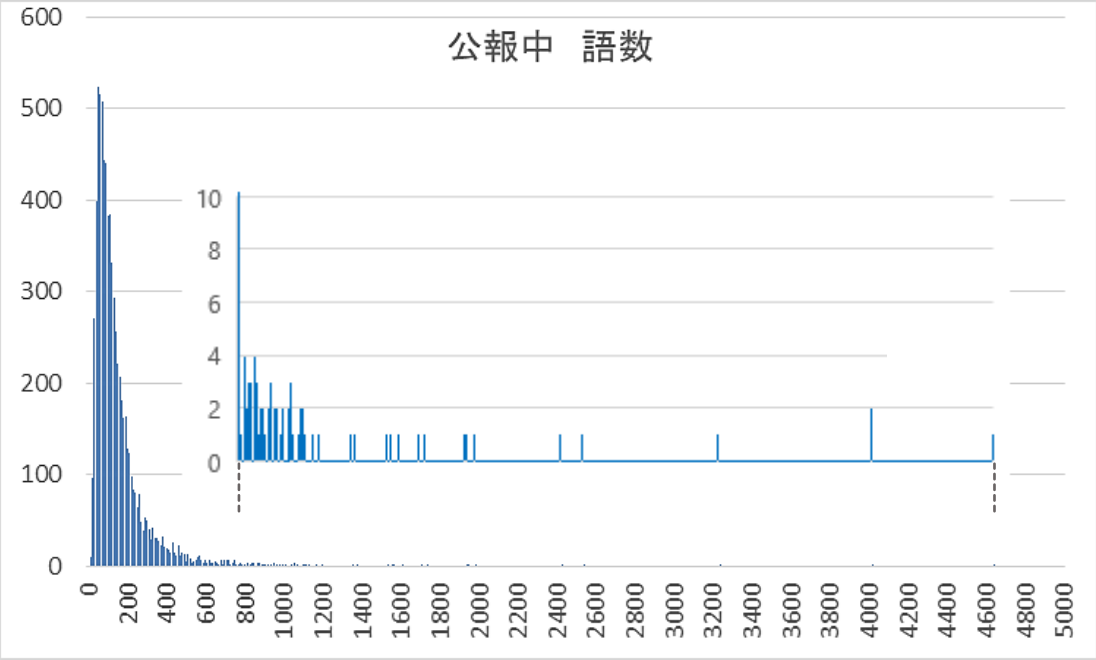
・単語数600より大きい公報はわずか

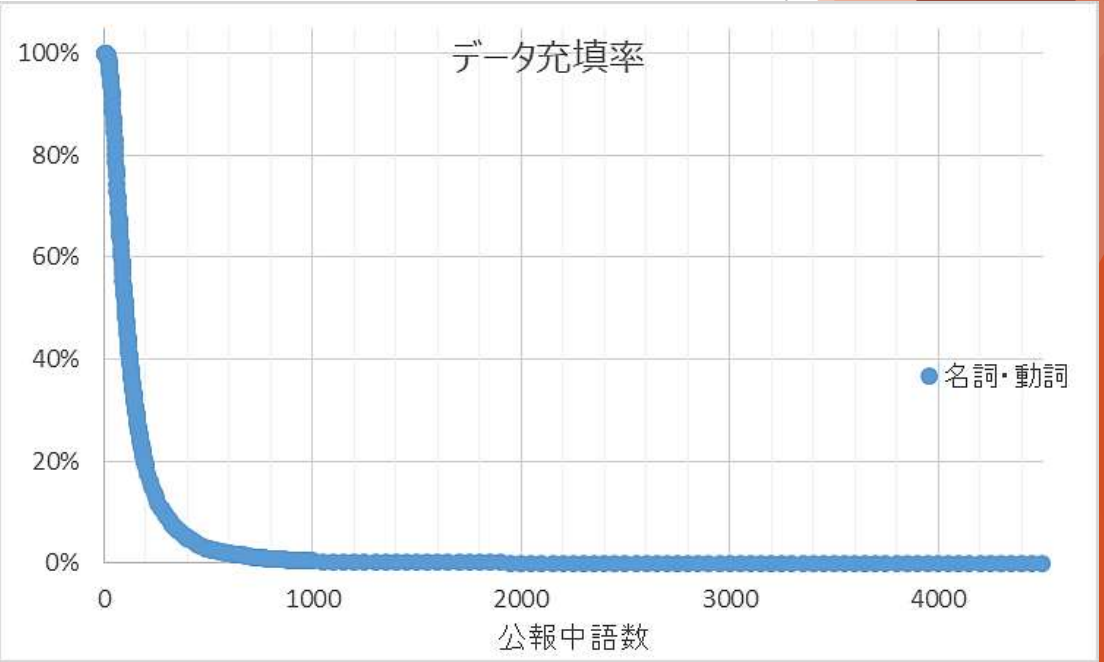
・特定の公報のみ請求項数が多い

・単語数を限定し、後方の単語をカットするとデータ充填率が上がる

データ充填率
= データを有する行列要素 / 全行列要素数

	FO	FP	FQ	FR	FS	FT	FU	FV	FW	FX
0	228	15	2899	29218	28693	202	43	130	401	5
6	44	244	118	4	504	3	2	7577	8329	40066
9	22	549	3528	1026	534	7	29	1708	277	285



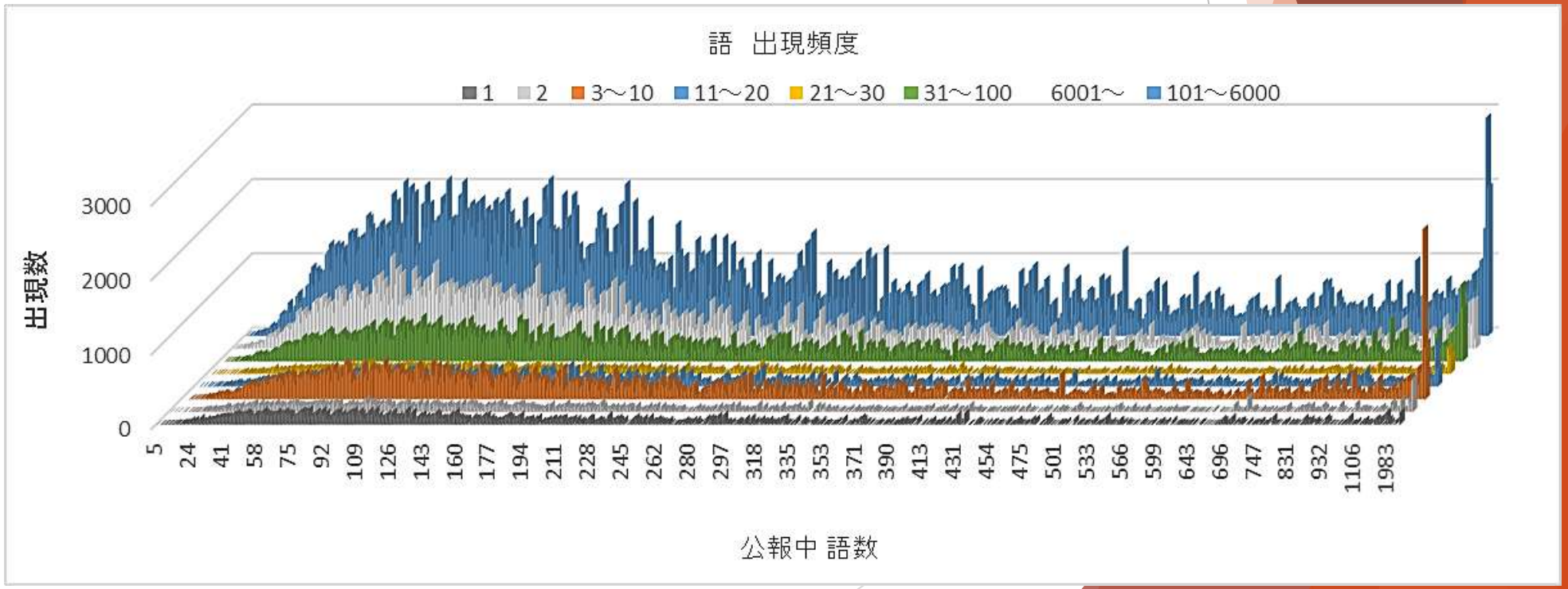


数値化テキストの特徴

単語の出願頻度は偏りが大きい
- 出現頻度1の単語数が42.5%
- 上位15単語でデータの19.1%を占める
- 単語数1.3%がデータの44%を占める

出現頻度	単語数	割合	頻度	割合
1	32914	42.5%	32555	3.1%
2	14477	18.7%	28663	2.8%
3～10	21112	27.3%	104682	10.0%
11～20	4060	5.2%	58344	5.6%
21～30	1492	1.9%	31292	3.0%
31～100	2334	3.0%	128915	12.4%
101～6000	1009	1.3%	458867	44.0%
6001～	15	0.0%	198588	19.1%

1	こと	31546
2	する	30074
3	特徴	28091
4	ある	25263
5	有する	11740
6	範囲	9917
7	含む	9376
8	なる	9156
9	2	8642
10	1	8316
11	含有する	7885
12	製造方法	7456
13	R	7232
14	あり	6421
15	3	6371
16	インク受容層	5411



文書ベクトル　one-hotベクトル

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	14	12	4	103	516	890	40	31083	7	1	516	890	2393
2	14	12	4	23217	2004	1411	226	51	345	51	476	30	643
3	14	12	4	39	67	2139	80	0	39435	156	28341	144	0
4	73	14	12	4	251	39	40	2388	1095	10	4730	0	1
5	73	14	12	4	233	24558	34	550	23276	22166	96	5391	15
6	14	12	4	6360	8107	5	8340	8	1	6360	1	8107	6
7	14	12	4	1747	8	71	51	166	19	2644	166	725	1891
8	14	12	4	33	1	33	8	71	709	11	17	49	185

出現頻度	単語数	割合	頻度	割合
1	32914	42.5%	32555	3.1%
2	14477	18.7%	28663	2.8%
3～10	21112	27.3%	104682	10.0%
11～20	4060	5.2%	58344	5.6%
21～30	1492	1.9%	31292	3.0%
31～100	2334	3.0%	128915	12.4%
101～6000	1009	1.3%	458867	44.0%
6001～	15	0.0%	198588	19.1%

合計
77,413

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	
1	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	
2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	←12
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	←14
4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	←4

機械学習では単語（整数表現）をone-hotベクトルに変換して入力

・1単語表現は辞書単語数（77413）次元の行列

・23217の単語は、23217番目が1で他の番号が0の行列

・1文書表現は文書中の単語数の行×辞書単語数の列を持つ行列

→　疎(スパース)で膨大な次元数のデータ　計算量が増大

1文章のone-hotベクトル表現

```
array([[0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 1., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

184の単語で構成されている文書は184×77413次元

文書ベクトル　Bag of Words

文書中に出現する単語を数えあげたもの(出現頻度)

・ one-hotベクトルの足し合わせでできる

・1文書表現は文書中の単語数の行×辞書単語数の列を持つ行列 → 1行×辞書単語数の列(次元削減効果)

・0-1に規格化して使用する

・それでも辞書単語数×文書数の行列を入力する必要があり計算量膨大

入力数値列を削減して計算量を削減する

特徴量抽出による方法

・TF-IDF(＋次元削減)

分散表現による方法

・Word2Vec(→Doc2Vec)

・Fasttext

ニューラルネットワークを用いた方法

・入力層におけるエンベッド層

1文書におけるベクトルの例

```
[1., 0., 0., ..., 0., 0., 0.] # 特許
[0., 1., 0., ..., 0., 0., 0.] # 請求の
[0., 0., 1., ..., 0., 0., 0.] # 範囲
 ...,
[0., 0., 0., ..., 1., 0., 0.] # 特徴とする
[0., 0., 0., ..., 0., 1., 0.] # インクジェット用
[0., 0., 0., ..., 0., 0., 1.] # インク
```

↓

足し合わせ

```
[1., 1., 1., ..., 2., 2., 4.]
```

↓

規格化(0-1)

```
[0.25, 0.25, 0.25, ..., 0.5, 0.5, 1.]
```

Point
Pythonでは
・gensimのdoc2bow
・scikit-learnのCountVectorizer
を使って
単語文字列からBoWに変換できる

文書ベクトル　TF-IDF

TF:Term Frequency（単語の出現頻度）

$$tf = \frac{\text{文書}A\text{における単語}X\text{の出現頻度}}{\text{文書}A\text{における全単語の出現頻度の和}}$$

IDF:Inverse Document Frequency（逆文書頻度）レアな語の値が高い

$$idf = log(\frac{\text{全文書数}}{\text{単語}X\text{を含む文書数}})$$

TFとIDFを掛け合わせると重要な単語が抽出できる

$$tfidf = tf * idf \quad : (\text{単語の出現頻度}) * (\text{各単語のレア度})$$

ほとんどの要素が0の行列
＝スパース（疎）な行列（ベクトル）

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [68.4151047, 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

max_featuresで指定した列数のベクトルができる

```python
from sklearn.feature_extraction.text import TfidfVectorizer

# TFIDFモデル作成（784次元）
tfidf = TfidfVectorizer(max_features=784, use_idf=True,
token_pattern=u'(?u)\\b\\w+\\b', ngram_range=(1, 1))
#   u'(?u)\\b\\w+\\b' とすれば文字列長が 1 の単語を処理対象に含める

# TFIDFモデル実行
vecs = tfidf.fit_transform(np.array(num_data))
```

Point

ここでは辞書IDをnumpy形式で渡したが、通常は単語を渡す

このとき辞書も作成される

辞書はvectorizer.vocabulary_で呼び出すことができる

次元削減

max_featuresを大きくしておいて

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [68.4151047, 0., 0., ..., 0.,
      0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]
       ])
```

密なベクトルに変換（Isomap法の例）

```
array([[-219.33025933, 217.3182299,
       -90.77008777, ..., 29.99313503,
       5.3929687, 7.37745395],
       [-232.59221305, 149.48820295,
       -32.00263952, ..., 36.79850616,
       5.67802302, -1.72013636],
       ...,
       [-435.99553792, 14.24237615,
       148.84790117, ..., 13.02483332,
       33.11647377, -15.58084723],
       [-361.41995336, -85.20967575,
       274.38743453, ..., 19.44095653,
       41.85568707, -19.91077782]])
```

```python
from sklearn import (random_projection, decomposition, ensemble, manifold)

# SRP
rp = random_projection.SparseRandomProjection(n_components=2, random_state=42)
vector = rp.fit_transform(vecs).toarray().tolist()

# t-SVD
vector = decomposition.TruncatedSVD(n_components=2).fit_transform(vecs)

# RTE
hasher = ensemble.RandomTreesEmbedding(n_estimators=200, random_state=0, max_depth=5)
X_transformed = hasher.fit_transform(vecs)
pca = decomposition.TruncatedSVD(n_components=2)
vector = pca.fit_transform(X_transformed)

# MDS
clf = manifold.MDS(n_components=2, n_init=1, max_iter=100)
vector = clf.fit_transform(vecs.toarray())

# Isomap
vector = manifold.Isomap(n_neighbors, n_components=2).fit_transform(vecs.toarray())

# LTSA
clf = manifold.LocallyLinearEmbedding(n_neighbors, n_components=2, method='ltsa')
vector = clf.fit_transform(vecs.toarray())

# LLE
clf = manifold.LocallyLinearEmbedding(n_neighbors, n_components=2, method='standard')
vector = clf.fit_transform(vecs.toarray())

# t-SNE　3次元以下
tsne = manifold.TSNE(n_components=2, init='pca', random_state=0)
vector = tsne.fit_transform(vecs.toarray())
```

Word2Vec

Googleの研究者らによって提唱されたモデル

CBOW　周辺の語から中央の語を予測
語順は順不同

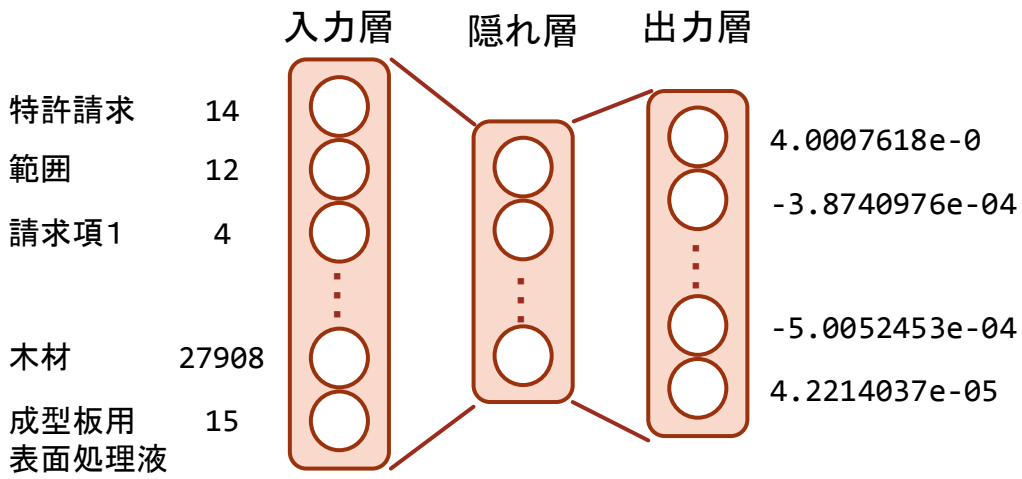
特許	?	範囲	範囲	?	特許

Skip-gram　中心語が与えられ周囲の語を予測

?	防止	?	?	剤	?

?	含有	?

ニューラルネットワークで上記予測問題を解く



単語ベクトルから文書ベクトルの作成

ベクトル合成（ベクトルの平均）

```
from gensim.models import word2Vec

# 分かち書き文書ファイル：word_data
sentences = word2vec.LineSentence(word_data)
# sentences = word2vec.Text8Corpus(word_data) # これでもよい

# モデルの定義および実行
model = word2vec.Word2Vec(sentences, size=784, min_count=20,
window=5)
model.save('model_filename.model')

# 文書ベクトルの取り出し
vector = []
for i in range(len(word_data)):
    vector.append(model.wv[word_data[i]])

# 文書ベクトル作成
feature_vec = np.zeros((784,), dtype='float32') # 受け皿用空行列
for word in vector:
    feature_vec = np.add(feature_vec, model[word_data[i]]) # 加算
feature_vec = np.divide(feature_vec, len(word_data)) # 単語数で割る
```

Doc2Vec

Doc2Vecを使えば一発で文書ベクトルが得られる

タグ付き文書の構成（TaggedDocument）

ワード部分　`words=word_data[i].tolist()`
　文書ごとにユニークな値（公報番号等）のリスト

タグ部分　`tags=label_data[i].tolist()`
　単語リストを文書ごとに並べた2次元リスト
　今回は辞書IDを str 形式のリストにして渡した

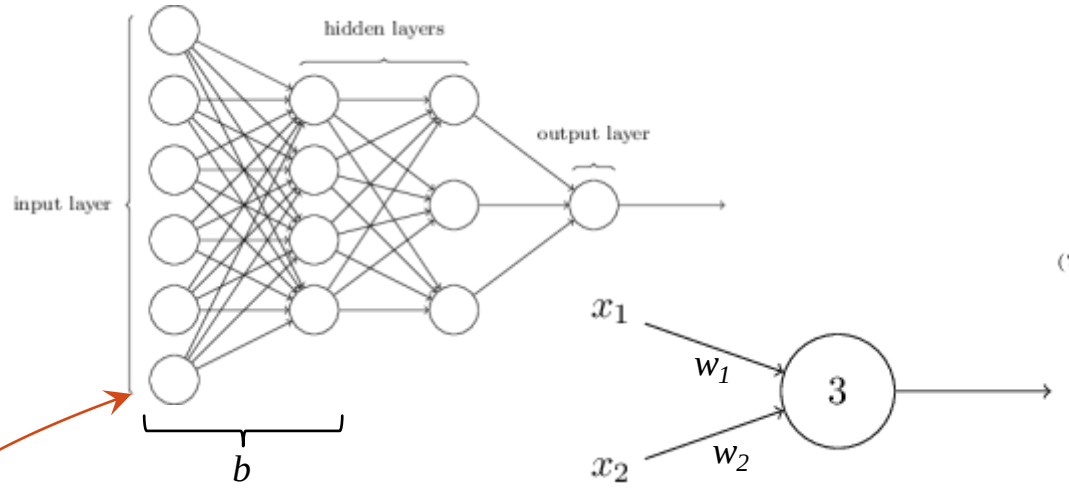
```python
from gensim.models.doc2vec import Doc2Vec
from gensim.models.doc2vec import TaggedDocument

# タグ付き文書の作成
trainings = []
for i in range(word_data.shape[0]):
    trainings.append(TaggedDocument(words=word_data[i].tolist(),
tags=label_data[i].tolist()))

# モデルの定義および実行
model = Doc2Vec(documents = trainings, dm = 0, dbow = 0,
vector_size= 784, window=3, alpha=0.025, min_count=1, sample=0,
seed=0, workers=1, min_alpha=0.0001, hs=1, negative=0, dm_mean=0)

model.save('model_filename.model')

# 文書ベクトルの取り出し
for i in range(len(train_x)):
    data = model.docvecs[train_label_data[i]]
for i in range(len(test_x)):
    data = model.docvecs[test_label_data[i]]
```

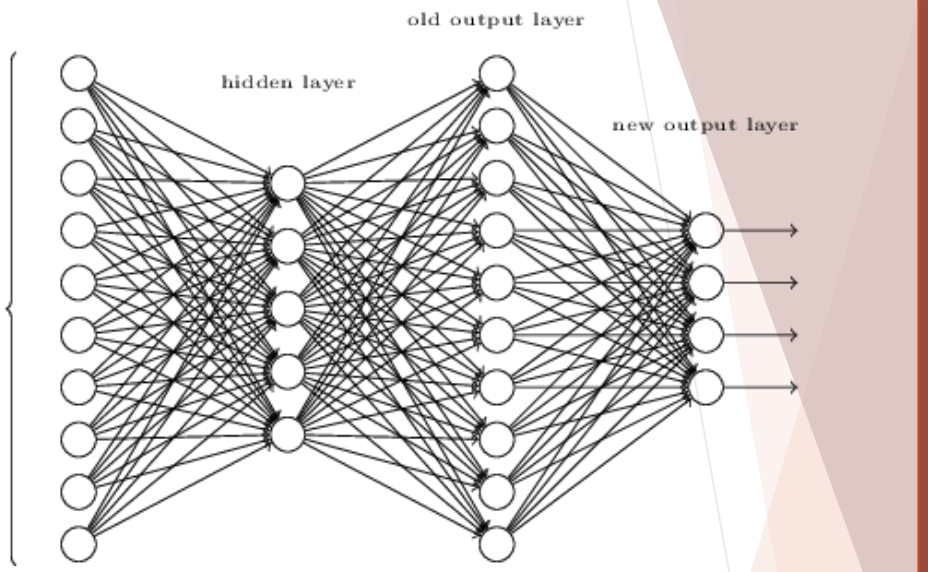
出力
文書ごとに vector_size = 784 次元のベクトル（numpy形式）

```
array([0.38228413, 0.5496637, 0.42696872,
       0.5104867, 0.44964933,
       ...
       0.40989193, 0.53345186, 0.5641854,
       0.46606722], dtype=float32)
```

Embed層

ニューラルネットワークにおける学習は





$x_1w_1+x_2w_2+\cdots+x_nw_n+b$ の $w_1, w_2, \cdots, w_n, b$ を繰り返し繰り返し調整する作業

```
# Kerasの入力層
model.add(Embedding(max_features, 50, input_length=maxlen))
```

単語語彙数=max_features、入力文の長さ=maxlenの辞書ID行列が、50次元のベクトルに変換される

```
array([
[1.400e+01, 1.200e+01, 4.000e+00, ..., 0.000e+00, 0.000e+00],
       [1.400e+01, 1.200e+01, 4.000e+00, ..., 0.000e+00, 0.000e+00],
           ...,
       [1.451e+03, 1.200e+01, 3.000e+01, ..., 0.000e+00, 0.000e+00],
       [1.451e+03, 1.200e+01, 4.312e+03, ..., 0.000e+00, 0.000e+00]
])
```

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_8 (Embedding)      (None, 4664, 50)          6532600
=================================================================
Total params: 6,884,345
Epoch 8/8
1097/1097 [==============================] - 112s 102ms/step -
loss: 1.9499e-05 - acc: 1.0000 - val_loss: 0.9095 - val_acc: 0.8249
```

ここまでのまとめ

特許情報（テキスト）の特徴

- 可変長かつ極端に長いものが存在（0で埋めるか？末尾を切るか？）
- 単語の数が多い
- 出現頻度が1,2の単語が単語全体の50%を超える
- 出現頻度上位の単語は技術用語ではない

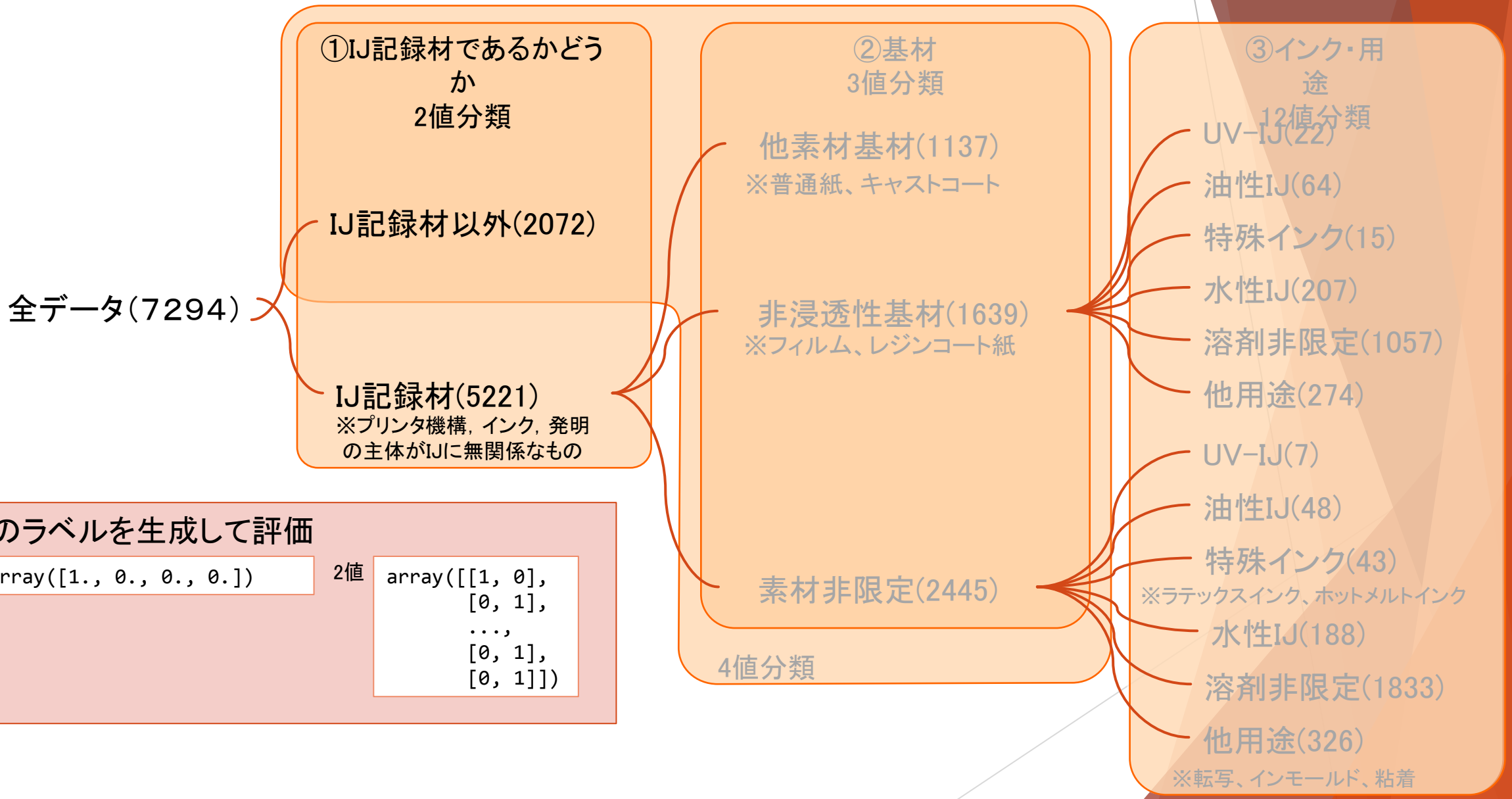
出現頻度	単語数	割合	頻度	割合
1	32914	42.5%	32555	3.1%
2	14477	18.7%	28663	2.8%
3～10	21112	27.3%	104682	10.0%
11～20	4060	5.2%	58344	5.6%
21～30	1492	1.9%	31292	3.0%
31～100	2334	3.0%	128915	12.4%
101～6000	1009	1.3%	458867	44.0%
6001～	15	0.0%	198588	19.1%

自然言語の数値化処理

- 不要な記号等を除去（クレンジング）
- 形態素に分割 → 品詞を考慮した連結
- 辞書IDの付与
- 使用頻度（TF）、特徴量抽出（TF-IDF）
- サイズが小さなベクトル表現（分散表現，ニューラルネットワークの入力層）

Next → 本研究で用いた機械学習モデル

データ分類(ラベル付与)基準



2値のラベルを生成して評価

2値
```
array([1., 0., 0., 0.])
```

2値
```
array([[1, 0],
       [0, 1],
       ...,
       [0, 1],
       [0, 1]])
```

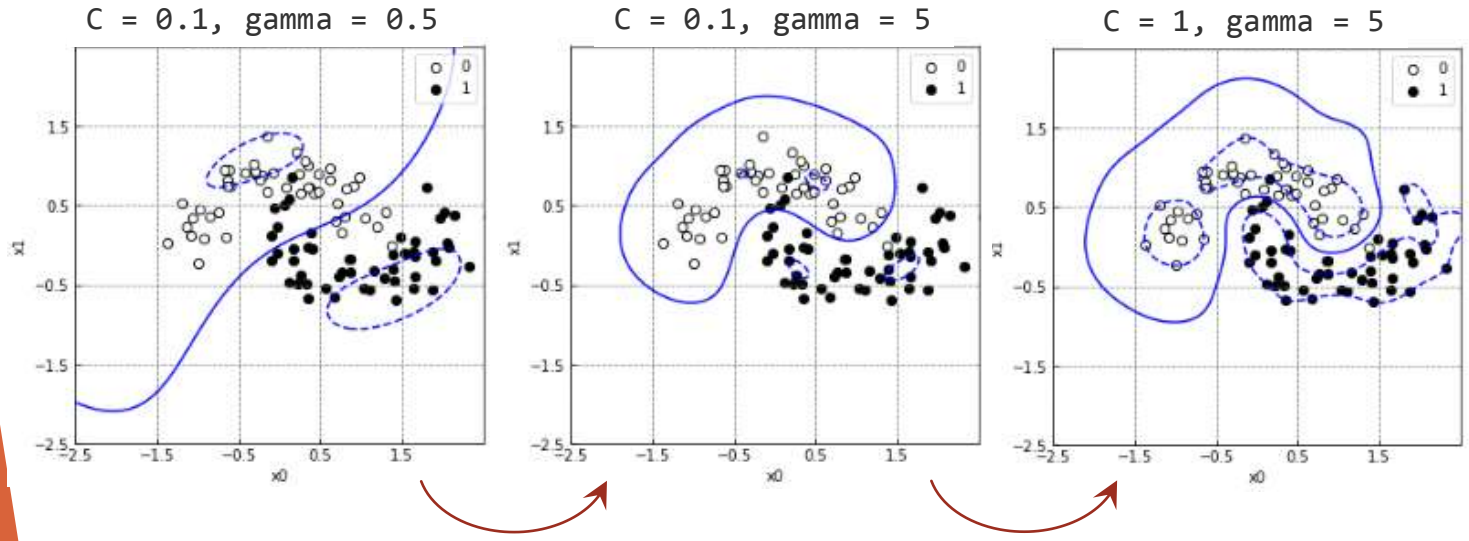
非線形SVM（教師あり学習として）

決めるべきハイパーパラメータ
コストパラメータ: C 誤分類をどの程度許容するか
RBFカーネルのパラメータ: γ　境界線の複雑さ

Scikit-learn がもつ月形の2値データセットの場合

```
from sklearn.datasets import make_moons
```

実線：識別境界
破線：マージン



γ増大：境界線が複雑に　　C増大：マージン大きく

```
from sklearn.svm import SVC

# 非線形SVMで学習
classifier = SVC(kernel='rbf')

# パラメータのレンジを設定 各パラメータ毎に学習を行う
c_range = [0.1, 1, 10, 100]
gamma_range = [0.5, 5, 15, 50]

# 学習
cnt = 0
for i in range(X_dataset.shape[0]):
    Zd = classifier.predict([X_dataset[i]])[0]
    if Zd == y_dataset[i]:
        cnt = cnt + 1
    print('train_accuracy =', cnt/X_dataset.shape[0])

# 検証
cnt = 0
for i in range (x_test.shape[0]):
    Zd = classifier.predict([x_test[i]])[0]
    if Zd == y_test[i]:
        cnt = cnt + 1
    print('test_accuracy =', cnt/x_test.shape[0])
```

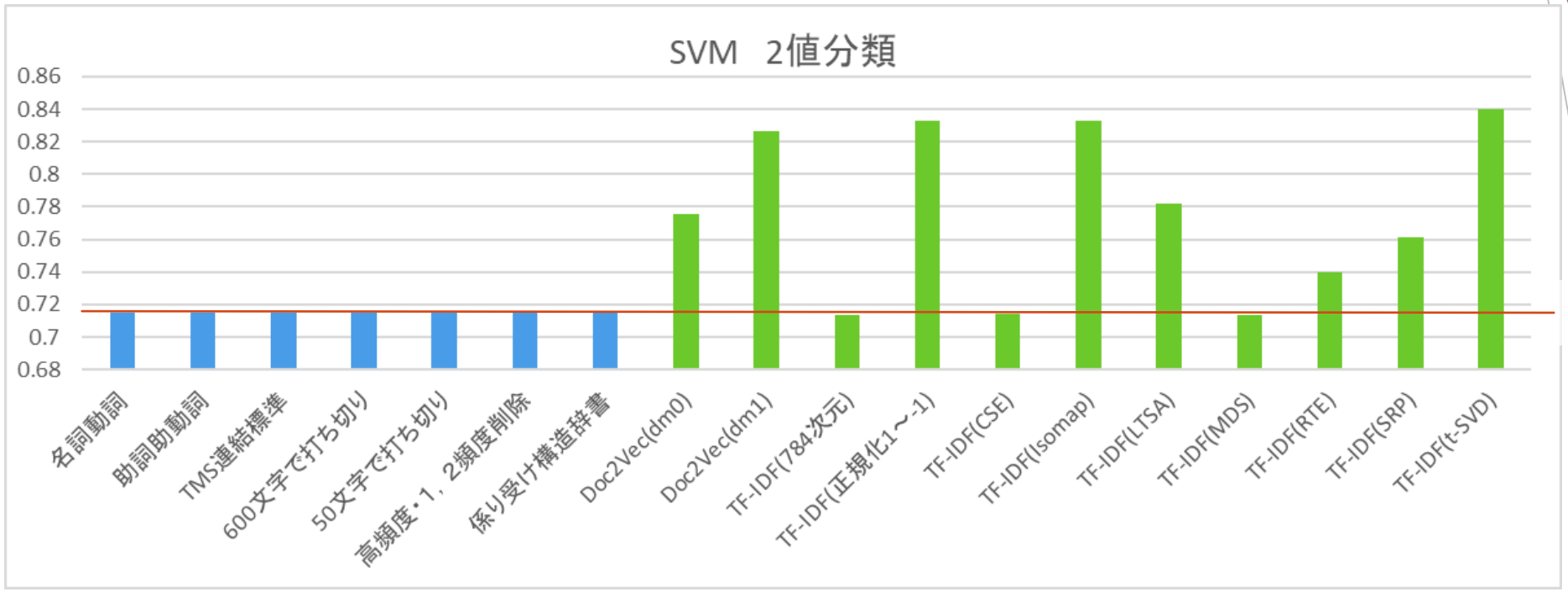
特許データセット
X_dataset:(1097, 1701)
x_test:(6197, 1701)

C と gamma を適宜設定し
最もよくフィットする
組み合わせを見つける

```
C = 0.8 gamma = 10
train_accuracy = 1.0
test_accuracy = 0.7150
```

学習データにはよく適合
未知のデータには不適合

2値分類　結果

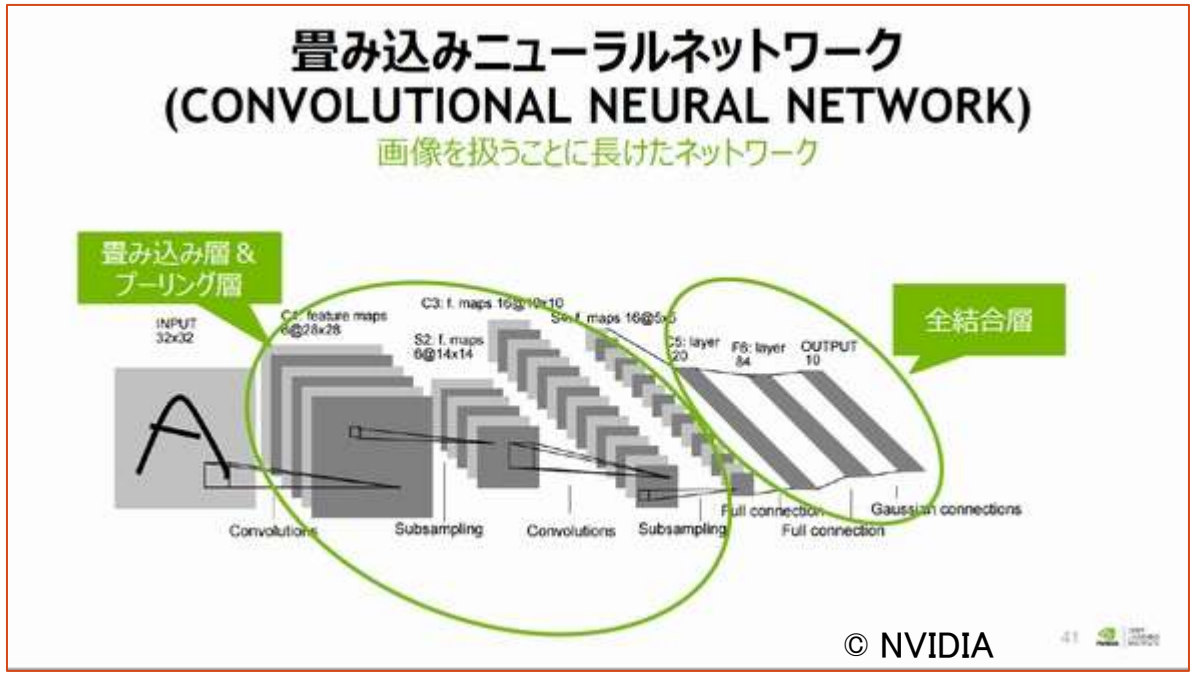


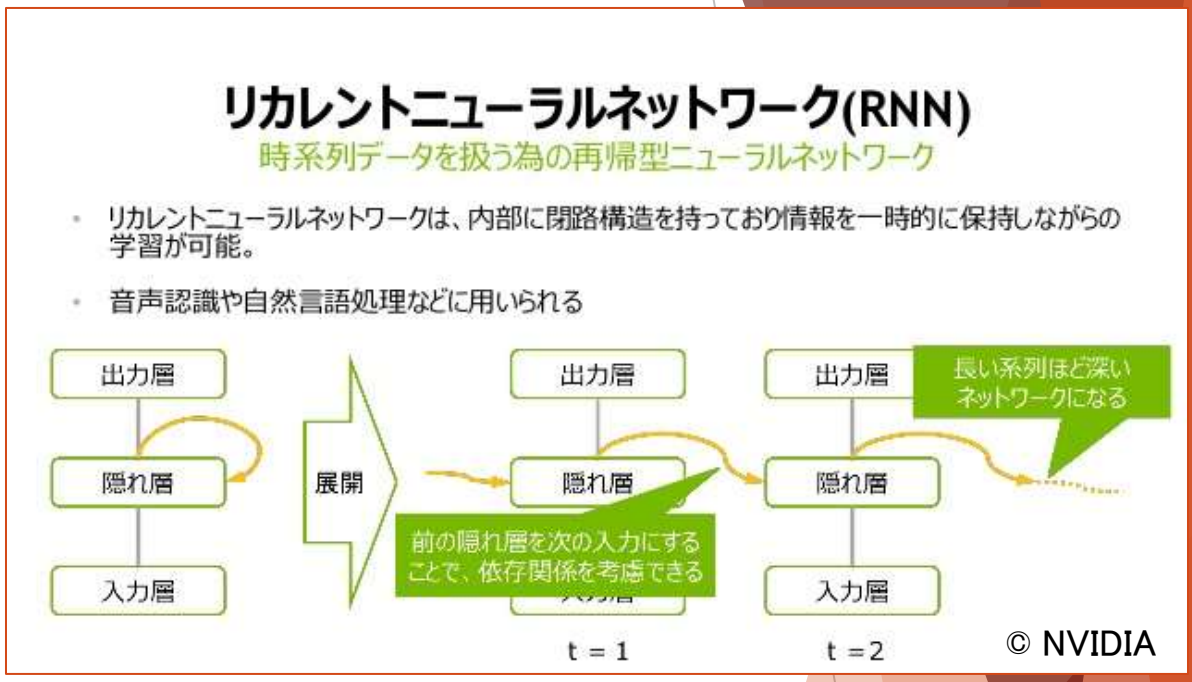
青のバー：辞書ID入力 をそのまま入力しても学習が進まない

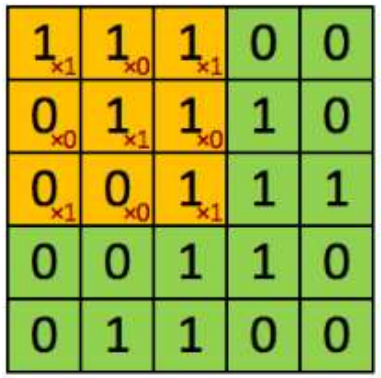
緑のバー：文書ベクトル入力系
・Doc2Vecを入力すると学習が進む
・TF-IDFの結果をそのまま入力しても学習が進まない
・規格化（-1～1）または次元削減（一部の方法）すると学習が進む

```
c= 0.8 gamma= 30
train_accuracy = 1.0
test_accuracy = 0.826690334032596
```

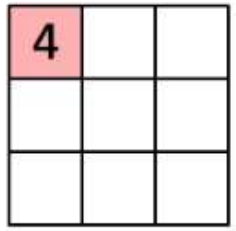
CNN,RNN（教師あり学習）







Image



Convolved
Feature

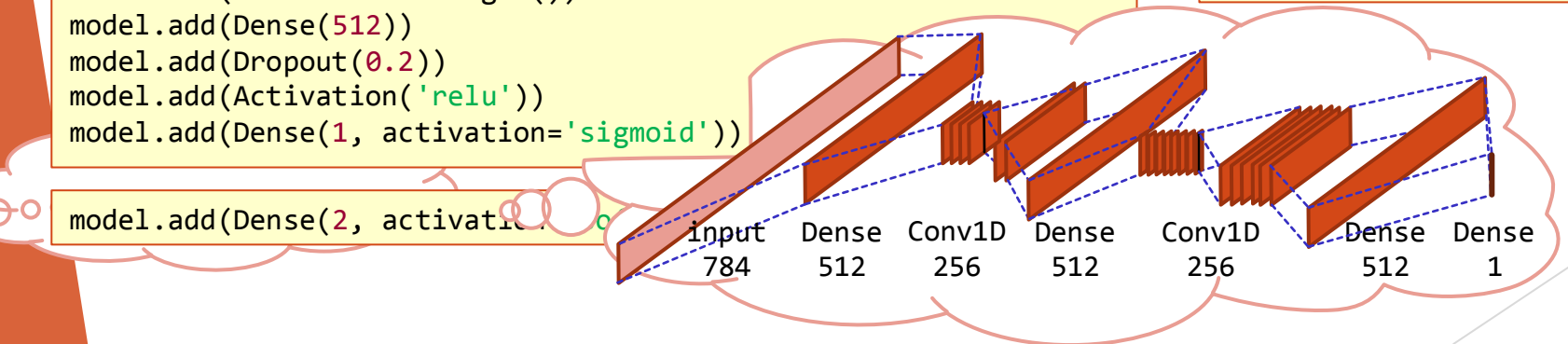
← 2次元畳み込み層を持つネットワークは2次元データ(画像)の処理が得意

自然言語処理は1次元畳み込みネットワーク　または
再帰型ネットワークが適している

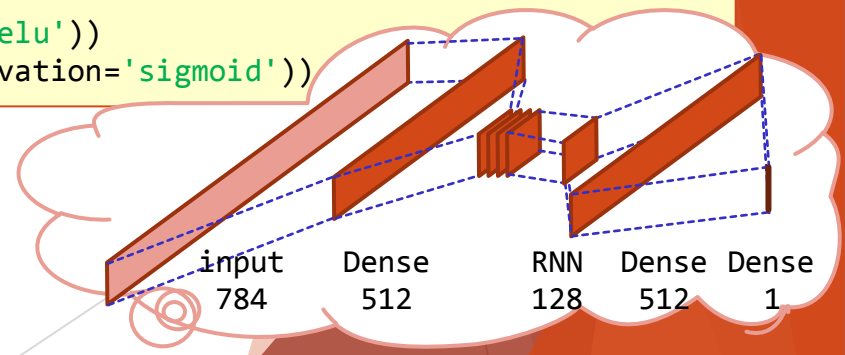
CNN,RNN（教師あり学習）

```
from keras.models import Sequential
from keras.layers.core import Activation, Dense
from keras.layers import Dropout, Embedding, Reshape
from keras.layers import Conv1D, GlobalMaxPooling1D
model = Sequential()
# 分散表現になっている場合
model.add(Dense(512, input_shape=(784,)))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Reshape((-1, 128)))
# 分散表現になっていない場合
model.add(Embedding(max_features, 50, input_length=maxlen))
# ----------------------------------------------------
model.add(Dropout(0.2))
model.add(Conv1D(256, 3, padding='valid', activation='relu',
      strides=1))
model.add(GlobalMaxPooling1D())
model.add(Dense(512))
model.add(Dropout(0.2))
model.add(Activation('relu'))
model.add(Reshape((8, 64)))
model.add(Conv1D(256, 3, padding='valid', activation='relu',
      strides=1))
model.add(GlobalMaxPooling1D())
model.add(Dense(512))
model.add(Dropout(0.2))
model.add(Activation('relu'))
model.add(Dense(1, activation='sigmoid'))
```

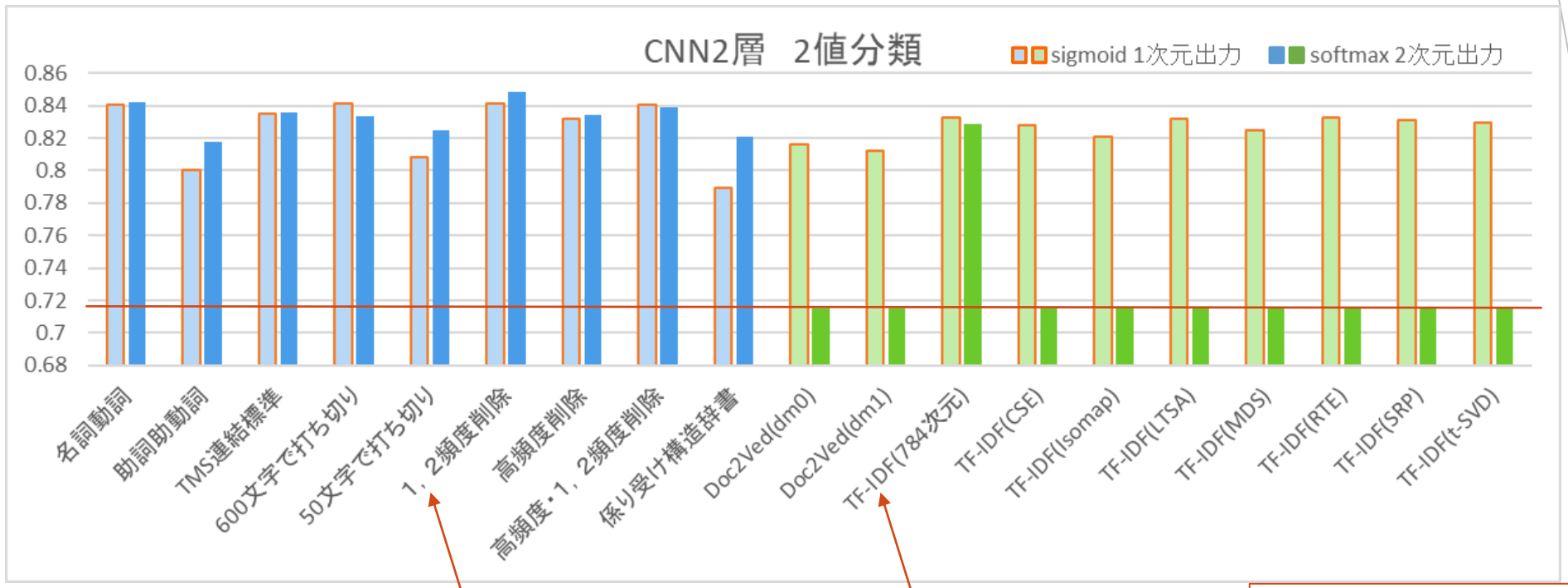
```
model.add(Dense(2, activati...  o
```



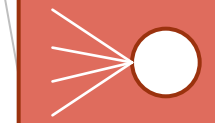

```
from keras.models import Sequential
from keras.layers.core import Activation, Dense
from keras.layers import Dropout, Embedding, Reshape
from keras.layers import SimpleRNN, LSTM, GRU, Bidirectional
model = Sequential()
# 分散表現になっている場合
model.add(Dense(512, input_shape=(784,)))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Reshape((-1, 128)))
# 分散表現になっていない場合
model.add(Embedding(max_features, 50, input_length=maxlen))
# ----------------------------------------------------
model.add(Dropout(0.2))
model.add(Bidirectional(SimpleRNN(64, dropout=0.2,
      recurrent_dropout=0.3)))
model.add(Dense(512))
model.add(Dropout(0.2))
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Activation('relu'))
model.add(Dense(1, activation='sigmoid'))
```



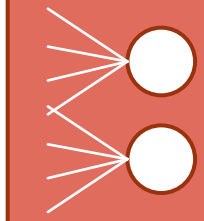
2値分類　結果

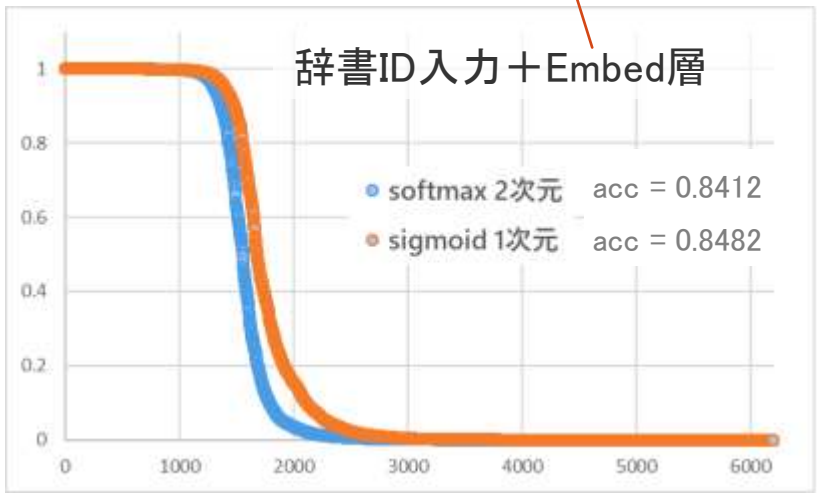


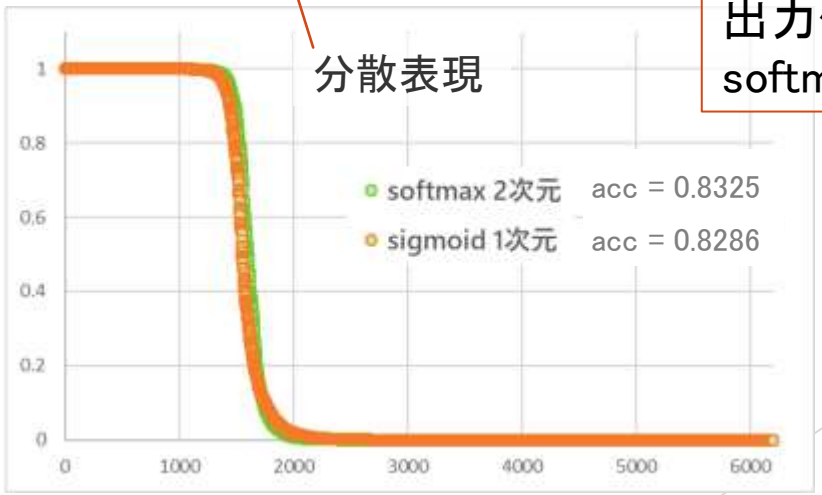
出力1次元（sigmoid）



出力2次元（softmax）







出力値をプロットすると
softmaxを使用した出力2次元が急峻

分類精度≠急峻度

ここまでのまとめ

２値分類

非ニューラルネットワーク系

・代表としてSVM　分散表現ベクトルを入力した

・Doc2Vec、TF-IDFの特定の場合に学習効果が見られた

・規格化手法、次元削減手法の影響が大きい

ニューラルネットワーク系

・代表として1次元CNN2層モデルを使用した

・Embedd層を取り払い文書ベクトルを直接入力することで計算資源の省力化になる

・通常のsigmoid関数を使用した1次元出力とsoftmax関数を用いた2次元出力を比較

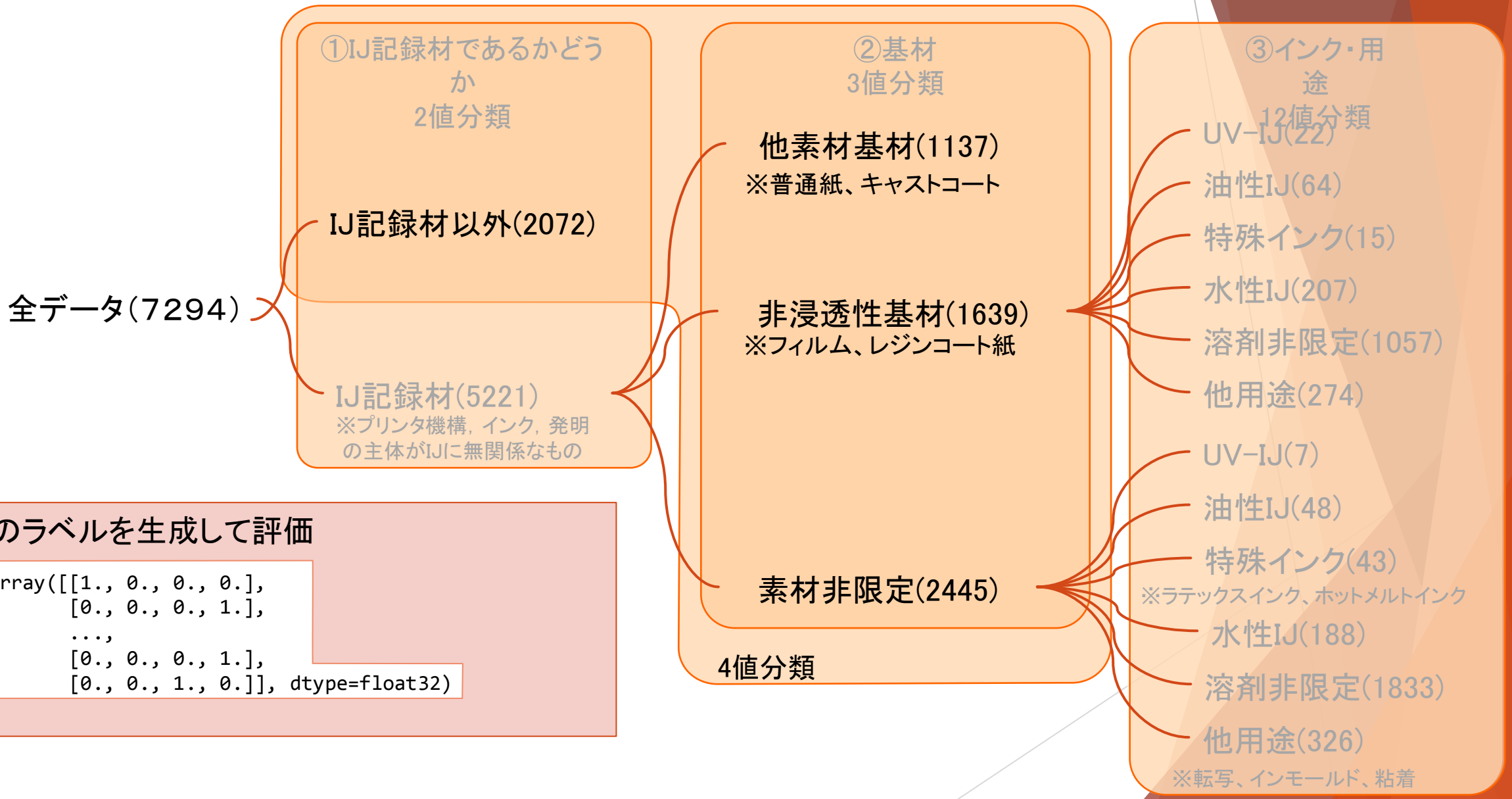
・softmax2次元出力は確率分布が急峻になる傾向がある

過学習状態から脱することはできなかった

・誤答は確率0～1の全範囲に渡って存在し、正答/誤答を確率から分離できない

Next → 4値分類に挑戦

データ分類(ラベル付与)基準



4値のラベルを生成して評価

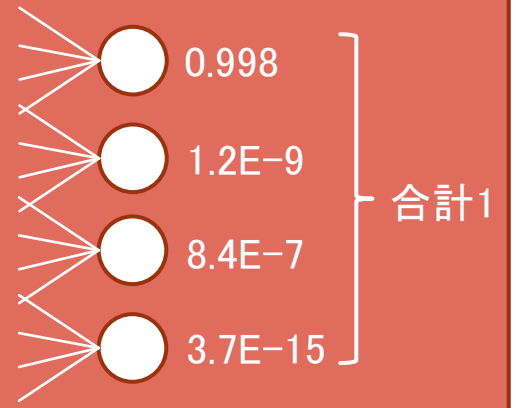
4値

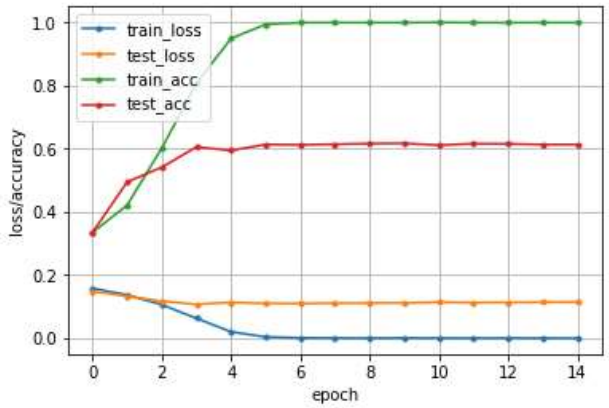
```
array([[1., 0., 0., 0.],
       [0., 0., 0., 1.],
       ...,
       [0., 0., 0., 1.],
       [0., 0., 1., 0.]], dtype=float32)
```

4値分類

```
model.add(Dense(4, activation='softmax'))
```

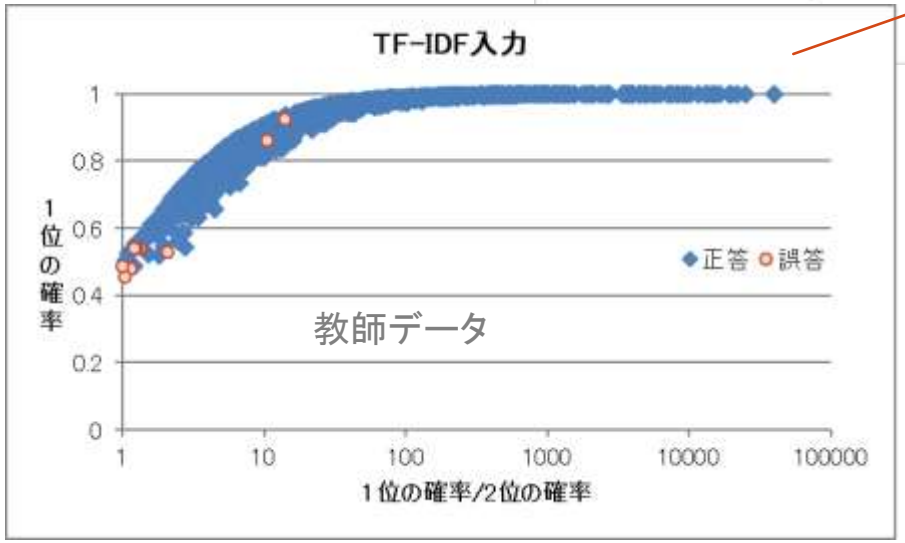
出力4次元（softmax）

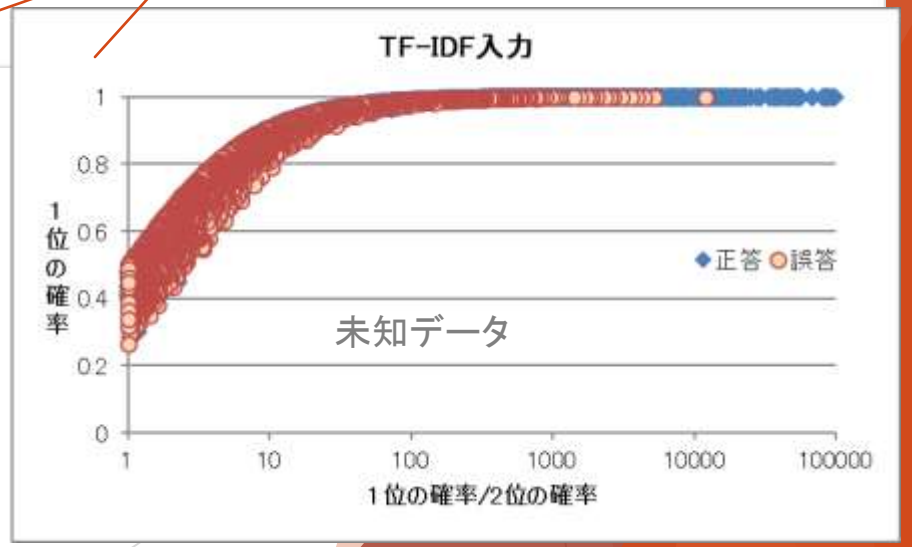




（仮説）
判定に自信がある場合
上位1位の値が突出

横軸：1位の値/2位の値
（対数軸プロット）
縦軸：1位の値（実数プロット）

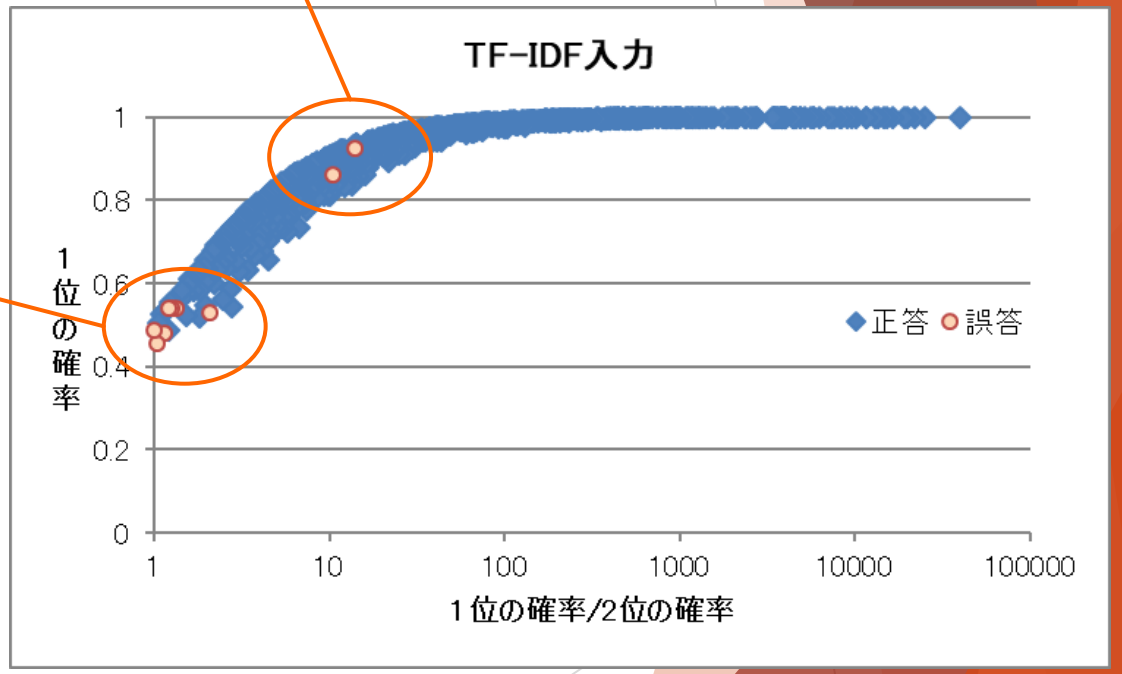




4値分類

キーとなる記載	人間判定	機械判定	正答
合成樹脂層の一部に多孔質領域を形成する･･･3次元造形物の製造方法。	非浸透原紙	インクジェッ記録材ではない	機械判定
カチオン性樹脂変性シリカ分散液の製造方法。	インクジェッ記録材ではない	原紙の種類問わない	人間判定
インク受理層を支持体表面に連続して積層	インクジェッ記録材ではない	非浸透原紙	人間判定
前記インク受容層から前記支持体が剥離可能	インクジェッ記録材ではない	非浸透原紙	悩んで再付与
前記インク受容層はキャストコート法により設けられてなる	パルプ原紙	原紙の種類問わない	人間判定
該保護シートは該最外のインク受容層面と接する面のみ耐水性加工され･･･ることを特徴とするインクジェット記録用紙の包装体。	インクジェッ記録材ではない	非浸透原紙	人間判定
支持体上に･･･及びアクリル系樹脂を含有する下塗り層を設け	インクジェッ記録材ではない	非浸透原紙	人間判定

キーとなる記載	人間判定	機械判定	正答
記録用原紙に塗工して、	パルプ原紙	原紙の種類問わない	機械判定
インクジェット記録用紙の	パルプ原紙	原紙の種類問わない	機械判定



1位の確率/2位の確率 の値によってミスラベルの検出ができそう

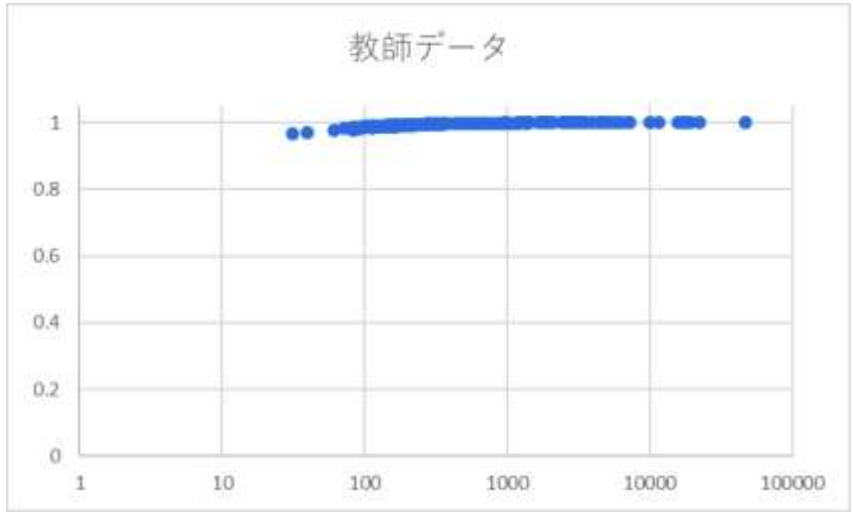
新規ラベル付与の挑戦

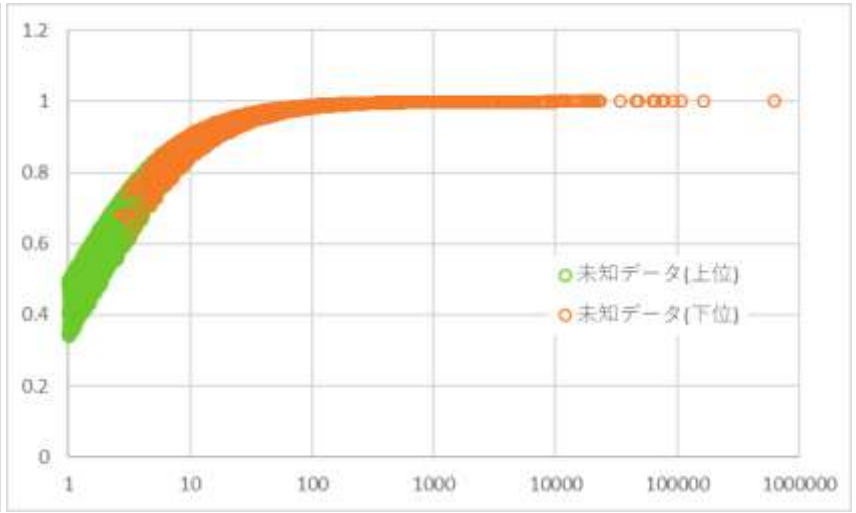
python™ Keras

ここまで
教師データ1097件
検証データも含め全ラベル付与

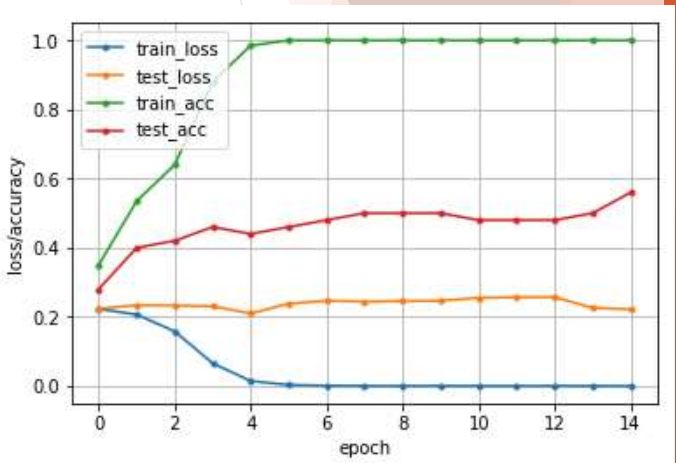
⇒

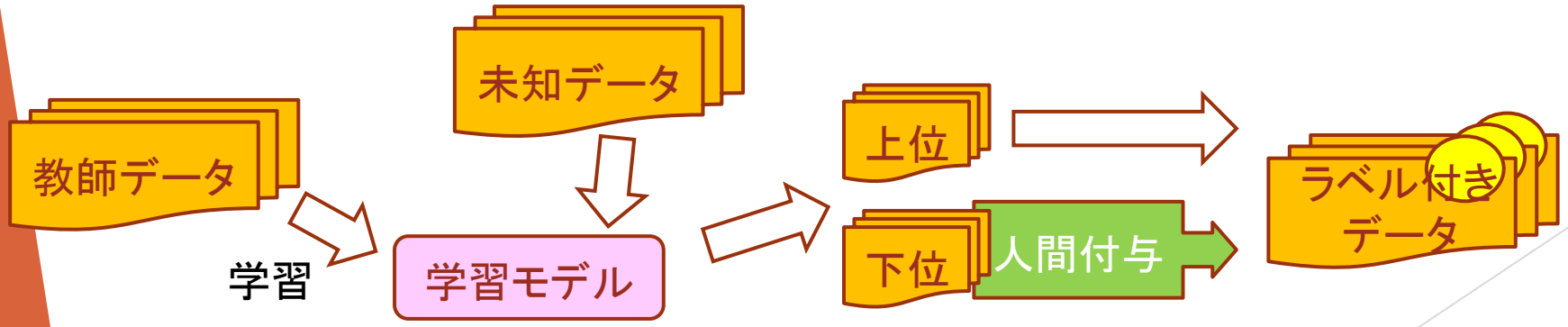
そこで
教師データ　少量で（100～200件）
検証データのラベル付与を試みる





ミスラベルを含む教師データへの過剰適合





	正答数	データ数	正答率
上位	2111	3461	60.99%
下位	1570	3513	44.69%

ここまでのまとめ

4値分類

ニューラルネットワーク系

・過学習状態から脱することはできなかった

・特定番号の分類確率が突出することに注目

・softmax関数出力の1位/2位を用いる

・多量の教師データに紛れ込んだミスラベルを検出できる可能性がある

・少量の教師データから分類用ラベルを付与することはできなかった

文書ベクトルの今後

より効率よく情報を詰め込むために

・最適なベクトルサイズ

・Doc2VecとTFとのコラボレーション　SCDV

・Fasttext

・Skip-thought

・係り受け構造等、文章構造を利用したモデル

応用場面

・文章中の技術用語を分類に直接関連づける

・OCRの読み取りエラーの訂正

謝辞

本報告は2018年度の「アジア特許情報研究会」のワーキングの一環として報告するものであり、研究会の皆様には情報の提供及び数々のアドバイスをいただきました。ここに改めてお礼申し上げます。

おことわり

本資料に掲載のソースコードは、そのままでは動作しません。

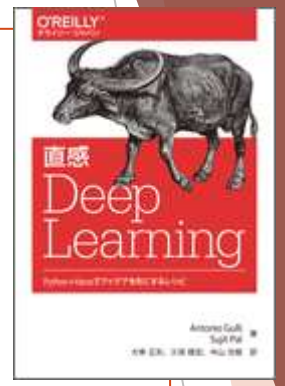
また、ソースコードを使用したことによる一切の損害に対し筆者は責任を負いません。

参考にした書籍

直感Deep Learning: Python × Kerasでアイデアを形にするレシピ

Antonio Gulli、Sujit Pal 著, 大串 正矢, 久保 隆宏, 中山 光樹 訳

O'Reilly Japan, 2018/8/17



入門 自然言語処理

Steven Bird, Ewan Klein, Edward Loper 著、萩原 正人, 中山 敬広 訳

O'Reilly Japan, 2010/11/11

ゼロから作るDeep Learning ❷ —自然言語処理編

斎藤 康毅 著

O'Reilly Japan, 2018/7/21

Google Cloud Platformではじめる機械学習と深層学習

吉川 隼人 著

リックテレコム, 2017/12/12

pythonによるテキストマイニング入門

山内 長承 著

オーム社, 2017/11/28

参考にしたweb記事

高次元データの次元削減および2次元プロット手法

https://qiita.com/TomHortons/items/2064870af3f3f7f2b209

scikit-learnによる次元圧縮とクラスタリング

https://qiita.com/hiroto0227/items/951c8c8953f9f37f4a79

Doc2Vecの仕組みとgensimを使った文書類似度算出チュートリアル

https://deepage.net/machine_learning/2017/01/08/doc2vec.html

[gensim]Doc2Vecの使い方

https://qiita.com/asian373asian/items/1be1bec7f2297b8326cf

文章をよしなに分散表現しよう

https://trap.jp/post/295/

tf-idfについてざっくりまとめ_理論編

https://dev.classmethod.jp/machine-learning/yoshim_2017ad_tfidf_1-2/

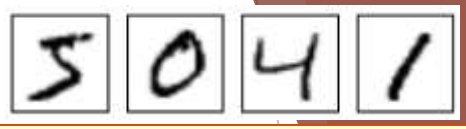
Keras Documentation （日本語）

https://keras.io/ja/

Keras examples

https://github.com/keras-team/keras/tree/master/examples

TensorFlow と Keras：MNIST実装例



Kerasはニューラルネットワーク層構造を直感的に表現できる
Keras内部でTensorFlowが動作
verbose=1 で学習の進行を可視化でき、historyデータでグラフも

```
Epoch 8/8
1097/1097 [==============================] - 1s 1ms/step - loss:
    1.8391e-04 - acc: 1.0000 - val_loss: 1.3657 - val_acc: 0.8359
```

```python
import tensorflow as tf

x = tf.placeholder(tf.float32, [None, 784])    # 訓練画像用変数
W = tf.Variable(tf.zeros([784, 10]))           # 重み
b = tf.Variable(tf.zeros([10]))                # バイアス
y = tf.nn.softmax(tf.matmul(x, W) + b)         # ソフトマックス回帰を実行

y_ = tf.placeholder(tf.float32, [None, 10])    # y_は正解データのラベル
cross_entropy = -tf.reduce_sum(y_*tf.log(y))   # 損失関数

# 勾配降下法を用い交差エントロピーが最小となるようyを最適化する
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
init = tf.initialize_all_variables()           # 変数Veriableの初期化を実行する

sess = tf.Session()                            # 学習のSessionを開始する
sess.run(init)
for i in range(1000):
    sess.run(train_step, feed_dict={x: train_x, y_:train_y})

correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))  # 予測
accuracy = tf.reduce_mean(tf.cast(correct_prediction, 'float'))
# 精度の計算 Trueならば1、Falseならば0を出力
print(sess.run(accuracy, feed_dict={x: test_x.images, y_: test_y.labels}))
```

Keras

```python
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam

# モデル作成
model = Sequential()
model.add(Dense(512, activation='relu',
input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))

batch_size = 128  # バッチサイズ
epochs = 20       # エポック数

# モデルの生成（コンパイル）
model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])

# モデルの最適化とデータの保持
history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    verbose=1,
                    validation_data=(x_test, y_test))

# 精度の計算 Trueならば1、Falseならば0を出力
score = model.evaluate(x_test, y_test, verbose=1)
print('Test loss:', score[0], 'Test accuracy:', score[1])
```