機械学習

情報技術グループ　澤田麻沙代

はじめに

近年、AI が飛躍的な進歩を遂げている。一昨年度、京都大学で開催されるということで 10 年ぶりに参加した日本地震学会秋季大会では、AI を取り入れた研究発表をいくつか聴講した。また、昨年度は新型コロナ渦のため一堂に会しての技術室研修が中止となり、各自で個人研修を受講することになったため、機械学習・ディープラーニングの基礎となる「Python+数学講座」の e-learning を受講した。今後、機械学習・ディープラーニングを学んでいきたいと思っているものの、何から手をつけてよいかわからない状態である。そこで、この技術室報告書の執筆を機に、学んだことを整理しながら機械学習に触れてみたい。

Python とライブラリ・ツール

Python は、多くのデータサイエンスアプリケーションの共通言語となっており、データの読込、可視化、統計、画像処理などの豊富なライブラリが用意されている。昨年度受講した「Python+数学講座」の e-learning では、機械学習で使われる数学の基礎と Python の基本的な文法、Numpy, Pandas, matpotlib の使い方について学習した。今回は、与えられた課題ではなく、自分でデータを準備して試行錯誤してみる。使用する主なライブラリやツールの概要は以下のとおり。

NumPy

Python で科学技術計算をするための基本的なツールの 1 つ
多次元配列、線形代数やフーリエ変換、疑似乱数生成器など、高レベルの数学関数が用意されている

SciPy

Python で科学技術計算ライブラリ
Numpy よりも関数が充実している

Scikit-learn

Numpy と SciPy に依存するオープンソース機械学習ライブラリ
最先端の機械学習アルゴリズムが用意されている

Jupyter Notebook

ブラウザ上でコードを実行できるツール（インタラクティブな環境）
Python だけでなく、さまざまなプログラミング言語をサポートしている

※Google のアカウントと Web ブラウザさえあれば、クラウド上で Jupyter Notebook を利用することができる Google Colaboratory というサービスもある

matplotlib
科学技術計算向けのグラフ描画ライブラリ

Pandas
データを変換したり解析したりするためのライブラリ
Pandas の DataFrame はテーブル（表）のようなもので、エクセルのスプレッドシートに似ている

Anaconda
大規模データ処理、予測解析、科学技術計算向けの Python ディストリビューション
上記に挙げたライブラリ・ツールをすべて含んでいる

Anaconda のダウンロードページより Windows 64bit 版の実行ファイルを入手し、Windows10 の PC にインストールした。インストール後、Anaconda Navigator より Jupyter Notebook を Launch するとブラウザが起動し、Jupyter Notebook が開く。新しい Notebook を作成し簡単なコードを書いて実行すると、コードのすぐ下に実行結果が表示される。

Pandas を読み込んだコードを書いて実行すれば表で出力できるし、matplotlib を読み込んだコードを書いて実行すればグラフで出力できる。Jupyter Notebook は、使用するデータ項目や変数、閾値、条件などを変えながら、処理に時間を要することなく「実行⇔結果確認」を繰り返すことができるので、データがどのようなものであるか眺めるのに適している。

機械学習について
【手順】
⓪目的を考える
①データを準備
②データを観察
③モデルを決める
④モデルを訓練する
⑤訓練したモデルを評価する

⓪目的を考える
「はじめに」でも書いたように、技術報告書を執筆するという必要に迫られた状況に自分をおいて、機械学習を学び始めることが大きな目的である。

とはいえ、データを準備する前段階として、機械学習そのものの目的を考える必要が

ある。技術室報告を書きあげることを考えると、なるべく単純なものがよいだろう。できれば、業務で目にしているような情報を使いたい。そして、全く相関関係のなさそうなデータでは意味がない。また、データは多いほうがよいが、準備に時間がかからないようにしたい。

そこで思いついたのが、気象庁や USGS が公開している震源データであった。地震が繰り返し起こる場所はだいたい決まっているので、「地震発生日時」「震源の位置」「マグニチュードの大きさ」を何らかの機械学習モデルに当てはめてみようと思った。結果は全く予想できていないが、適合しないとしても、いくつかのモデルに当てはめてその違いを報告できればよいかと思った。気象庁よりも USGS のほうが地球全体の傾向を見ることができて面白そう、という理由で、USGS のデータを使うことにした。

①データの準備

USGS の Search Earthquake Catalog よりダウンロード

データ期間：1920/01/01～2021/04/15

抽出条件：マグニチュード 4.5 以上

使用したデータ項目：time　latitude　longitude　depth　mag　magType

※ダウンロードしたデータの「time」項目は「日時」であったが、「年」「月」「日」「時」「分」に項目を分けて使用

②データを観察

以下は Jupyter Notebook の画面である。「In」に Python コードを書き、実行すると「Out」に結果が出力される。

describe メソッドを使えば、統計量を瞬時に表示できる。データの個数は count 値からわかるように 260708 個（depth 列が 260075 なのは欠損値 NaN が 633 個あるため）。count 以外に表示される統計量の項目は、mean: 平均値、std: 標準偏差、min: 最小値、25%: 第一四分位数、50%: 中央値（=median）、75%: 第三四分位数、max:最大値。

In [1]:
```
# pandas インポート
import pandas as pd
# csvファイル読み込み
df = pd.read_csv('./19200101-20210415_2.csv')
```

In [2]:
```
# 読み込んだデータの先頭10行表示
df.head(10)
```

Out[2]:

	year	month	day	time_h	time_m	latitude	longitude	depth	mag	magType
0	1920	1	1	2	35	33.200	-116.700	NaN	5.0	ml
1	1920	1	24	7	9	48.800	-123.000	NaN	5.5	ml
2	1920	2	2	11	22	-4.967	152.017	35.0	7.8	mw
3	1920	2	8	5	24	-35.809	109.537	15.0	6.4	mw
4	1920	2	10	22	7	18.619	-67.339	15.0	6.4	mw
5	1920	3	17	18	37	1.575	95.555	15.0	6.3	mw
6	1920	3	20	18	31	-35.848	-110.726	10.0	7.0	mw
7	1920	3	29	5	8	50.957	-127.233	15.0	6.4	mw
8	1920	4	2	1	4	-8.075	148.027	15.0	6.3	mw
9	1920	4	11	23	3	48.881	151.124	15.0	6.3	mw

```
In [3]: # 読み込んだデータの列ごとの統計量表示
df.describe()
```

Out[3]:

	year	month	day	time_h	time_m	latitude	longitude	depth	mag
count	260708.000000	260708.000000	260708.000000	260708.000000	260708.000000	260708.000000	260708.000000	260075.000000	260708.00000
mean	1998.989751	6.502869	15.732329	11.594734	29.422638	4.435789	42.687531	71.882408	4.92502
std	15.977575	3.481261	8.736928	6.888519	17.321767	29.185393	121.365554	117.590837	0.46826
min	1920.000000	1.000000	1.000000	0.000000	0.000000	-84.133000	-179.999000	-4.000000	4.50000
25%	1988.000000	3.000000	8.000000	6.000000	14.000000	-17.286000	-71.688000	10.820000	4.60000
50%	2003.000000	6.000000	16.000000	12.000000	29.000000	0.589000	101.062000	33.000000	4.80000
75%	2012.000000	10.000000	23.000000	18.000000	44.000000	29.631250	142.579150	63.000000	5.10000
max	2021.000000	12.000000	31.000000	23.000000	59.000000	87.386000	180.000000	700.900000	9.50000

magType（Magnitude Types）については、USGS のサイトに説明があるので省略する。本報告書ではツールの使い方習得や機械学習に触れてみることに重点を置いているため、magType を考慮せずデータを使うことにするが、どの割合で magType が混ざっているかということだけ最初に確認しておくことにする。Pandas の groupby メソッドを使用。

```
In [4]: df.groupby('magType').size()

Out[4]: magType
Mb            4
Md            6
Mi            1
Ml            6
fa           36
lg           11
m           302
ma            2
mb       200670
mb_lg         1
mblg         19
mc            7
md          796
mh           89
ml         2921
mlg           3
mlr          14
ms         4568
ms_20         2
mw        17884
mwb        3148
mwc       20394
mwp           7
mwr        2516
mww        7201
uk           99
dtype: int64
```

次に、magType 列を取り除いた df2 を定義、さらに欠損値を含む行も取り除いた df3 を定義。最後に、欠損値が正常に取り除けたか、df2 と df3 の行数をチェックした。csv を編集し、別ファイルとして保存しなくても、必要なデータのみをデータフレームとして定義して使えるのが便利。

In [5]:
```
# magTypeの列を取り除いたdf2を定義し、先頭行を表示
df2=df.drop(columns=['magType'])
df2.head()
```

Out[5]:

	year	month	day	time_h	time_m	latitude	longitude	depth	mag
0	1920	1	1	2	35	33.200	-116.700	NaN	5.0
1	1920	1	24	7	9	48.800	-123.000	NaN	5.5
2	1920	2	2	11	22	-4.967	152.017	35.0	7.8
3	1920	2	8	5	24	-35.809	109.537	15.0	6.4
4	1920	2	10	22	7	18.619	-67.339	15.0	6.4

In [6]:
```
# NaNを含む行を取り除いたdf3を定義
df3=df2.dropna(how='any')
```

In [7]:
```
# df2の行数表示
len(df2)
```

Out[7]: 260708

In [8]:
```
# df3の行数表示
len(df3)
```

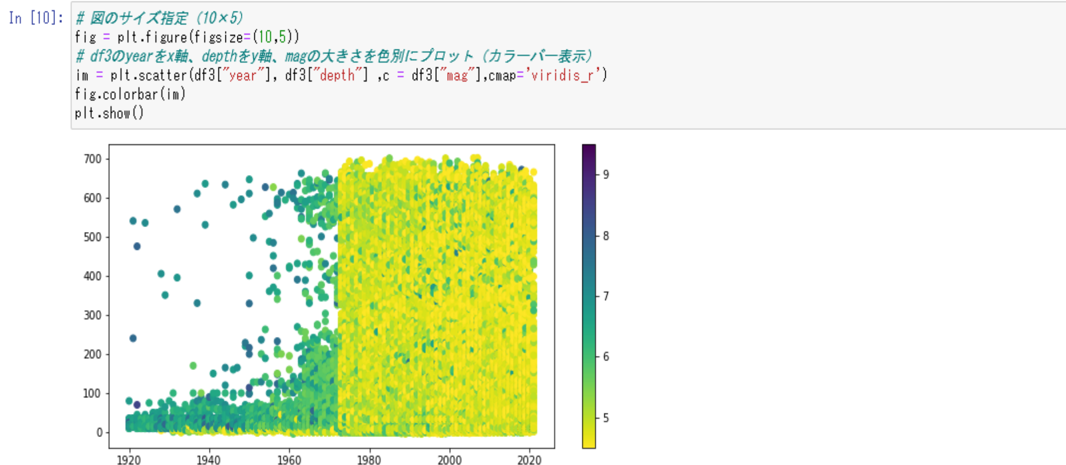
Out[8]: 260075

以降は、データを可視化することで、特徴を捉えることを試みる。matplotlib をインポートし、inline モードを指定しておくと Jupyter Notebook 内にグラフを表示できる。

In [9]:
```
# matplotlib をインポート（inlineモードで出力）
%matplotlib inline
import matplotlib.pyplot as plt
```

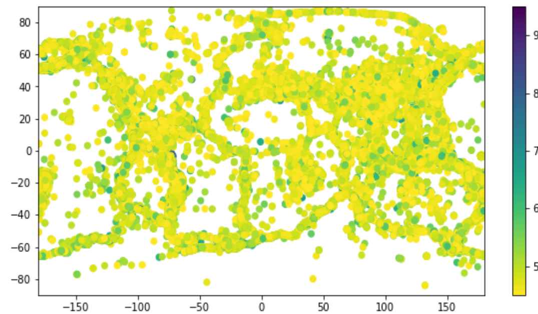
まず、ダウンロードした全データ（ただし欠損値は除く）を使い、「地震の発生年」を x 軸に「震源の深さ」を y 軸に、マグニチュードの大きさで色分けしてプロットしてみる。1970 年代（詳しく確認すると 1973 年が境目）から、マグニチュードの小さい地震の検出数が増えたことが一目瞭然である。

In [10]:
```
# 図のサイズ指定（10×5）
fig = plt.figure(figsize=(10,5))
# df3のyearをx軸、depthをy軸、magの大きさを色別にプロット（カラーバー表示）
im = plt.scatter(df3["year"], df3["depth"] ,c = df3["mag"],cmap='viridis_r')
fig.colorbar(im)
plt.show()
```



次に、「経度」「緯度」をそれぞれ x 軸 y 軸としてプロットする。M4.5 以上の地震が発生していない（あるいは検出されていない）地域が空白部分となる。

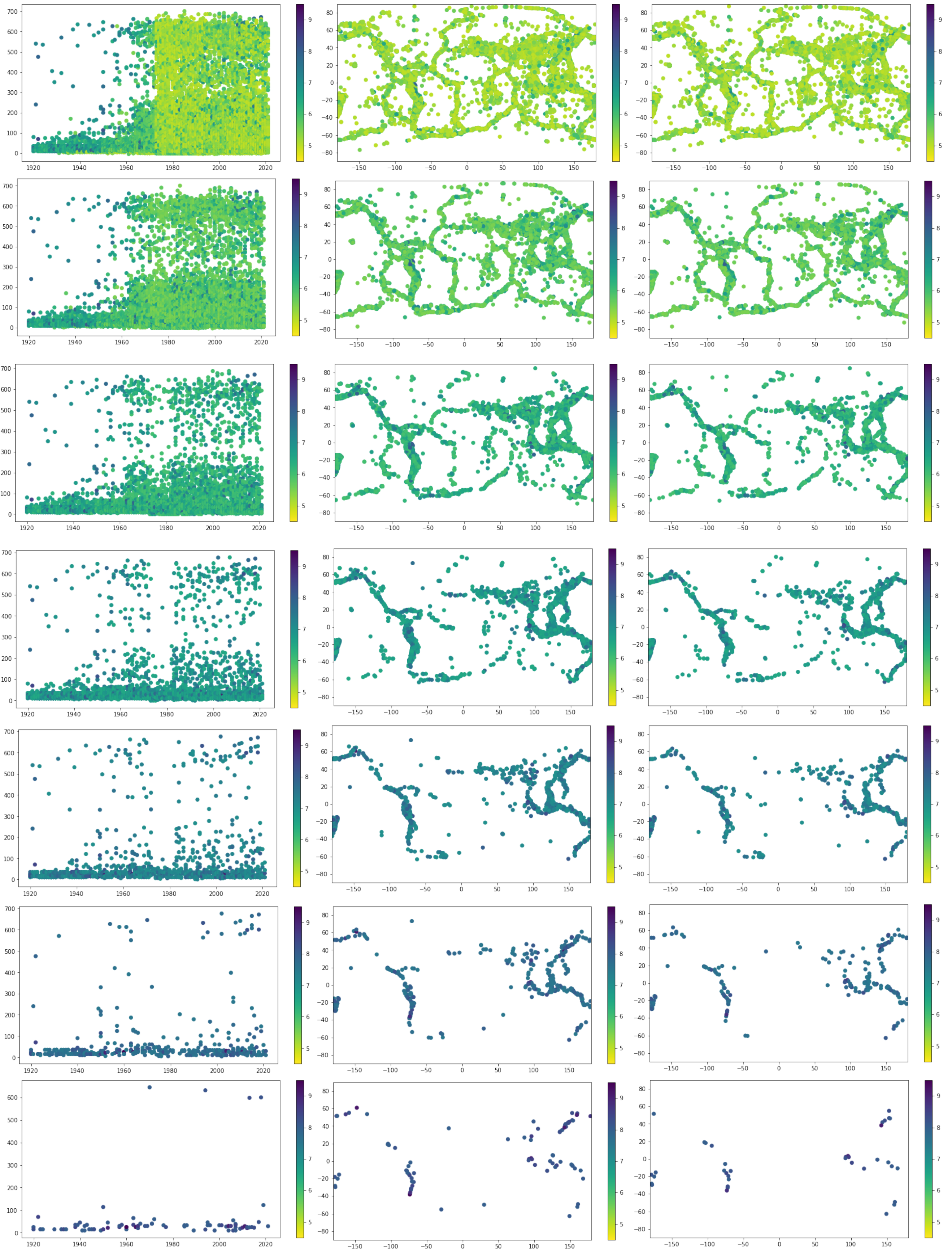
In [11]:
```
fig = plt.figure(figsize=(10,5))
# df3のlongitudeをx軸、latitudeをy軸、magの大きさを色別にプロット（カラーバー表示）
# x軸latitudeの最小値-180, 最大値180 指定
# y軸latitudeの最小値-90, 最大値90 指定

im = plt.scatter (df3["longitude"], df3["latitude"],c = df3["mag"] ,cmap='viridis_r')
fig.colorbar(im)
plt.xlim(-180, 180)
plt.ylim(-90, 90)
plt.show()
```



次ページの画像は、一番上段がM5以上の地震について、左から順に「x軸：地震の発生年，y軸：震源の深さ」、「x軸；経度，y軸：緯度　※1973年以前のデータも使用」、「x軸；経度，y軸：緯度　※1973年以降のデータのみ使用」をマグニチュードの大きさで色分けしてプロットしたものである。2段目以降は、プロットするデータをM5.5以上、M6以上・・・と、0.5刻みの条件で上げていき、最下段はM8以上の地震についてプロットしたものになる。

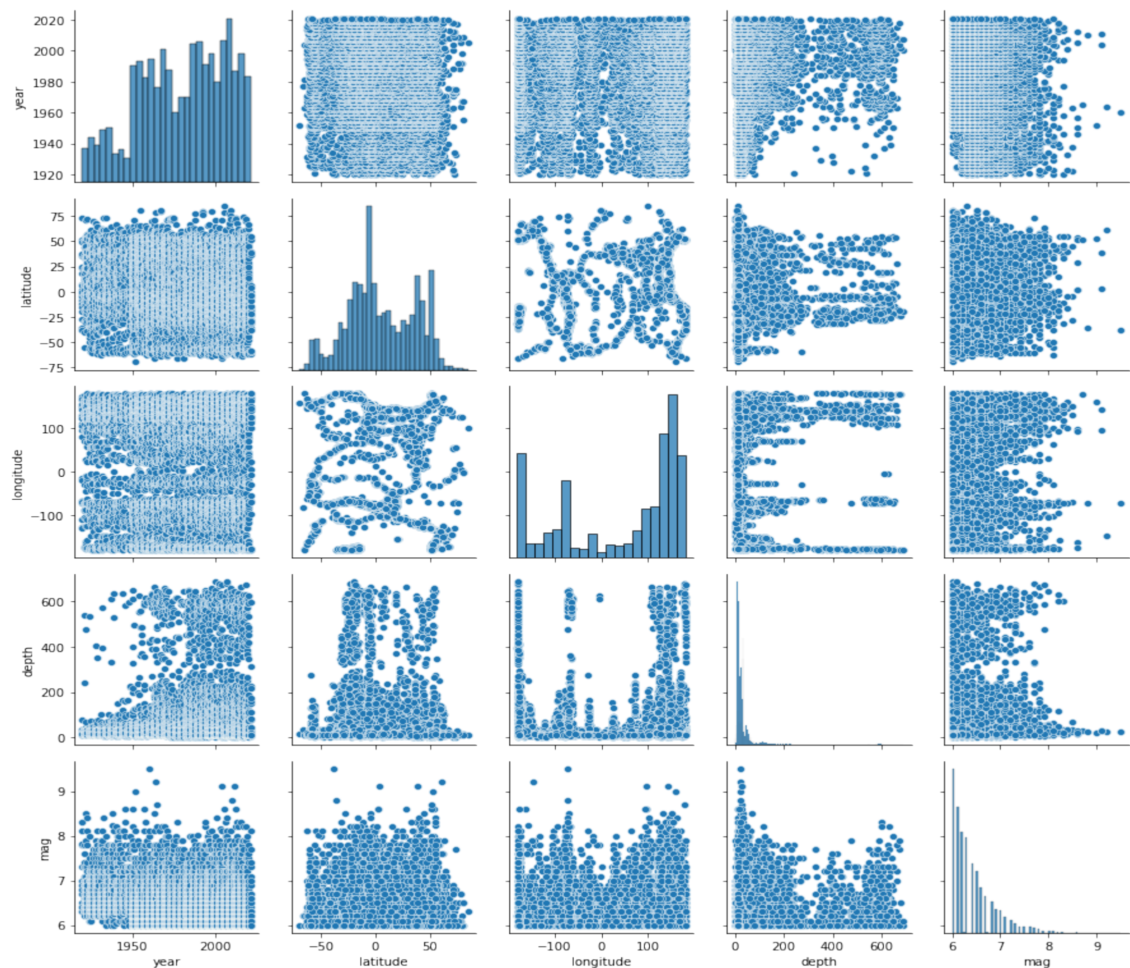
データ数が大きく異なる1973年以前のデータも含めるか省いたほうがよいか確認するため地震発生場所（x軸；経度　y軸：緯度）の図は2パターン作成した。傾向が変わるわけではないので、1973年以前のデータも含めて大きな問題はなさそうだ。

M5.5以上の地震の発生年と震源の深さ分布において（2段目左端）、深さ300kmあたりに空白の横ラインが見え、地震が発生しにくいことがわかる。また、M6.0以上の地震の発生年と震源の深さ分布（3段目）において、1970年〜1980年あたりにやや空白部分が目立ちだし、M6.5以上（4段目）ではその空白部分がはっきりと見てとれる。



今度は seaborn ライブラリを使い、プロットするデータを M4.5 以上、M5 以上・・・と、0.5 刻みの条件で上げていきながら、それぞれのペアプロット図を作成し眺めてみた。M6 以上の散布図において、プロット点が分離していく様子が見て取れたので一例として以下に示しておく。ペアプロット図の一番下の右端、マグニチュードの大きさと発生回数を示すグラフは「グーテンベルグ・リヒター測」のとおりであった。

```
# インポート
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# csvファイル読み込み
df = pd.read_csv('./19200101-20210415_2.csv')
# NaNを含む行を取り除いてdfを再定義
df = df.dropna(how='any')
# mag6以上の行のみを抽出したdfを再定義
df = df.query('mag >= 6')

# year, latitude, longitude, depth, mag
df = df.iloc[:,[0,5,6,7,8]]

# ペアプロット図を作成
sns.pairplot(df)
plt.tight_layout()
plt.show()
```



③～⑤モデルを決め、訓練し、評価する

以下、回帰と分類の 2 パターンを試した。

【回帰】

グーテンベルグ・リヒター測を確かめるべく mag ごとに地震の発生回数を集計してみる。データをそのまま使うと mag の値が細かく分けて集計され、発生回数にバラつきが生じて結果が悪くなってしまった。そこで、mag の値を round 関数で丸めてから mag ごとに地震の発生回数を集計することにした。

```
# pandas インポート
import pandas as pd
# numpy インポート
import numpy as np
# csvファイル読み込み
df = pd.read_csv('./19200101-20210415_2.csv')
# NaNを含む行を取り除いてdfを再定義
df = df.dropna(how='any')
# df内容確認
df
```

	year	month	day	time_h	time_m	latitude	longitude	depth	mag	magType
2	1920	2	2	11	22	-4.9670	152.0170	35.00	7.8	mw
3	1920	2	8	5	24	-35.8090	109.5370	15.00	6.4	mw
4	1920	2	10	22	7	18.6190	-67.3390	15.00	6.4	mw
5	1920	3	17	18	37	1.5750	95.5550	15.00	6.3	mw
6	1920	3	20	18	31	-35.8480	-110.7260	10.00	7.0	mw
...	...	...	...	...	...	...	...	...	...	...
260703	2021	4	15	16	42	51.2274	100.4112	17.13	4.6	mb

```
# df['mag']の値をround関数で丸める
df['mag']=df['mag'].apply(lambda x:round(x,0))
# df内容確認
df
```

	year	month	day	time_h	time_m	latitude	longitude	depth	mag	magType
2	1920	2	2	11	22	-4.9670	152.0170	35.00	8.0	mw
3	1920	2	8	5	24	-35.8090	109.5370	15.00	6.0	mw
4	1920	2	10	22	7	18.6190	-67.3390	15.00	6.0	mw
5	1920	3	17	18	37	1.5750	95.5550	15.00	6.0	mw
6	1920	3	20	18	31	-35.8480	-110.7260	10.00	7.0	mw
...	...	...	...	...	...	...	...	...	...	...
260703	2021	4	15	16	42	51.2274	100.4112	17.13	5.0	mb

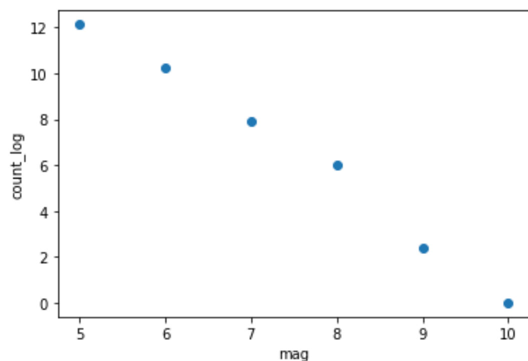
ダウンロードしたデータは mag が 4.5 以上であり、さらに mag の値を丸めたので、mag=4.0 のデータは使わないことにした。また、発生回数の対数を count_log として df に入れた。

```
# mag(x)の値ごとに地震発生回数(y)をカウント
x=df.groupby('mag').size().index
y=df.groupby('mag').size()
# 発生回数(y)の対数をylogとして取得
ylog=np.log(y)
# df再定義
df=pd.DataFrame(data=[x,y,ylog],index=['mag','count','count_log']).T
# df再定義 (mag=4.0 を除外)
df=df[df['mag'] != 4.0]
# df内容確認
df
```

	mag	count	count_log
1	5.0	184701.0	12.126494
2	6.0	27527.0	10.222923
3	7.0	2726.0	7.910591
4	8.0	400.0	5.991465
5	9.0	11.0	2.397895
6	10.0	1.0	0.000000

比例関係が成り立つであろうことは上記ペアプロット図（の、一番下の段の右端）より予想できるが、基本にのっとって mag と count_log の関係を先に可視化しておく。

```
# matplotlibをインポート（inlineモードで出力）
%matplotlib inline
import matplotlib.pyplot as plt

plt.scatter(X, Y)
plt.xlabel('mag')
plt.ylabel('count_log')
plt.show()
```



比例関係となることが確認できたので、訓練データとテストデータを 8：2 の比で分割し、線形回帰モデルの一つである LinearRegression を使って、訓練データで学習。

```python
# XにDF['mag']をYにdf['cout_log']を入れる
X=df.iloc[:, [0]]
Y=df.iloc[:, [2]]
# 訓練データとテストデータを分割するメソッドのインポート
from sklearn.model_selection import train_test_split
# 線形回帰モデルのインポート
from sklearn.linear_model import LinearRegression
# 訓練データ・テストデータを8:2の比でランダムに分割
X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size = 0.8, test_size = 0.2, random_state = 0)
# 線形回帰モデルの呼び出し
model = LinearRegression()
# モデルを訓練
model.fit(X_train, y_train)
```

```
LinearRegression()
```

```python
# 説明変数の係数を表示
print('coefficient = ', model.coef_[0])
# 回帰直線の切片を表示
print('intercept = ', model.intercept_)
```

```
coefficient =  [-2.36886547]
intercept =  [24.26675229]
```

```python
# 訓練データのスコア表示
print(model.score(X_train,y_train))
# テストデータのスコア表示
print(model.score(X_test,y_test))
```

```
0.9846599917152754
0.987688020248685
```

学習した結果

　グラフの傾き[-2.36886547]

　切片[24.26675229]

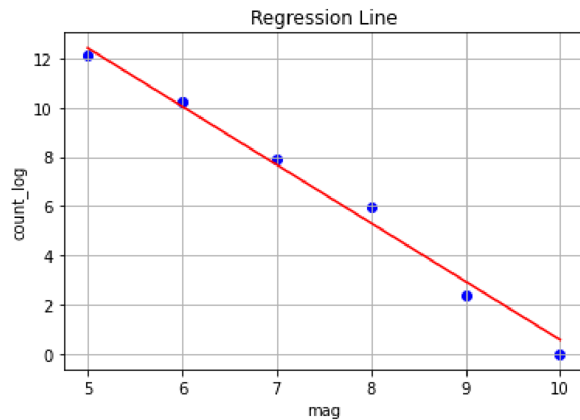
　訓練データのスコア[0.9846599917152754]

　テストデータのスコア[0.987688020248685]

であることがわかった。

　回帰直線を赤で、データを青でプロットしてみる。グラフからもよく近似できていることがわかる。

```python
# matplotlib をインポート（inlineモードで出力）
%matplotlib inline
import matplotlib.pyplot as plt

# 散布図をプロット
plt.scatter(X, Y, color = 'blue')
# 回帰直線をプロット
plt.plot(X, model.predict(X), color = 'red')

plt.title('Regression Line')
plt.xlabel('mag')
plt.ylabel('count_log')

plt.grid()

plt.show()
```



参考までに、データを整形せずに（ただし欠損値は除いた）学習させた結果も以下に示しておく。初めて自分で用意したデータで線形回帰した結果がこれであった。

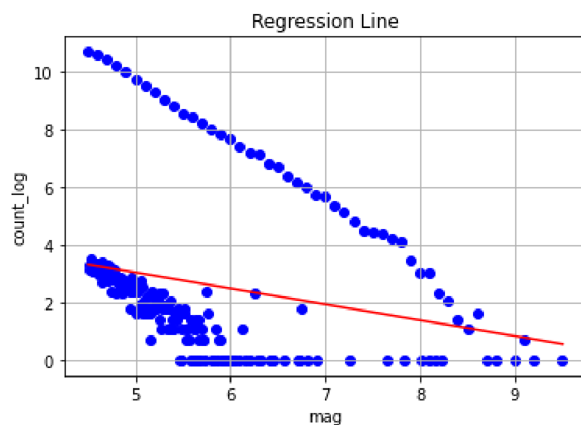
結果

グラフの傾き[-0.55242447]

切片[5.80427143]

訓練データのスコア[0.05878475105336156]

テストデータのスコア[-0.03512513282475216]



このあと、mag の値を丸めさらに mag=4 は使わないことにしたのだが、大雑把にやりすぎた感があるので、もう少しだけ別のパターンでも試してみる。

まずは、ダウンロードした全データのうち、magType が mb※のデータのみを使用。mag のデータに手を加えることなく、LinearRegression モデルで学習。

※「②データを観察」のところで magType=mb のデータが一番多いことを確認していたため

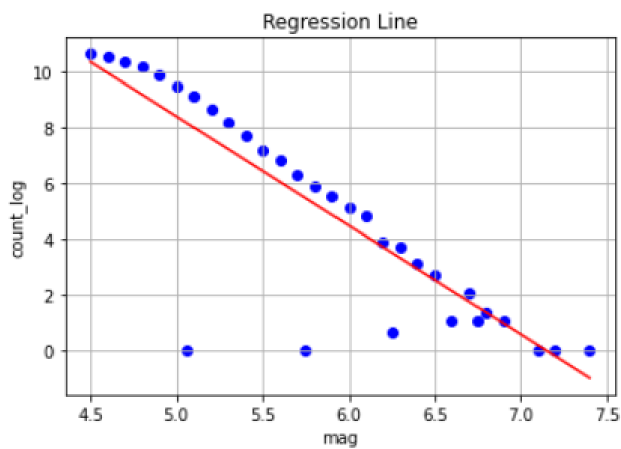
結果

グラフの傾き[-3.91172315]

切片[27.98097757]

訓練データのスコア[0.7579291805524109]

テストデータのスコア[0.619153949214442]



次に、magType が mb のデータの mag の値を小数第一位までに丸めて学習。

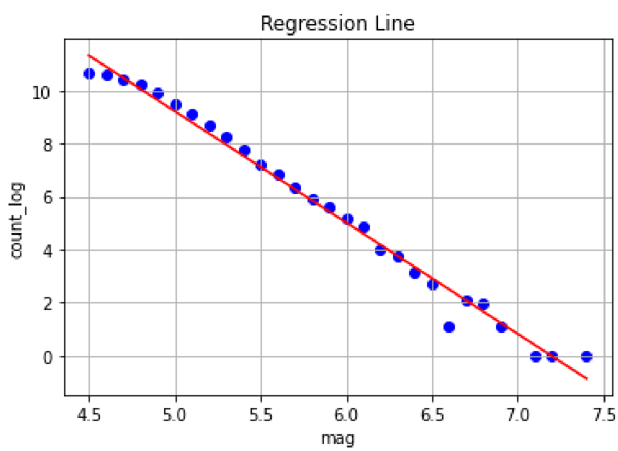
<u>結果</u>

グラフの傾き[-4.20461174]

切片[30.24195427]

訓練データのスコア[0.986058875975727]

テストデータのスコア[0.9968839099052986]



【分類】

「②データを観察」のところで、報告書に載せた以外にも、色々な図をプロットして眺めてみたが、どのように分類するのがよいか全く見当がつかなかった。そこで、mag と magType 以外の情報を配列として説明変数に入れ込んでしまって、mag を目的変数として分類できないかと考えた。多層パーセプトロンの分類器 MLPClassifier を使用。

```
import pandas as pd
import numpy as np

# csvファイル読み込み
df = pd.read_csv('./19200101-20210415_2.csv')
# NaNを含む行を取り除いたdfを定義
df=df.dropna(how='any')

# df['mag']の値をround関数で丸めて整数にしておく
df['mag']=df['mag'].apply(lambda x:round(x,0))
df['mag']=df['mag'].apply(lambda x:int(x))

# 丸めた結果のdf['mag']の値のうち5以上9以下を使う
df=df.query('5<= mag <= 9')

# Xにdfのyear,month,day,time_h,time_m,latitude,longitude,depth（0～7列目）の値、Yにmag(8列目)の値を入れる
X = np.array(df.iloc[:, :8].values)
Y = df['mag']

# 訓練データとテストデータを分割するメソッドのインポート
from sklearn.model_selection import train_test_split
# 訓練データ・テストデータを8:2の比でランダムに分割
X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size=0.8, test_size=0.2, random_state=2)

# MLPClassifierの呼び出し
from sklearn.neural_network import MLPClassifier
# モデルを定義（学習を繰り返す最大回数max_iter=1000）
clf = MLPClassifier(max_iter=1000, random_state=2)
# モデルを訓練
clf.fit(X_train, y_train)
```

```
MLPClassifier(max_iter=1000, random_state=2)
```

```
df
```

	year	month	day	time_h	time_m	latitude	longitude	depth	mag	magType
2	1920	2	2	11	22	-4.9670	152.0170	35.00	8	mw
3	1920	2	8	5	24	-35.8090	109.5370	15.00	6	mw
4	1920	2	10	22	7	18.6190	-67.3390	15.00	6	mw
5	1920	3	17	18	37	1.5750	95.5550	15.00	6	mw
6	1920	3	20	18	31	-35.8480	-110.7260	10.00	7	mw
...	...	...	...	...	...	...	...	...	...	...
260703	2021	4	15	16	42	51.2274	100.4112	17.13	5	mb
260704	2021	4	15	18	10	14.8686	-94.1175	40.32	5	mb
260705	2021	4	15	19	16	-8.6533	-79.8503	35.00	5	mb
260706	2021	4	15	22	26	32.4880	78.7179	10.00	5	mb
260707	2021	4	15	23	57	46.1291	152.9228	10.00	5	mb

215365 rows × 10 columns

```
X
```

```
array([[ 1.920000e+03,  2.000000e+00,  2.000000e+00, ..., -4.967000e+00,
         1.520170e+02,  3.500000e+01],
       [ 1.920000e+03,  2.000000e+00,  8.000000e+00, ..., -3.580900e+01,
         1.095370e+02,  1.500000e+01],
       [ 1.920000e+03,  2.000000e+00,  1.000000e+01, ...,  1.861900e+01,
        -6.733900e+01,  1.500000e+01],
       ...,
       [ 2.021000e+03,  4.000000e+00,  1.500000e+01, ..., -8.653300e+00,
        -7.985030e+01,  3.500000e+01],
       [ 2.021000e+03,  4.000000e+00,  1.500000e+01, ...,  3.248800e+01,
         7.871790e+01,  1.000000e+01],
       [ 2.021000e+03,  4.000000e+00,  1.500000e+01, ...,  4.612910e+01,
         1.529228e+02,  1.000000e+01]])
```

結果は以下のとおり。訓練データ、テストデータともにスコア（正解率）は 85%を超えており、良い結果のように見える。しかし、テストデータの予測値をマグニチュード別に集計すると、43073 個のデータのうち 43072 個を「5」と予測し、1 個を「7」と予測したことがわかる。この学習結果は、使用したデータ 215365 個のうち「5」が 184701 個、使用したデータに対する「5」のデータの割合が「0.857618462」であることに影響を受けている。このような偏りの大きいデータは不均衡データと呼ばれ、取り扱いには注意が必要となる。スコアの表示だけでなく、再現率や適合率も要確認。この場合、mag＝5 のクラスのみ再現率「1」で、その他のクラスは「0」であることがわかる。

```
# 訓練データのスコア表示
print(clf.score(X_train,y_train))
# テストデータのスコア表示
print(clf.score(X_test,y_test))
```

```
0.8575673856011887
0.8577995496018387
```

```
clf.predict(X_test)
```

```
array([5, 5, 5, ..., 5, 5, 5], dtype=int64)
```

```
X_predict = clf.predict(X_test)
df_Xpredict = pd.DataFrame(X_predict, columns=['mag'])
df_Xpredict
```

	mag
0	5
1	5
2	5
3	5
4	5
...	...
43068	5
43069	5
43070	5
43071	5
43072	5

43073 rows × 1 columns

```
# テストデータの予測値をマグニチュード別に集計
df_Xpredict.groupby('mag').size()
```

```
mag
5    43072
7        1
dtype: int64
```

```
# テストデータの正解ラベルをマグニチュード別に集計
df_ytest = pd.DataFrame(y_test, columns=['mag'])
df_ytest.groupby('mag').size()
```

```
mag
5    36948
6     5474
7      567
8       82
9        2
dtype: int64
```

```
# 使用するデータをマグニチュード別に集計
df.groupby('mag').size()
```

```
mag
5    184701
6     27527
7      2726
8       400
9        11
dtype: int64
```

```
# テストデータの予測値をpredに入れる
pred = clf.predict(X_test)
```

```
# 正解率を表示
from sklearn.metrics import accuracy_score
accuracy_score(y_test,pred)
```

```
0.8577995496018387
```

```
# 再現率を表示
from sklearn.metrics import recall_score
recall_score(y_test, pred, average=None)
```

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

```
# 適合率を表示
from sklearn.metrics import precision_score
precision_score(y_test, pred, average=None)
```

```
C:¥Users¥sawada¥anaconda3¥lib¥site-packages¥sklearn¥metrics¥_classification.py:1248: UndefinedMetricWarning: Precision is ill-defined and be
ing set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
array([0.85781947, 0.        , 0.        , 0.        , 0.        ])
```

不均衡データの分類では、クラスごとのデータ数をそろえるアンダーサンプリングやオーバーサンプリング、少数派のサンプルに対して重みづけをする、異常検知問題として扱う、などいくつか方法があるようだ。今回は一番試しやすそうな SMOTE を使ってオーバーサンプリングしたのち学習させてみる。合わせて、すべての特徴量がだいたい同じスケールになるようにデータの前処理も実施した※。

※StandardScaler を用いれば自動でできるが、ここでは手動で変換

```
import pandas as pd
import numpy as np

# csvファイル読み込み
df = pd.read_csv('./19200101-20210415_2.csv')
# NaNを含む行を取り除いたdfを定義
df=df.dropna(how='any')

# df['mag']の値をround関数で丸めて整数にしておく
df['mag']=df['mag'].apply(lambda x:round(x,0))
df['mag']=df['mag'].apply(lambda x:int(x))

# 丸めた結果のdf['mag']の値のうち5以上9以下を使う
df=df.query('5<= mag <= 9')

# XにdfのYear,month,day,time_h,time_m,latitude,longitude,depth（0～7列目）の値、Yにmag(8列目)の値を入れる
X = np.array(df.iloc[:, :8].values)
Y = df['mag']

# 訓練データとテストデータを分割するメソッドのインポート
from sklearn.model_selection import train_test_split
# 訓練データ・テストデータを8:2の比でランダムに分割
X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size=0.8, test_size=0.2, random_state=2)

# 訓練データの特徴量ごとの平均値を算出
mean_on_train = X_train.mean(axis=0)
# 訓練データの特徴量ごとの標準偏差を算出
std_on_train = X_train.std(axis=0)
# 平均を引き、標準偏差の逆数でスケール変換（mean=0, std=1となる）
X_train_scaled =(X_train - mean_on_train) / std_on_train
# テストデータも同様に変換
X_test_scaled = (X_test - mean_on_train) / std_on_train
```

```
# SMOTEをインポートしてオーバーサンプリング
from imblearn.over_sampling import SMOTE
sm = SMOTE()
X_resampled, y_resampled = sm.fit_resample(X_train_scaled, y_train)
```

```
# オーバーサンプリング後のy_resampledをマグニチュード別で集計
df_yresampled=pd.DataFrame(y_resampled)
df_yresampled.groupby('mag').size()
```

```
mag
5    147753
6    147753
7    147753
8    147753
9    147753
dtype: int64
```

```
# MLPClassifierの呼び出し
from sklearn.neural_network import MLPClassifier
# モデルを定義(学習の反復の最大回数max_iter=1000)
clf = MLPClassifier(max_iter=1000, random_state=2)
# モデルを訓練
clf.fit(X_resampled, y_resampled)
```

```
MLPClassifier(max_iter=1000, random_state=2)
```

```
# 訓練データ(resample済み)のスコア表示
print(clf.score(X_resampled,y_resampled))
# テストデータ(前処理済み)のスコア表示
print(clf.score(X_test_scaled,y_test))
```

```
0.6835488895792302
0.5371114155039l2
```

```
# テストデータの正解ラベルをマグニチュード別で集計
df_ytest=pd.DataFrame(y_test)
df_ytest.groupby('mag').size()
```

```
mag
5    36948
6     5474
7      567
8       82
9        2
dtype: int64
```

```
# テストデータの予測値をpredに入れる
pred = clf.predict(X_test_scaled)
```

```
# テストデータの予測値をマグニチュード別で集計
df_Xpredict = pd.DataFrame(pred,columns=['mag'])
df_Xpredict.groupby('mag').size()
```

```
mag
5    22820
6    12243
7     5122
8     2838
9       50
dtype: int64
```

```
# 正解率を表示
from sklearn.metrics import accuracy_score
accuracy_score(y_test,pred)
```

```
0.5371114155039l2
```

```
# 再現率を表示
from sklearn.metrics import recall_score
recall_score(y_test, pred, average=None)
```

```
array([0.56441485, 0.37431494, 0.39153439, 0.12195122, 0.        ])
```

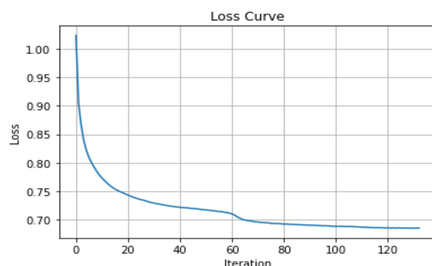
```
# 適合率を表示
from sklearn.metrics import precision_score
pre_score = precision_score(y_test, pred, average=None)
print(pre_score)
```

```
[0.9138475  0.16736094 0.04334244 0.00352361 0.        ]
```

損失値（訓練データと予測データとのズレ）は clf.loss_curve_ に保存されるので、これをそのままプロットすることができる。

```
%matplotlib inline
import matplotlib.pyplot as plt
plt.plot(clf.loss_curve_)
plt.title('Loss Curve')
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.grid(True)
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



get_params にて、デフォルトのパラメータを確認。活性化関数（activation）を指定していなかったので、relu を使って学習した結果であることがわかる。

```
clf.get_params()
```

```
{'activation': 'relu',
 'alpha': 0.0001,
 'batch_size': 'auto',
 'beta_1': 0.9,
 'beta_2': 0.999,
 'early_stopping': False,
 'epsilon': 1e-08,
 'hidden_layer_sizes': (100,),
 'learning_rate': 'constant',
 'learning_rate_init': 0.001,
 'max_fun': 15000,
 'max_iter': 1000,
 'momentum': 0.9,
 'n_iter_no_change': 10,
 'nesterovs_momentum': True,
 'power_t': 0.5,
 'random_state': 2,
 'shuffle': True,
 'solver': 'adam',
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': False,
 'warm_start': False}
```

パラメータ activation に logistic、identity、tanh を指定してそれぞれ学習させ、次に、パラメータ hidden_layer_sizes を(100,100)※に変更して学習（活性化関数は指定せず＝relu）させた。結果を以下の表にまとめておく。

※2 層で 100 ニューロンずつ配置

mag	正解ラベル（mag別集計）	予測値（mag別集計）					再現率					適合率				
		relu	logistic	identity	tanh	(100,100)	relu	logistic	identity	tanh	(100,100)	relu	logistic	identity	tanh	(100,100)
5	36948	22820	29146	22162	28054	32570	0.5644	0.7170	0.5438	0.6889	0.7988	0.9138	0.9089	0.9067	0.9073	0.9062
6	5474	12243	7589	7701	6949	7968	0.3743	0.2859	0.1829	0.2526	0.3767	0.1674	0.2062	0.1300	0.1990	0.2588
7	567	5122	3985	2313	5382	1997	0.3915	0.3721	0.1270	0.4127	0.1711	0.0433	0.0529	0.0311	0.0435	0.0486
8	82	2838	2294	4938	2635	514	0.1220	0.1220	0.3049	0.1098	0.0244	0.0035	0.0044	0.0051	0.0034	0.0039
9	2	50	59	5959	53	24	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

mag	正解ラベル（mag別集計）	正解数 正解ラベル（集計数）×再現率 ≒予測値（集計数）×適合率				
		relu	logistic	identity	tanh	(100,100)
5	36948	20854	26490	20094	25452	29515
6	5474	2049	1565	1001	1383	2062
7	567	222	211	72	234	97
8	82	10	10	25	9	2
9	2	0	0	0	0	0

正解率				
relu	logistic	identity	tanh	(100,100)
0.5371	0.6565	0.4920	0.6287	0.7354

最後に、サポートベクタマシンの分類器である SVC を使っての学習を試してみた。学習を反復する最大回数 max_iter を 10000 にしても下記のエラーが出た。MinMaxScaler を使ったデータの前処理も試してみたが同様のエラーとなった。

ConvergenceWarning: Solver terminated early (max_iter=10000). Consider pre-processing your data with StandardScaler or MinMaxScaler. warnings.warn('Solver terminated early (max_iter=%i).'

以下は、max_iter を指定せずに実行。結果が表示されるまで何時間もかかった。

```
import pandas as pd
import numpy as np

# csvファイル読み込み
df = pd.read_csv('./19200101-20210415_2.csv')
# NaNを含む行を取り除いたdfを定義
df=df.dropna(how='any')

# df['mag']の値をround関数で丸めて整数にしておく
df['mag']=df['mag'].apply(lambda x:round(x,0))
df['mag']=df['mag'].apply(lambda x:int(x))

# 丸めた結果のdf['mag']の値のうち5以上9以下を使う
df=df.query('5<= mag <= 9')

# Xにdfのyear,month,day,time_h,time_m,latitude,longitude,depth（0～7列目）の値、Yにmag(8列目)の値を入れる
X = np.array(df.iloc[:, :8].values)
Y = df['mag']

# 訓練データとテストデータを分割するメソッドのインポート
from sklearn.model_selection import train_test_split
# 訓練データ・テストデータを8:2の比でランダムに分割
X_train, X_test, y_train, y_test = train_test_split(X, Y, train_size=0.8, test_size=0.2, random_state=2)

# 訓練データの特徴量ごとの平均値を算出
mean_on_train = X_train.mean(axis=0)
# 訓練データの特徴量ごとの標準偏差を算出
std_on_train = X_train.std(axis=0)
# 平均を引き、標準偏差の逆数でスケール変換（mean=0, std=1となる）
X_train_scaled =(X_train - mean_on_train) / std_on_train
# テストデータも同様に変換
X_test_scaled = (X_test - mean_on_train) / std_on_train
```

```
# SMOTEをインポートしてオーバーサンプリング
from imblearn.over_sampling import SMOTE
sm = SMOTE()
X_resampled, y_resampled = sm.fit_resample(X_train_scaled, y_train)
```

```
# svmの呼び出し
from sklearn import svm
# モデルを定義(学習の反復の最大回数指定なし)
clf = svm.SVC()
# モデルを訓練
clf.fit(X_resampled, y_resampled)
```

```
SVC()
```

```
# 訓練データ(resample済み)のスコア表示
print(clf.score(X_resampled,y_resampled))
# テストデータ(前処理済み)のスコア表示
print(clf.score(X_test_scaled,y_test))
```

```
0.7032534026381867
0.6571866366401226
```

```
# テストデータの正解ラベルをマグニチュード別で集計
df_ytest=pd.DataFrame(y_test)
df_ytest.groupby('mag').size()
```

```
mag
5    36948
6     5474
7      567
8       82
9        2
dtype: int64
```

```
# テストデータの予測値をpredに入れる
pred = clf.predict(X_test_scaled)
```

```
# テストデータの予測値をマグニチュード別で集計
df_Xpredict = pd.DataFrame(pred,columns=['mag'])
df_Xpredict.groupby('mag').size()
```

```
mag
5    29418
6     6287
7     3481
8     3698
9      189
dtype: int64
```

```
# 正解率を表示
from sklearn.metrics import accuracy_score
accuracy_score(y_test,pred)
```

```
0.6571866366401226
```

```
# 再現率を表示
from sklearn.metrics import recall_score
recall_score(y_test, pred, average=None)
```

```
array([0.72141929, 0.26835952, 0.30687831, 0.1097561 , 0.        ])
```

```
# 適合率を表示
from sklearn.metrics import precision_score
pre_score = precision_score(y_test, pred, average=None)
print(pre_score)
```

```
[0.90607791 0.23365675 0.04998564 0.00243375 0.        ]
```

おわりに

ひとまず機械学習に触れてみるという目標は達成できたということにしておきたい。わからないことや知らないことが多すぎる状態で、とりあえずやってみるというのは、やはりなかなか無謀であったことを実感した。業務の合間、期限内に技術室報告を書きあげるということに意識がいくと、じっくり学んだり検証したりということも途中からおろそかになってしまった。もっと基礎を学んでから出直したい。

機械学習やディープラーニングを使った研究は、今後さらに増えていくのではないかと思う。技術職員として少しでもお役に立てるよう、ツールの使い方、モデルやパラメータに関する知識、データの取り扱いのノウハウについて学んでいきたい。

参考文献など：

・Andreas C. Muller、Sarah Guido　著、中田 秀基　訳
「Python ではじめる機械学習
　scikit-learn で学ぶ特徴量エンジニアリングと機械学習の基礎」
オライリージャパン

・斎藤　康毅　著
「ゼロから作る Deep Learning
　Python で学ぶディープラーニングの理論と実装」
オライリージャパン

・USGS website
Search Earthquake Catalog
https://earthquake.usgs.gov/earthquakes/search/
Magnitude Types
https://www.usgs.gov/natural-hazards/earthquake-hazards/science/magnitude-types?qtscience_center_objects=0#qt-science_center_objects

・Python 公式ドキュメント
https://docs.python.org/ja/3/

・スタビジ「不均衡データを SMOTE で解消！」
https://toukei-lab.com/imbalance-data-smote