FORECASTING EARTHQUAKE AFTERSHOCK LOCATIONS WITH AI-ASSISTED SCIENCE

A Project Reort

Submitted in the partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

INTERNET-OF-THINGS

Submitted by:

Rishabh Anand

Shefali Yadav

Abhishek Singh

University Roll Number

19BCS4525

19BCS4524

19BCS4508

Under the Supervision of:

Mr. Nikhil Aggarwal

CHANDIGARH UNIVERSITY, GHARUAN, MOHALI - 140413, PUNJAB

Table of Contents

ABSTRACT

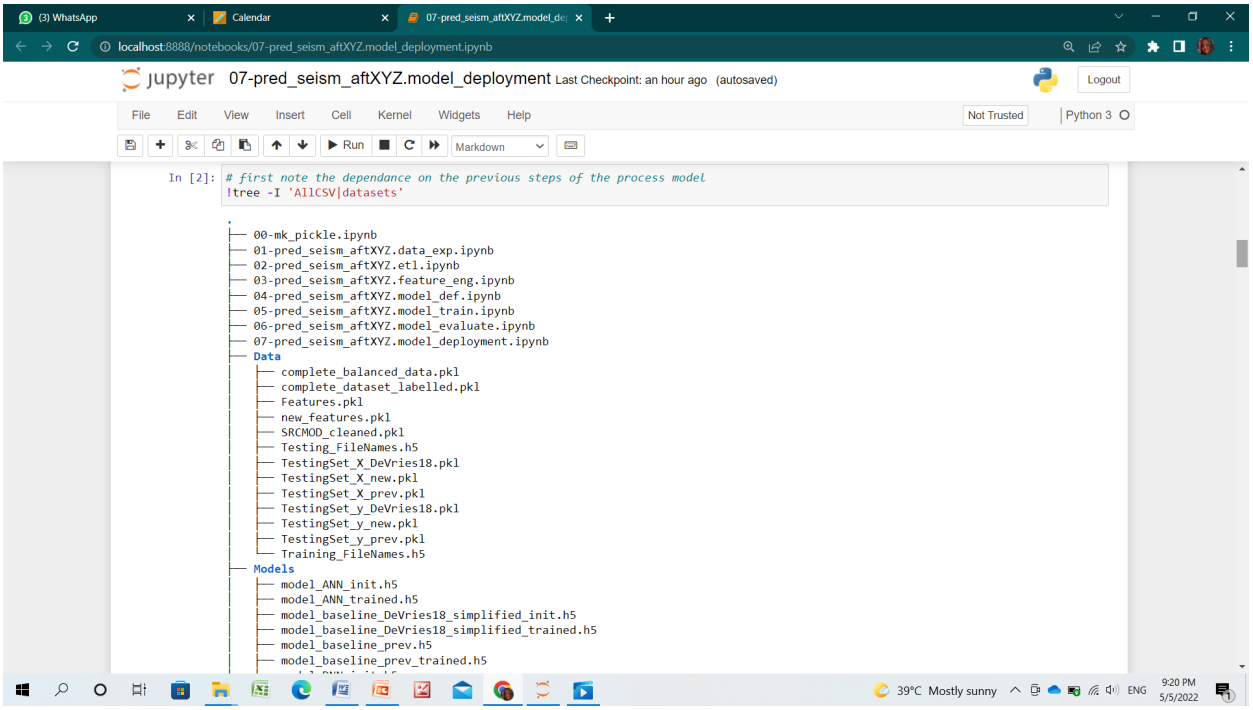
In a world, divided by fear, of losing your loved ones, of losing your loved belongings,of losing your life, we hope to come up with a solution that should keep you and your dreams safe. Because that's what EarthQuake's take away... Even after the major tremor, what hurts more is the AfterShocks that follow. These are produced by the stress that was caused by the earthquake. This project gives us a second chance at saving lives by using Artificial Intelligence to determine where the next tremor is going to be. So that you can move, and get to a safer place. Methods like Columnb's Stress Criterion are being used in current times to explain the spatial distributions of AfterShocks, but as the advent of science & technology is improving, we hope to introduce Machine Learning models that can find an undiscovered pattern which will be helpful in predicting the fair locations of AfterShocks.

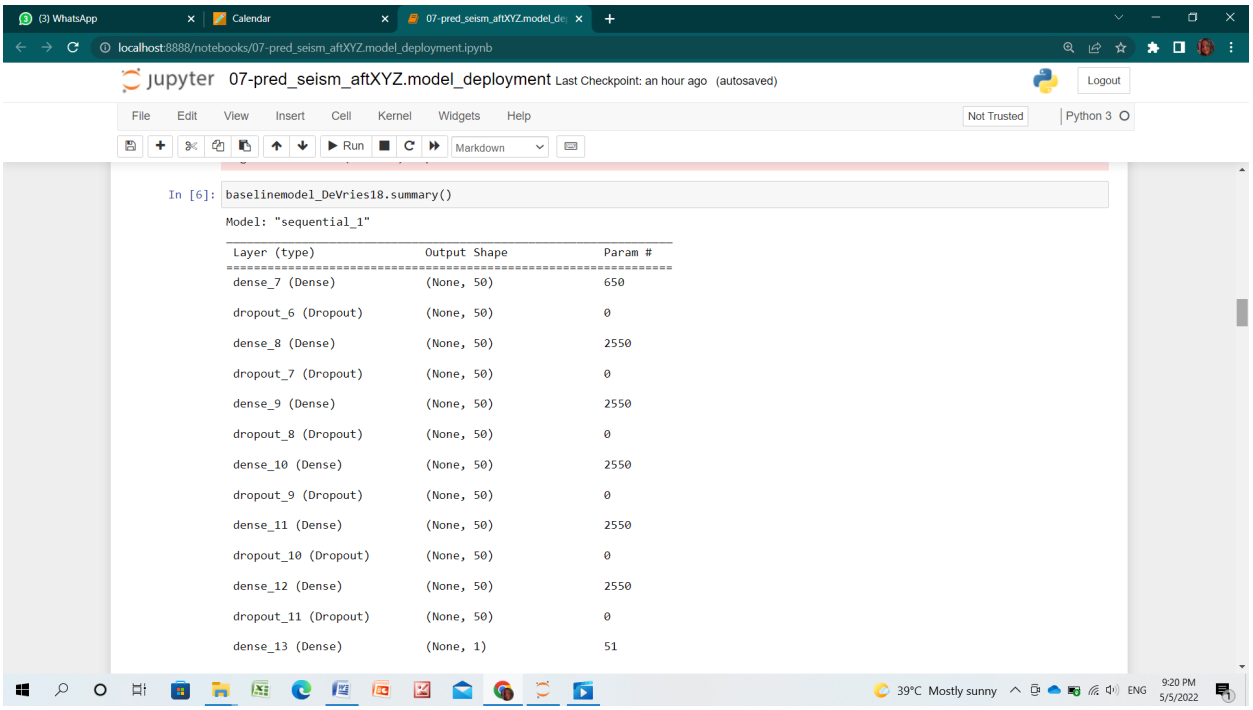
Once we have our predictions, it is very important to display them in a good manner so that Uncle Bob can understand them and move himself to safety. We have created a React web-app just for this purpose so that it is easily acessible to people and move

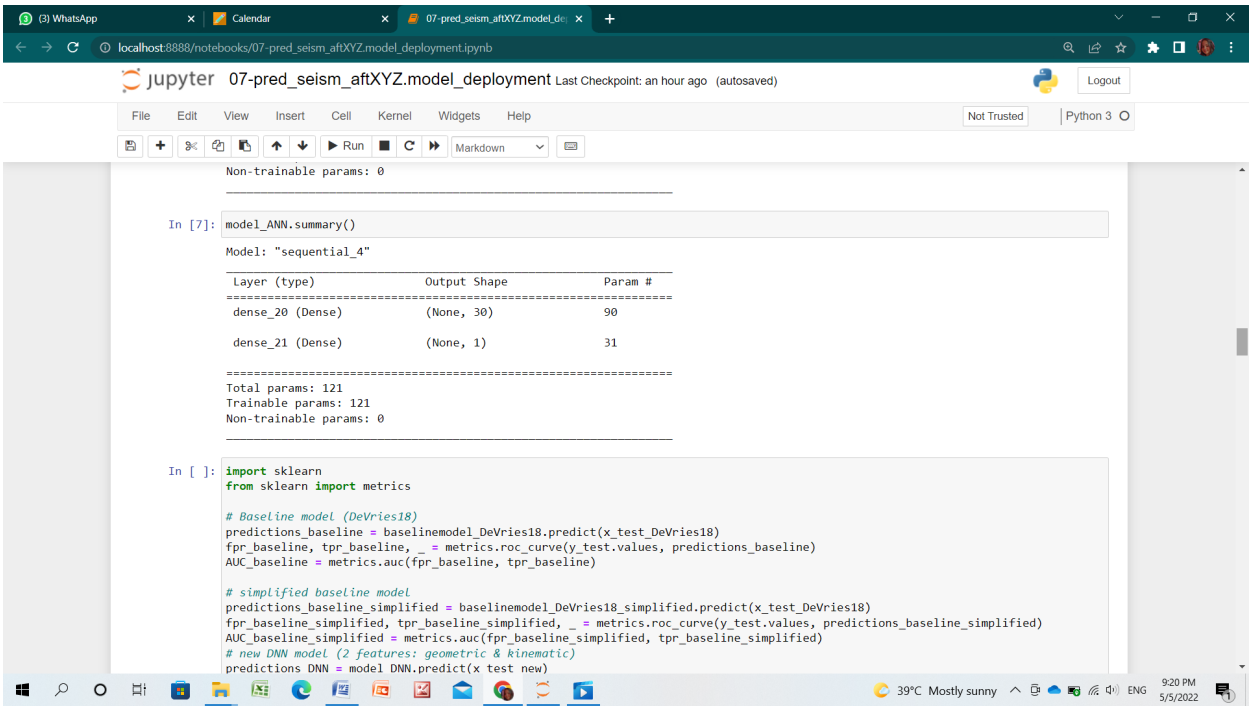
them from harm's way. Thereby, reducing the damage to both people and resources,

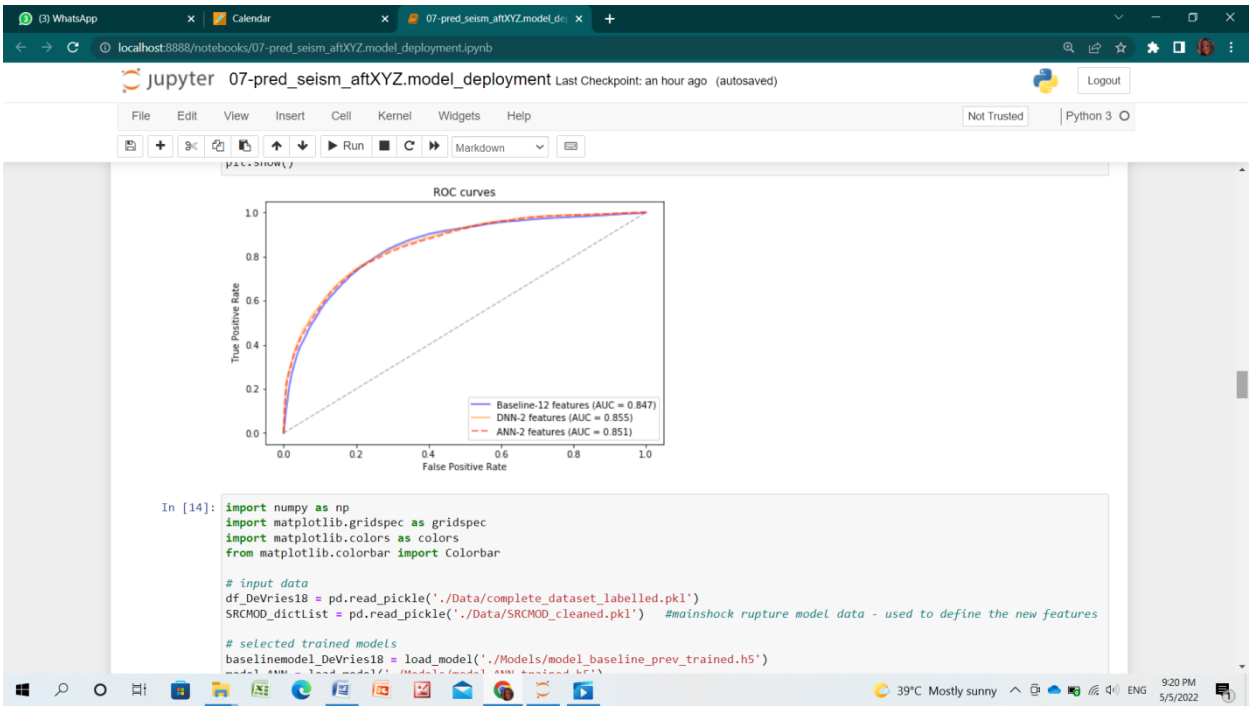
thus, making this world a better place.

LIST OF PHOTOGRAPHS

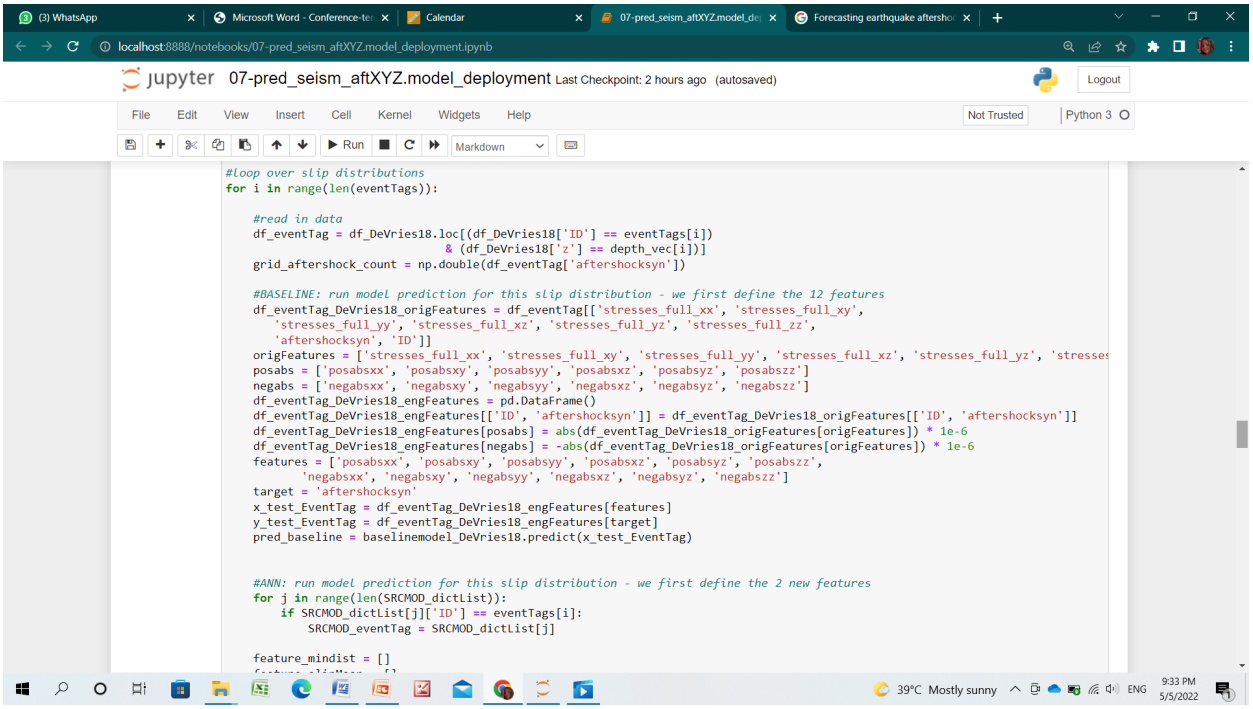


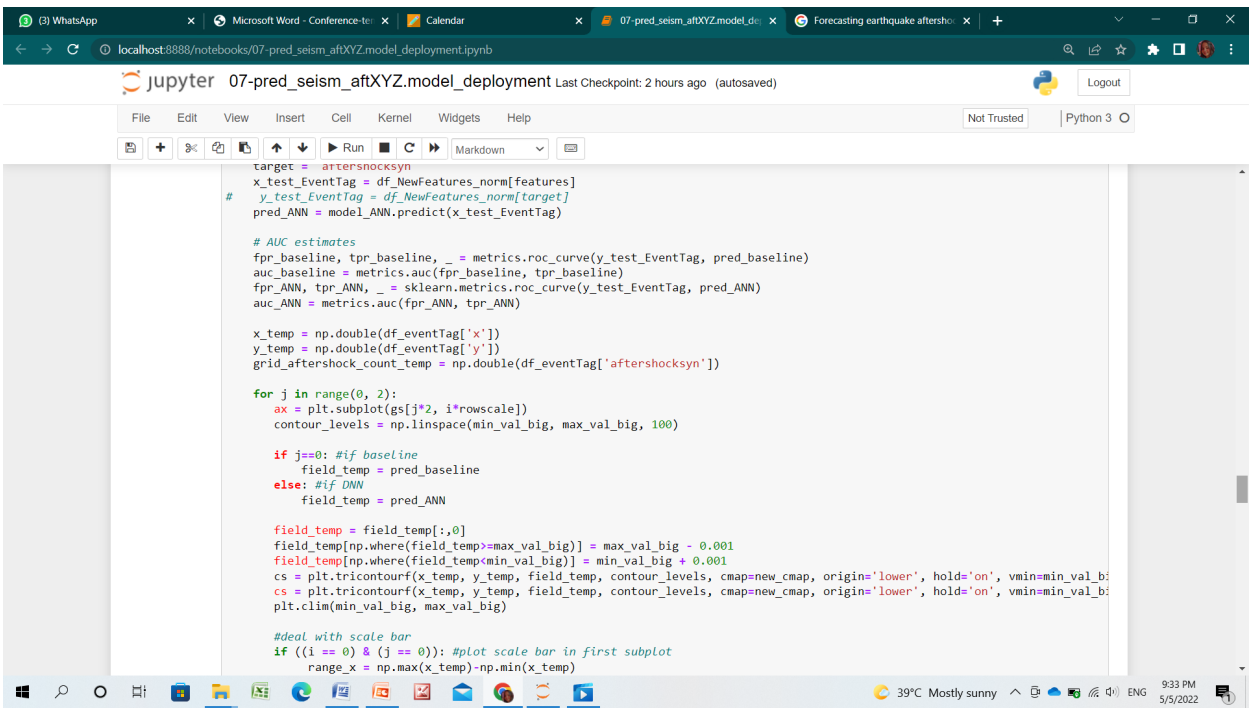

(3) WhatsApp
Calendar
07-pred_seism_aftXYZ.model_de
localhost:8888/notebooks/07-pred_seism_aftXYZ.model_deployment.ipynb
jupyter 07-pred_seism_aftXYZ.model_deployment Last Checkpoint: an hour ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Markdown
In [6]:
baselinemodel_DeVries18.summary()
Model: "sequential_1"
Layer (type) Output Shape Param #
dense_7 (Dense) (None, 50) 650
dropout_6 (Dropout) (None, 50) 0
dense_8 (Dense) (None, 50) 2550
dropout_7 (Dropout) (None, 50) 0
dense_9 (Dense) (None, 50) 2550
dropout_8 (Dropout) (None, 50) 0
dense_10 (Dense) (None, 50) 2550
dropout_9 (Dropout) (None, 50) 0
dense_11 (Dense) (None, 50) 2550
dropout_10 (Dropout) (None, 50) 0
dense_12 (Dense) (None, 50) 2550
dropout_11 (Dropout) (None, 50) 0
dense_13 (Dense) (None, 1) 51
39°C Mostly sunny
ENG
9:20 PM
5/5/2022

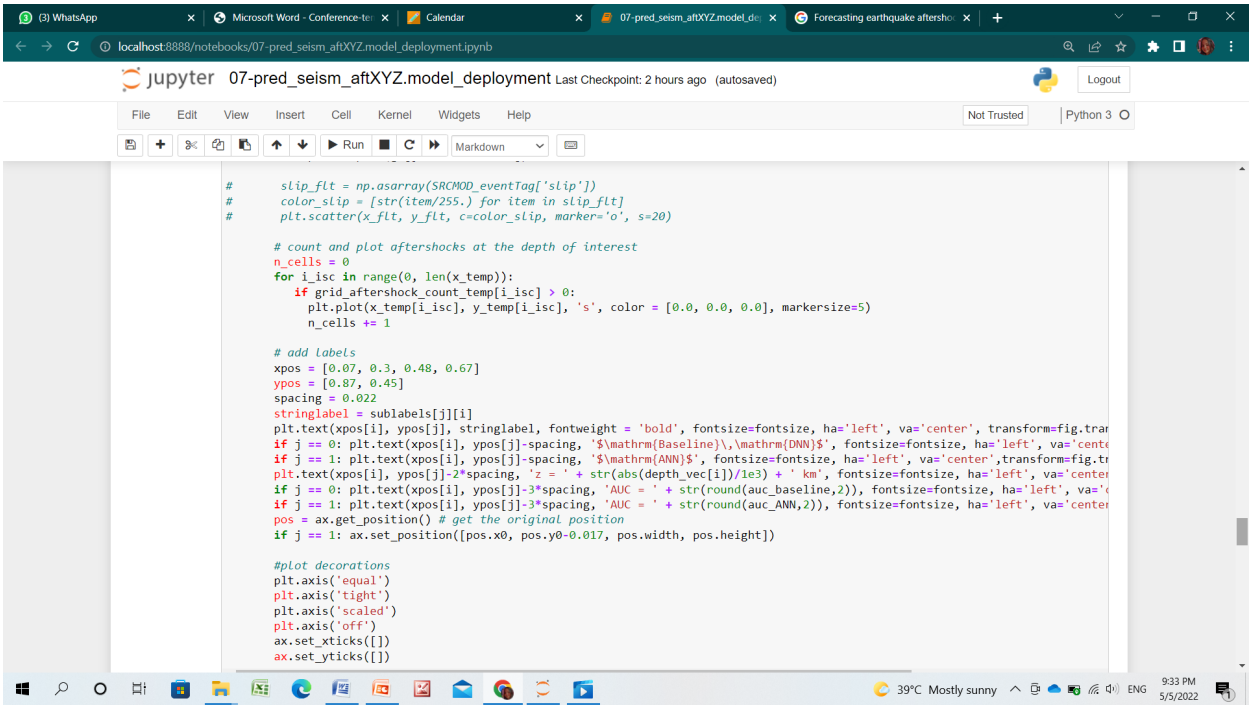

(3) WhatsApp
Calendar
07-pred_seism_aftXYZ.model_de
localhost:8888/notebooks/07-pred_seism_aftXYZ.model_deployment.ipynb
jupyter 07-pred_seism_aftXYZ.model_deployment Last Checkpoint: an hour ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Markdown
Non-trainable params: 0
In [7]:
model_ANN.summary()
Model: "sequential_4"
Layer (type) Output Shape Param #
dense_20 (Dense) (None, 30) 90
dense_21 (Dense) (None, 1) 31
Total params: 121
Trainable params: 121
Non-trainable params: 0
In [ ]:
import sklearn
from sklearn import metrics
# Baseline model (DeVries18)
predictions_baseline = baselinemodel_DeVries18.predict(x_test_DeVries18)
fpr_baseline, tpr_baseline, _ = metrics.roc_curve(y_test.values, predictions_baseline)
AUC_baseline = metrics.auc(fpr_baseline, tpr_baseline)
# simplified baseline model
predictions_baseline_simplified = baselinemodel_DeVries18_simplified.predict(x_test_DeVries18)
fpr_baseline_simplified, tpr_baseline_simplified, _ = metrics.roc_curve(y_test.values, predictions_baseline_simplified)
AUC_baseline_simplified = metrics.auc(fpr_baseline_simplified, tpr_baseline_simplified)
# new DNN model (2 features: geometric & kinematic)
predictions DNN = model DNN.predict(x test new)
39°C Mostly sunny
ENG
9:20 PM
5/5/2022

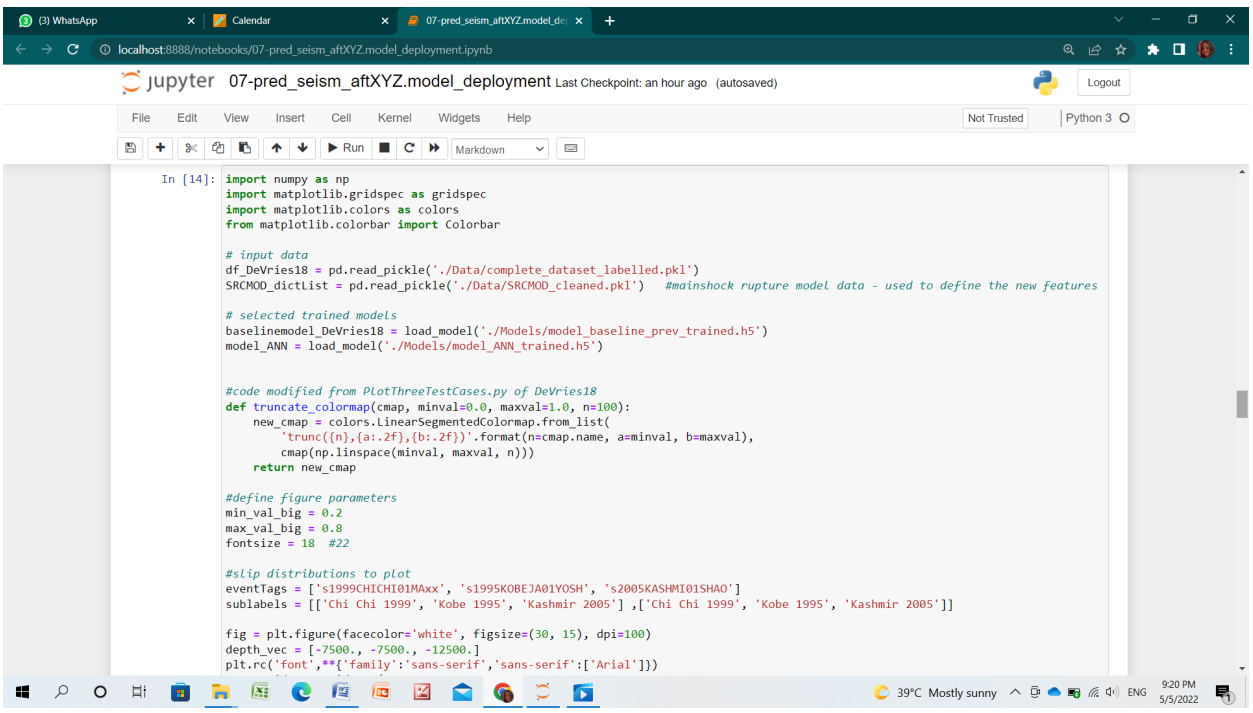


In [14]:
```
import numpy as np
import matplotlib.gridspec as gridspec
import matplotlib.colors as colors
from matplotlib.colorbar import Colorbar

# input data
df_DeVries18 = pd.read_pickle('./Data/complete_dataset_labelled.pkl')
SRCMOD_dictList = pd.read_pickle('./Data/SRCMOD_cleaned.pkl')   #mainshock rupture model data - used to define the new features

# selected trained models
baselinemodel_DeVries18 = load_model('./Models/model_baseline_prev_trained.h5')
```


(3) WhatsApp
Microsoft Word - Conference-te
Calendar
07-pred_seism_aftXYZ.model_de
Forecasting earthquake aftersho
localhost:8888/notebooks/07-pred_seism_aftXYZ.model_deployment.ipynb
jupyter 07-pred_seism_aftXYZ.model_deployment Last Checkpoint: 2 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Markdown
#loop over slip distributions
for i in range(len(eventTags)):
#read in data
df_eventTag = df_DeVries18.loc[(df_DeVries18['ID'] == eventTags[i])
& (df_DeVries18['z'] == depth_vec[i])]
grid_aftershock_count = np.double(df_eventTag['aftershocksyn'])
#BASELINE: run model prediction for this slip distribution - we first define the 12 features
df_eventTag_DeVries18_origFeatures = df_eventTag[['stresses_full_xx', 'stresses_full_xy',
'stresses_full_yy', 'stresses_full_xz', 'stresses_full_yz', 'stresses_full_zz',
'aftershocksyn', 'ID']]
origFeatures = ['stresses_full_xx', 'stresses_full_xy', 'stresses_full_yy', 'stresses_full_xz', 'stresses_full_yz', 'stresses
posabs = ['posabsxx', 'posabsxy', 'posabsyy', 'posabsxz', 'posabsyz', 'posabszz']
negabs = ['negabsxx', 'negabsxy', 'negabsyy', 'negabsxz', 'negabsyz', 'negabszz']
df_eventTag_DeVries18_engFeatures = pd.DataFrame()
df_eventTag_DeVries18_engFeatures[['ID', 'aftershocksyn']] = df_eventTag_DeVries18_origFeatures[['ID', 'aftershocksyn']]
df_eventTag_DeVries18_engFeatures[posabs] = abs(df_eventTag_DeVries18_origFeatures[origFeatures]) * 1e-6
df_eventTag_DeVries18_engFeatures[negabs] = -abs(df_eventTag_DeVries18_origFeatures[origFeatures]) * 1e-6
features = ['posabsxx', 'posabsxy', 'posabsyy', 'posabsxz', 'posabsyz', 'posabszz',
'negabsxx', 'negabsxy', 'negabsyy', 'negabsxz', 'negabsyz', 'negabszz']
target = 'aftershocksyn'
x_test_EventTag = df_eventTag_DeVries18_engFeatures[features]
y_test_EventTag = df_eventTag_DeVries18_engFeatures[target]
pred_baseline = baselinemodel_DeVries18.predict(x_test_EventTag)
#ANN: run model prediction for this slip distribution - we first define the 2 new features
for j in range(len(SRCMOD_dictList)):
if SRCMOD_dictList[j]['ID'] == eventTags[i]:
SRCMOD_eventTag = SRCMOD_dictList[j]
feature_mindist = []
39°C Mostly sunny
ENG
9:33 PM
5/5/2022


(3) WhatsApp
Microsoft Word - Conference-te
Calendar
07-pred_seism_aftXYZ.model_de
Forecasting earthquake aftersho
localhost:8888/notebooks/07-pred_seism_aftXYZ.model_deployment.ipynb
jupyter 07-pred_seism_aftXYZ.model_deployment Last Checkpoint: 2 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Markdown
target = 'aftershocksyn'
x_test_EventTag = df_NewFeatures_norm[features]
# y_test_EventTag = df_NewFeatures_norm[target]
pred_ANN = model_ANN.predict(x_test_EventTag)
# AUC estimates
fpr_baseline, tpr_baseline, _ = metrics.roc_curve(y_test_EventTag, pred_baseline)
auc_baseline = metrics.auc(fpr_baseline, tpr_baseline)
fpr_ANN, tpr_ANN, _ = sklearn.metrics.roc_curve(y_test_EventTag, pred_ANN)
auc_ANN = metrics.auc(fpr_ANN, tpr_ANN)
x_temp = np.double(df_eventTag['x'])
y_temp = np.double(df_eventTag['y'])
grid_aftershock_count_temp = np.double(df_eventTag['aftershocksyn'])
for j in range(0, 2):
ax = plt.subplot(gs[j*2, i*rowscale])
contour_levels = np.linspace(min_val_big, max_val_big, 100)
if j==0: #if baseline
field_temp = pred_baseline
else: #if DNN
field_temp = pred_ANN
field_temp = field_temp[:,0]
field_temp[np.where(field_temp>=max_val_big)] = max_val_big - 0.001
field_temp[np.where(field_temp<min_val_big)] = min_val_big + 0.001
cs = plt.tricontourf(x_temp, y_temp, field_temp, contour_levels, cmap=new_cmap, origin='lower', hold='on', vmin=min_val_bi
cs = plt.tricontourf(x_temp, y_temp, field_temp, contour_levels, cmap=new_cmap, origin='lower', hold='on', vmin=min_val_bi
plt.clim(min_val_big, max_val_big)
#deal with scale bar
if ((i == 0) & (j == 0)): #plot scale bar in first subplot
range_x = np.max(x_temp)-np.min(x_temp)
39°C Mostly sunny
ENG
9:33 PM
5/5/2022


(3) WhatsApp
Microsoft Word - Conference-te
Calendar
07-pred_seism_aftXYZ.model_de
Forecasting earthquake aftersho
localhost:8888/notebooks/07-pred_seism_aftXYZ.model_deployment.ipynb
jupyter 07-pred_seism_aftXYZ.model_deployment Last Checkpoint: 2 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Markdown
# slip_flt = np.asarray(SRCMOD_eventTag['slip'])
# color_slip = [str(item/255.) for item in slip_flt]
# plt.scatter(x_flt, y_flt, c=color_slip, marker='o', s=20)
# count and plot aftershocks at the depth of interest
n_cells = 0
for i_isc in range(0, len(x_temp)):
if grid_aftershock_count_temp[i_isc] > 0:
plt.plot(x_temp[i_isc], y_temp[i_isc], 's', color = [0.0, 0.0, 0.0], markersize=5)
n_cells += 1
# add labels
xpos = [0.07, 0.3, 0.48, 0.67]
ypos = [0.87, 0.45]
spacing = 0.022
stringlabel = sublabels[j][i]
plt.text(xpos[i], ypos[j], stringlabel, fontweight = 'bold', fontsize=fontsize, ha='left', va='center', transform=fig.tra
if j == 0: plt.text(xpos[i], ypos[j]-spacing, '$\mathrm{Baseline}\,\mathrm{DNN}$', fontsize=fontsize, ha='left', va='cente
if j == 1: plt.text(xpos[i], ypos[j]-spacing, 'ANN', fontsize=fontsize, ha='left', va='center',transform=fig.t
plt.text(xpos[i], ypos[j]-2*spacing, 'z = ' + str(abs(depth_vec[i])/1e3) + ' km', fontsize=fontsize, ha='left', va='center
if j == 0: plt.text(xpos[i], ypos[j]-3*spacing, 'AUC = ' + str(round(auc_baseline,2)), fontsize=fontsize, ha='left', va='
if j == 1: plt.text(xpos[i], ypos[j]-3*spacing, 'AUC = ' + str(round(auc_ANN,2)), fontsize=fontsize, ha='left', va='center
pos = ax.get_position() # get the original position
if j == 1: ax.set_position([pos.x0, pos.y0-0.017, pos.width, pos.height])
#plot decorations
plt.axis('equal')
plt.axis('tight')
plt.axis('scaled')
plt.axis('off')
ax.set_xticks([])
ax.set_yticks([])
39°C Mostly sunny
ENG
9:33 PM
5/5/2022


(3) WhatsApp
Calendar
07-pred_seism_aftXYZ.model_de
localhost:8888/notebooks/07-pred_seism_aftXYZ.model_deployment.ipynb
jupyter 07-pred_seism_aftXYZ.model_deployment Last Checkpoint: an hour ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Not Trusted
Python 3
Run
Markdown
In [14]:
import numpy as np
import matplotlib.gridspec as gridspec
import matplotlib.colors as colors
from matplotlib.colorbar import Colorbar
# input data
df_DeVries18 = pd.read_pickle('./Data/complete_dataset_labelled.pkl')
SRCMOD_dictList = pd.read_pickle('./Data/SRCMOD_cleaned.pkl') #mainshock rupture model data - used to define the new features
# selected trained models
baselinemodel_DeVries18 = load_model('./Models/model_baseline_prev_trained.h5')
model_ANN = load_model('./Models/model_ANN_trained.h5')
#code modified from PlotThreeTestCases.py of DeVries18
def truncate_colormap(cmap, minval=0.0, maxval=1.0, n=100):
new_cmap = colors.LinearSegmentedColormap.from_list(
'trunc({n},{a:.2f},{b:.2f})'.format(n=cmap.name, a=minval, b=maxval),
cmap(np.linspace(minval, maxval, n)))
return new_cmap
#define figure parameters
min_val_big = 0.2
max_val_big = 0.8
fontsize = 18 #22
#slip distributions to plot
eventTags = ['s1999CHICHI01MAxx', 's1995KOBEJA01YOSH', 's2005KASHMI01SHAO']
sublabels = [['Chi Chi 1999', 'Kobe 1995', 'Kashmir 2005'] ,['Chi Chi 1999', 'Kobe 1995', 'Kashmir 2005']]
fig = plt.figure(facecolor='white', figsize=(30, 15), dpi=100)
depth_vec = [-7500., -7500., -12500.]
plt.rc('font',**{'family':'sans-serif','sans-serif':['Arial']})
39°C Mostly sunny
ENG
9:20 PM
5/5/2022

(3) WhatsApp | Calendar | 07-pred_seism_aftXYZ.model_de

localhost:8888/notebooks/07-pred_seism_aftXYZ.model_deployment.ipynb

jupyter 07-pred_seism_aftXYZ.model_deployment Last Checkpoint: an hour ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

Run Markdown

```
                        height_ratios=[30, 1, 30, 1]
                        )
rowscale = 2
cmap = plt.get_cmap('Reds')
new_cmap = truncate_colormap(cmap, 0.0, 0.75)

#Loop over slip distributions
for i in range(len(eventTags)):

    #read in data
    df_eventTag = df_DeVries18.loc[(df_DeVries18['ID'] == eventTags[i])
                                   & (df_DeVries18['z'] == depth_vec[i])]
    grid_aftershock_count = np.double(df_eventTag['aftershocksyn'])

    #BASELINE: run model prediction for this slip distribution - we first define the 12 features
    df_eventTag_DeVries18_origFeatures = df_eventTag[['stresses_full_xx', 'stresses_full_xy',
       'stresses_full_yy', 'stresses_full_xz', 'stresses_full_yz', 'stresses_full_zz',
       'aftershocksyn', 'ID']]
    origFeatures = ['stresses_full_xx', 'stresses_full_xy', 'stresses_full_yy', 'stresses_full_xz', 'stresses_full_yz', 'stresses
    posabs = ['posabsxx', 'posabsxy', 'posabsyy', 'posabsxz', 'posabsyz', 'posabszz']
    negabs = ['negabsxx', 'negabsxy', 'negabsyy', 'negabsxz', 'negabsyz', 'negabszz']
    df_eventTag_DeVries18_engFeatures = pd.DataFrame()
    df_eventTag_DeVries18_engFeatures[['ID', 'aftershocksyn']] = df_eventTag_DeVries18_origFeatures[['ID', 'aftershocksyn']]
    df_eventTag_DeVries18_engFeatures[posabs] = abs(df_eventTag_DeVries18_origFeatures[origFeatures]) * 1e-6
    df_eventTag_DeVries18_engFeatures[negabs] = -abs(df_eventTag_DeVries18_origFeatures[origFeatures]) * 1e-6
    features = ['posabsxx', 'posabsxy', 'posabsyy', 'posabsxz', 'posabsyz', 'posabszz',
           'negabsxx', 'negabsxy', 'negabsyy', 'negabsxz', 'negabsyz', 'negabszz']
    target = 'aftershocksyn'
    x_test_EventTag = df_eventTag_DeVries18_engFeatures[features]
    y_test_EventTag = df_eventTag_DeVries18_engFeatures[target]
    pred_baseline = baselinemodel_DeVries18.predict(x_test_EventTag)


    #ANN: run model prediction for this slip distribution - we first define the 2 new features
```

39°C Mostly sunny ENG 9:20 PM 5/5/2022

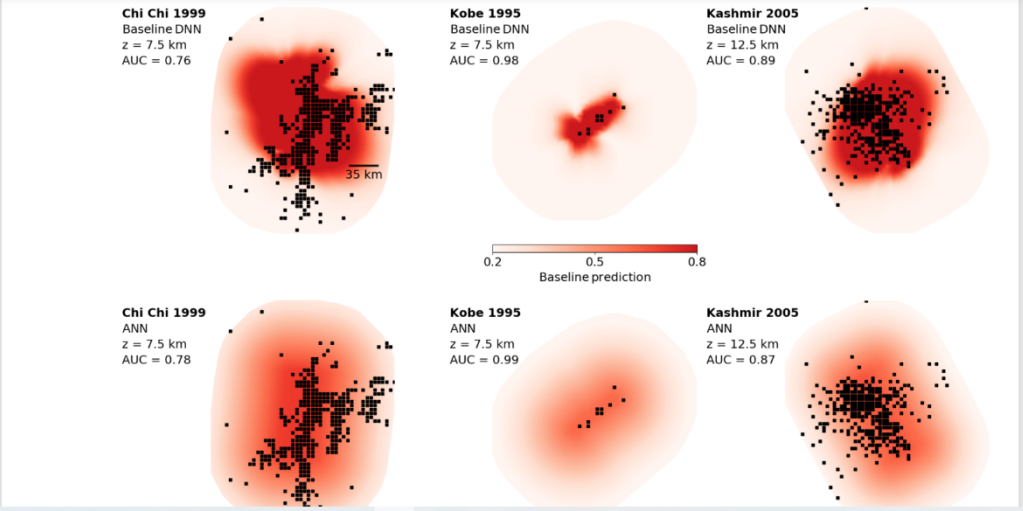
(3) WhatsApp | Calendar | 07-pred_seism_aftXYZ.model_de

localhost:8888/notebooks/07-pred_seism_aftXYZ.model_deployment.ipynb

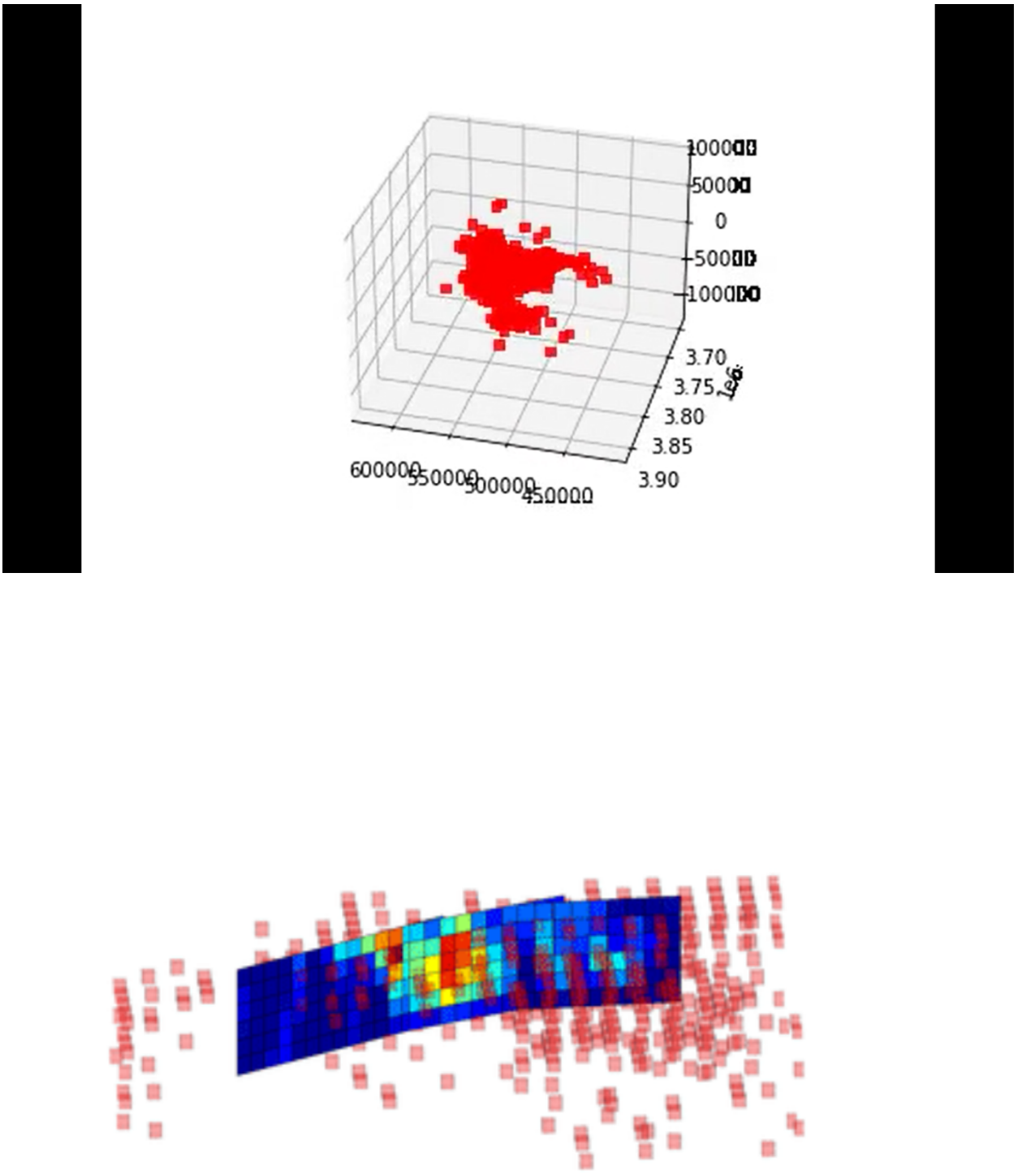
jupyter 07-pred_seism_aftXYZ.model_deployment Last Checkpoint: an hour ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

Run Markdown



39°C Mostly sunny ENG 9:21 PM 5/5/2022



A visual representation of the 1992 magnitude 7.3 southern California Landers earthquake where the multi-colored portion represents the initial quake and the red boxes represent aftershock locations.

PART-I

Chapter 1: Introduction

1.1PROBLEM DEFINITON

From hurricanes and floods to volcanoes and earthquakes, the Earth is continuously evolving in fits and spurts of dramatic activity. Earthquakes and subsequent tsunamis alone have caused massive destruction in the last decade—even over the course of writing this post, there were earthquakes in New Caledonia, Southern California, Iran, and Fiji, just to name a few.

Earthquakes typically occur in sequences: an initial "mainshock" (the event that usually gets the headlines) is often followed by a set of "aftershocks." Although these aftershocks are usually smaller than the main shock, in some cases, they may significantly hamper recovery efforts.

Although the timing and size of aftershocks has been understood and explained by established empirical laws, forecasting the locations of these events has proven more challenging. Aftershocks are a response to changes in stress generated by large earthquakes and represent the most common observations of the triggering of earthquakes. The maximum magnitude of aftershocks and their temporal decay are well described by empirical laws (such as Bath’s law and Omori’s law), but explaining and forecasting the spatial distribution of aftershocks is more difficult.

1.2 PROJECT SPECIFICATION

- Here we use a deep-learning approach to identify a static-stress-based criterion that forecasts aftershock locations without prior assumptions about fault orientation.

- We show that a neural network trained on more than 131,000 main shock–aftershock pairs can predict the locations of aftershocks in an independent test dataset of more than 30,000 main shock–aftershock pairs more accurately (area under curve of 0.849) than can classic Coulomb failure stress change (area under curve of 0.583).

- We find that the learned aftershock pattern is physically interpretable: the maximum change in shear stress, the von Mises yield criterion (a scaled version of the second invariant of the deviatoric stress-change tensor) and the sum of the absolute values of the independent components of the stress-change tensor each explain more than 98 per cent of the variance in the neural-network prediction.

- This machine-learning-driven insight provides improved forecasts of aftershock locations and identifies physical quantities.

1.3

HARDWARE REQUIREMENTS

- System : Intel Core i3, i5, i7 and 2 GHz Minimum
- RAM : 2GB or above
- Hard Disk : 10 GB or above
- Input Device : Keyboard and Mouse
- Output Device : Monitor or PC

1.4 SOFTWARE REQUIREMENTS

- Operating System : Windows 7, 10 or Higher Versions
- Platform : Jupyter Notebook
- Model Building and optimization: TensorFlow, Matplotlib, Scikit learn ,Pandas, Seaborn
- Programming Language : Python
- Deep Learning

Chapter 2: Theory

1.1 General Theory:

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU General Public License (GPL).

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

Python is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Web Development Domain. I will list down some of the key advantages of learning Python:

- Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- Python is Interactive − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- Python is Object-Oriented − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- Python is a Beginner's Language − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Following are important characteristics of **Python Programming** −

- It supports functional and structured programming methods as well as OOP.

- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA

1.2 Specifications of Python:

PYTHON IDENTIFIERS

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).

Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, Manpower and manpower are two different identifiers in Python.

Here are naming conventions for Python identifiers −

- Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
- Starting an identifier with a single leading underscore indicates that the identifier is private.
- Starting an identifier with two leading underscores indicates a strongly private identifier.
- If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

VARIABLES

Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory. Therefore, by assigning different data types to variables, you can store integers, decimals or characters in these variables.

Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable. The equal sign (=) is used to assign values to variables.

The operand to the left of the = operator is the name of the variable and the operand to the right of the = operator is the value stored in the variable.

OPERATORS

Operators are the constructs which can manipulate the value of operands.

Consider the expression 4 + 5 = 9. Here, 4 and 5 are called operands and + is called operator.

TYPES OF OPERATORS

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

MODUELS:

A module allows you to logically organize your Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes that you can bind and reference.

Simply, a module is a file consisting of Python code. A module can define functions, classes and variables. A module can also include runnable code.

I/O:

Sr.No.	Modes & Description
1	**r** Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
2	**rb** Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
3	**r+** Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
4	**rb+** Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
5	**w** Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.

6	**wb** Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
7	**w+** Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
8	**wb+** Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
9	**a** Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
10	**ab** Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
11	**a+** Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
12	**ab+** Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

Exception Handling:

Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them −

- **Exception Handling** − This would be covered in this tutorial. Here is a list standard Exceptions available in Python: Standard Exceptions.
- **Assertions** − This would be covered in Assertions in Python tutorial

OOPS CONCEPT:

Python has been an object-oriented language since it existed. Because of this, creating and using classes and objects are downright easy. This chapter helps you become an expert in using Python's object-oriented programming support.

If you do not have any previous experience with object-oriented (OO) programming, you may want to consult an introductory course on it or at least a tutorial of some sort so that you have a grasp of the basic concepts.

However, here is small introduction of Object-Oriented Programming (OOP) to bring you at speed −

- **Class** − A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.
- **Class variable** − A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.
- **Data member** − A class variable or instance variable that holds data associated with a class and its objects.

- **Function overloading** − The assignment of more than one behaviour to a particular function. The operation performed varies by the types of objects or arguments involved.
- **Instance variable** − A variable that is defined inside a method and belongs only to the current instance of a class.
- **Inheritance** − The transfer of the characteristics of a class to other classes that are derived from it.
- **Instance** − An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle.
- **Instantiation** − The creation of an instance of a class.
- **Method** − A special kind of function that is defined in a class definition.
- **Object** − A unique instance of a data structure that's defined by its class. An object comprises both data members (class variables and instance variables) and methods.
- **Operator overloading** − The assignment of more than one function to a particular operator.

- Highly Extensible and Easily Readable Language.

Deep Learning

Deep learning is a class of machine learning algorithms that uses multiple layers to progressively extract higher-level features from the raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.

In deep learning, each level learns to transform its input data into a slightly more abstract and composite representation. In an image recognition application, the raw input may be a matrix of pixels; the first representational layer may abstract the pixels and encode edges; the second layer may compose and encode arrangements of edges; the third layer may encode a nose and eyes; and the fourth layer may recognize that the image contains a face. Importantly, a deep learning process can learn which features to optimally place in which level on its own.

A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers. There are different types of neural networks but they always consist of the same components: neurons, synapses, weights, biases, and functions. These components functioning similar to the human brains and can be trained like any other ML algorithm.

For example, a DNN that is trained to recognize dog breeds will go over the given image and calculate the probability that the dog in the image is a certain breed. The user can review the results and select which probabilities the network should display (above a certain threshold, etc.) and return the proposed label. Each mathematical manipulation as such is considered a layer and complex DNN have many layers, hence the name "deep" networks.

DNNs can model complex non-linear relationships. DNN architectures generate compositional models where the object is expressed as a layered composition of primitives. The extra layers enable composition of features from lower layers, potentially modeling complex data with fewer units than a similarly performing shallow network. For instance, it was proved that sparse multivariate polynomials are exponentially easier to approximate with DNNs than with shallow networks.

Deep architectures include many variants of a few basic approaches. Each architecture has found success in specific domains. It is not always possible to compare the performance of multiple architectures, unless they have been evaluated on the same data sets.

DNNs are typically feed forward networks in which data flows from the input layer to the output layer without looping back. At first, the DNN creates a map of virtual neurons and assigns random numerical values, or "weights", to connections between them. The weights and inputs are multiplied and return an output between 0 and 1. If the network did not accurately recognize a particular pattern, an algorithm would adjust the weights. That way the algorithm can make certain parameters more influential, until it determines the correct mathematical manipulation to fully process the data.

Matplotlib

Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy. As such, it offers a viable open source alternative to MATLAB. Developers can also use matplotlib’s APIs (Application Programming Interfaces) to embed plots in GUI applications.

A Python matplotlib script is structured so that a few lines of code are all that is required in most instances to generate a visual data plot. The matplotlib scripting layer overlays two APIs:

The pyplot API is a hierarchy of Python code objects topped by matplotlib.pyplot

An OO (Object-Oriented) API collection of objects that can be assembled with greater flexibility than pyplot. This API provides direct access to Matplotlib’s backend layers.

The pyplot API has a convenient MATLAB-style stateful interface. In fact, matplotlib was originally written as an open source alternative for MATLAB. The OO API and

its interface is more customizable and powerful than pyplot, but considered more difficult to use. As a result, the pyplot interface is more commonly used, and is referred to by default in this article.

Understanding matplotlib's pyplot API is key to understanding how to work with plots:

matplotlib.pyplot.figure: Figure is the top-level container. It includes everything visualized in a plot including one or more Axes.

matplotlib.pyplot.axes: Axes contain most of the elements in a plot: Axis, Tick, Line2D, Text, etc., and sets the coordinates. It is the area in which data is plotted. Axes include the X-Axis, Y-Axis, and possibly a Z-Axis, as well.

PART-II

CHAPTER 3:

INTRODUCTION:

The present project aims at forecasting the location of aftershocks that follow the occurrence of any large earthquake. While mainshocks are not (yet) predictable, aftershock statistics are sound enough for us to try predict them in reasonably small magnitude-space-time windows. For any mainshock of magnitude M occurring, B˚ath's law combined to the Gutenberg-Richter law tells us that there is a c. 10% chance that another earthquake (i.e. an aftershock) of the same magnitude or greater will occur (B˚ath, 1965; Gutenberg & Richter, 1944). Aftershocks are also most likely to occur just after the mainshock with their likelihood decreasing with time (Mignan, 2015 and references therein). Predicting where they are most likely to occur represents an important endeavour to improve time-dependent seismic hazard and risk assessment (e.g. Mignan et al., 2018). So far, the preferred model to predict aftershock spatial patterns is the Coulomb stress model (e.g. King et al., 1994). Recent results (De Vries et al., 2018) as well as the present project show that machine learning (ML) can significantly improve those predictions.

OBJECTIVES OF THE STUDY:

The proposed work is aimed to carry out work leading to the development of an approach
for forecasting aftershock locations. The proposed aim will be achieved by dividing the work into following objectives:
1. This project aims at forecasting the location of aftershocks that follow the occurrence of any large earthquake
2. Aftershock forecasts can help a wide range of users including the public, emergency managers, and lifeline engineers prepare for, respond to, cope with, and recover from earthquakedisasters.
3. By bypassing stress computation entirely, using a simple ANN topology, and only mainshock-rupture-based features, we can streamline the process of aftershock pattern prediction for future post-mainshock crisis management (i.e. process faster and more transparent);

SCOPE OF THE WORK:

We have applied a neural net to analyze the relationships
between static stress changes caused by the mainshocks and
aftershock locations. The algorithm was able to identify useful
patterns.
The end result was an improved model to forecast aftershock
locations and while this system is still imprecise, it's a motivating
step forward. Machine learning-based forecasts may
one day help deploy emergency services and inform evacuation
plans for areas at risk of an aftershock

.These are some of the scopes and scalability scopes of the project I made and its prospectus and implementation in large part of finance industry.

CHAPTER 4:

METHODOLOGY ADOPTED:

The following methodology will be followed to achieve the objectives defined for proposed research work:

• The idea is to observe a volume which extends 100 kmhorizontally and 50 km vertically from the main shock.We then break that volume into 5kmx 5km x 5km small volumes and calculate the elastic stress change tensors ateach of their centroid.

• Now using that information we need to predict whether there was an aftershock in that small volume or not. In order to know the ground truth we use International Seismological Center(ISC) event catalogue, in which for each main shock we looked up for its corresponding aftershocks from 1 sec to 1 year time and using that information we created our ground truth i.e. whether there was an aftershock in that small volume or not.

• So this whole problem is now a binary classification problem, in which our neural network has to predict was there an aftershock in that small 5km x 5km x 5km region or not. The model will use deep learning techniques to predict whether there can be aftershock at a particular locations or not. I'll be taking the data provided by the SRCMOD http://equakerc. info/SRCMOD/searchmodels/allevents/.

• The format of the data will be FSP (finite-source rupture model). We'll not be working on how to process those SRCMOD files in order to create a csv. Rather we'll use already created CSVs.

• The data provided by the SCRMOD file contains the information about the hypocenter, latitude, longitude, magnitude, strike, dip, the inversion parameters and many other relevant data that is sufficient to get an insight of the earthquake.

• We'll be feeding these data to the neural network having several hidden layers (it depends on you, try and experiment with it), which will then try to extract the important features in order to predict the whether there is a chance of having an aftershock or not. we'll be using two activation functions tanh and ReLu (again experiment with it). The output layer will be a sigmoid layer, which will give us a probability between 0 - 1, as to how confident it is.

Step By Step

1) Generating Data

First download all labelled samples per mainshock rupture model, in csv format, into AllCSV folder from the DeVries18 Google Drive fromgiven link. Read the template files nd save to a handy format that can be used later.

2) Initial Data Exploration

Initial data exploration for the project ’Aftershock pattern prediction based on earthquake rupture data for improved seismic hazard assessment’. DeVries18 will refer to the article
’Deep learning of aftershock patterns following large earthquakes’ by Phoebe M. R. DeVries, Fernanda Vi´egas,m Martin Wattenberg & Brendan J. Meade, and published in Nature in 2018 (https://www.nature.com/articles/s41586-018- 0438-y). Data source : Finite-Source Rupture Model Database Import the original mainshock rupture SRCMOD data files from the project’s GitHub repository.
We first retrieved the same 199 SRCMOD.fsp files as used in DeVries18 from here

Sample FSP :

```
% --------------------------------  FINITE-SOURCE RUPTURE MODEL  --------------------------------
%
% Event : Hyuga-nada (Japan)           04/01/1968             [Yagi et al. (1998) ]
% EventTAG: s1968HYUGAx01YAGI
%
% Loc   : LAT  = 32.28          LON = 132.53           DEP = 15.0
% Size  : LEN  = 72.0 km        WID =  63.0 km         Mw = 7.53       Mo = 2.22e+20 Nm
% Mech  : STRK = 227.0          DIP = 12.0             RAKE = 90.0     Htop = 10.32 km
% Rupt  : HypX = 31.50 km       HypZ = 22.50 km                avTr = 12.0 s   avVr = 2.8 km/s
%
% --------------------------------  inversion-related parameters  --------------------------------
%
% Invs :  Nx  =  8      Nz  = 7         Fmin = 999.00 Hz        Fmax = 999.00 Hz
% Invs :  Dx  =  9.00 km        Dz  = 9.00 km
% Invs :  Ntw =  1      Nsg =  1                        (# of time-windows,# of fault segments)
% Invs :  LEN =  999.0 s        SHF =  0.0 s            (time-window length and time-shift)
% SVF  :  unknown                                       (type of slip-velocity function used)
%
% Data :        SGM     TELE    TRIL    LEVEL   GPS     INSAR   SURF    OTHER
% Data :        999     0       0       0       0       0       0       0
% PHImx:        999     0       0       0       0       0       0       0
% Rmin :        999     0       0       0       0       0       0       0
999.0   0.0     0.0     0.0     0.0     0.0     0.0     0.0
%
% ----------------------------------------------------------------------------------------------
%
```

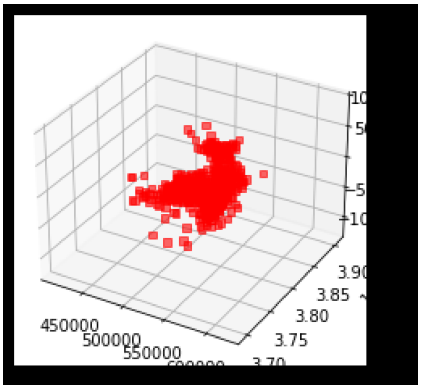
File Edit View Insert Cell Kernel Widgets Help Trusted | Python 3 (ipykernel)


```
% --------------------------------  FINITE-SOURCE RUPTURE MODEL  --------------------------------
%
% Event : Hyuga-nada (Japan)           04/01/1968             [Yagi et al. (1998) ]
% EventTAG: s1968HYUGAx01YAGI
%
% Loc   : LAT  = 32.28          LON = 132.53           DEP = 15.0
% Size  : LEN  = 72.0 km        WID =  63.0 km         Mw = 7.53       Mo = 2.22e+20 Nm
% Mech  : STRK = 227.0          DIP = 12.0             RAKE = 90.0     Htop = 10.32 km
% Rupt  : HypX = 31.50 km       HypZ = 22.50 km                avTr = 12.0 s   avVr = 2.8 km/s
%
% --------------------------------  inversion-related parameters  --------------------------------
%
% Invs :  Nx  =  8      Nz  = 7         Fmin = 999.00 Hz        Fmax = 999.00 Hz
% Invs :  Dx  =  9.00 km        Dz  = 9.00 km
% Invs :  Ntw =  1      Nsg =  1                        (# of time-windows,# of fault segments)
% Invs :  LEN =  999.0 s        SHF =  0.0 s            (time-window length and time-shift)
% SVF  :  unknown                                       (type of slip-velocity function used)
%
% Data :        SGM     TELE    TRIL    LEVEL   GPS     INSAR   SURF    OTHER
% Data :        999     0       0       0       0       0       0       0
% PHImx:        999     0       0       0       0       0       0       0
% Rmin :        999     0       0       0       0       0       0       0
999.0   0.0     0.0     0.0     0.0     0.0     0.0     0.0
%
% ----------------------------------------------------------------------------------------------
%
```


```
% VELOCITY-DENSITY STRUCTURE
% No. of layers =  1
%
% Crustal Model unknown for this model; assumed shear modulus:
%   [10**10 N/m^2]
%  3.30
```

```
% VELOCITY-DENSITY STRUCTURE
% No. of layers =   1
%
% Crustal Model unknown for this model; assumed shear modulus:
%   [10**10 N/m^2]
%  3.30
%
% ------------------------------------------------------------------------------------------------
% 23-Jul-2007 PMM (mai@sed.ethz.ch)
% ------------------------------------------------------------------------------------------------
%
% SOURCE MODEL PARAMETERS
%       Nsbfs = 56 subfaults
%               X,Y,Z coordinates in km; SLIP in m
%               if applicable: RAKE in deg, RISE in s, TRUP in s, slip in each TW in m
%
%   Coordinates are given for top-center of each subfault or segment: |'|
%   Origin of local coordinate system at epicenter: X (EW) = 0,  Y (NS) = 0
%    LAT       LON       X==EW       Y==NS       Z       SLIP
% ------------------------------------------------------------------------------------------------
32.3009  132.9000   34.7562    2.3181   10.3200    0.5400
32.2456  132.8299   28.1740   -3.8199   10.3200    0.5700
32.1904  132.7598   21.5918   -9.9579   10.3200    0.4500
32.1351  132.6898   15.0096  -16.0959   10.3200    0.5900
32.0799  132.6197    8.4275  -22.2339   10.3200    1.1000
32.0247  132.5496    1.8453  -28.3718   10.3200    1.6900
31.9694  132.4796   -4.7369  -34.5098   10.3200    1.1700
31.9142  132.4095  -11.3191  -40.6478   10.3200    0.3700
32.3588  132.8361   28.7523    8.7564   12.1912    0.5200
32.3036  132.7660   22.1701    2.6185   12.1912    0.8100
32.2483  132.6959   15.5880   -3.5195   12.1912    1.6800
```

Visualize one labelled sample - let us consider the 1992 Landers case for one specific rupture model (event tag '1992LANDER01COHE'). We here want to reproduce an animation similar to the one shown in the this Google blog post to check the aftershock 3D pattern.



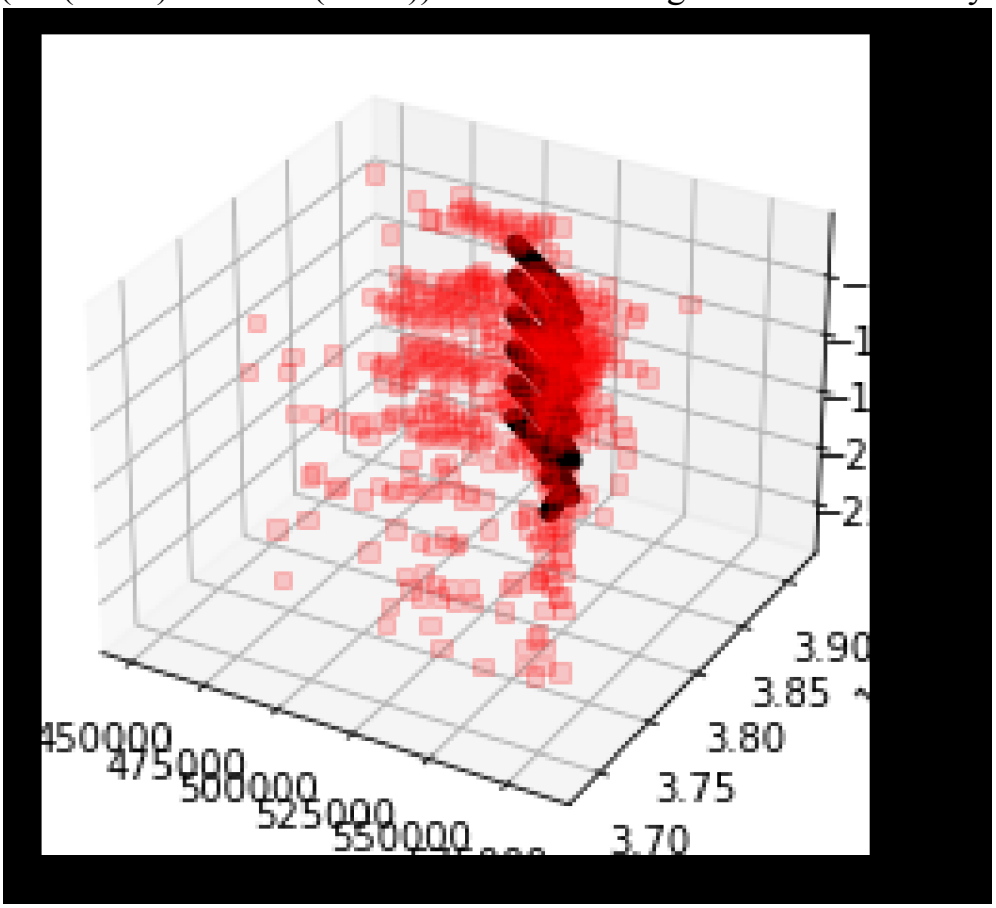
3) Extract-Transform-Load (ETL)
Extract::
• The data needed for this project were already extracted in the notebook predseismaftXY Z.dataexp for preliminary data exploration.

• Mainshock rupture files (SRCMOD) extracted from here, originally from here, then filtered manually, considering only the 199 mainshocks used in the DeVries18 publication
• Labelled data of the baseline model (completedatasetlabelled) imported as pickle file, created from a list of csv files originally imported from
the Google Drive
• The fsp format is quite complex and must be cleaned for feature engineering
We here directly 'extract' all needed data from our working directory.
Transform ::
Fortunately, we can use the srcmod.py program which reads and transform fsp files to be (originally) used as input for Coulomb stress modelling. The original code is available on the following Google GitHub repository. Although we won't do any Coulomb stress modelling, using the same input format will simplify feature engineering in the next step of the process model. Note that the original code was done with Python 2. For Python 3, we made the following modifications:
• printx changed to print(x),
• .haskey changed to in
• We also removed the gcs module with
gcs.F ile(filename) changed to open(filename,′ r′).
• Changed proj utm source.

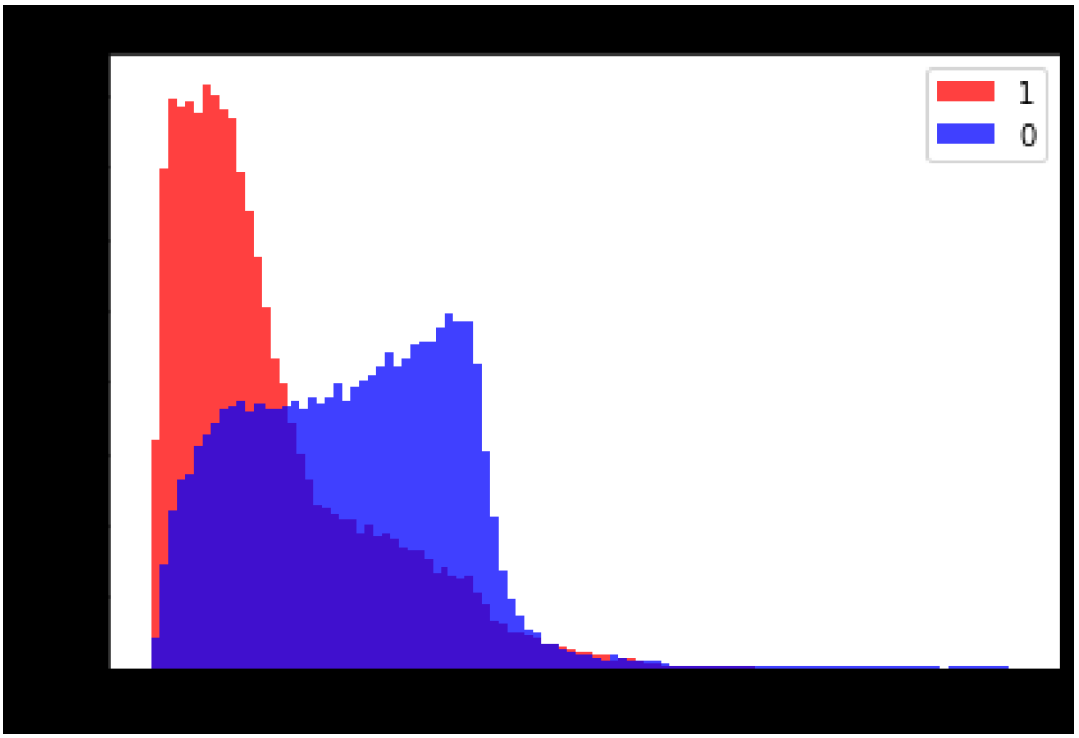
The updated code srcmod.py is available in this project's GitHub repository under utils. The original srcmod reader was a little fragile, failing to read in many of the .mat files. ReadSrcmodFile fixes this – it reads in the raw ascii from a .fsp file, and returns the data in the same format. There are some small differences in the data, mostly due to the fact that .mat files are 64-bit, and we read in the ascii, and covert to floats. As such, there's some small differences in the read in values, but this routine does work with all the data. Far more manageable than the fsp file format, we will use defaultdict objects in the next step of the process model to define new features based on the SRCMOD rupture parameters (geometry and kinematics). The structure of the dictionary created by ReadSrcmodFile is first further simplified to use cell centers instead of 4 corners per cell. No transformation needed for the labelled dataset of the baseline model, thus , we will directly use the dataframe dfDeV ries18 for feature engineering.

Load::
Two objects will be used for feature definition:
• SRCMODdictList , a list of dictionaries, which contains all required rupture parameters per mainshock 'ID' (saved as SRCMODcleaned.pkl);
• dfDeV ries18, a dataframe, which contains the labeled dataset of the baseline model and where mainshocks are identified by 'ID' (already saved as completedatasetlabelled.pkl).
4) Feature Engineering
• Make balanced dataset

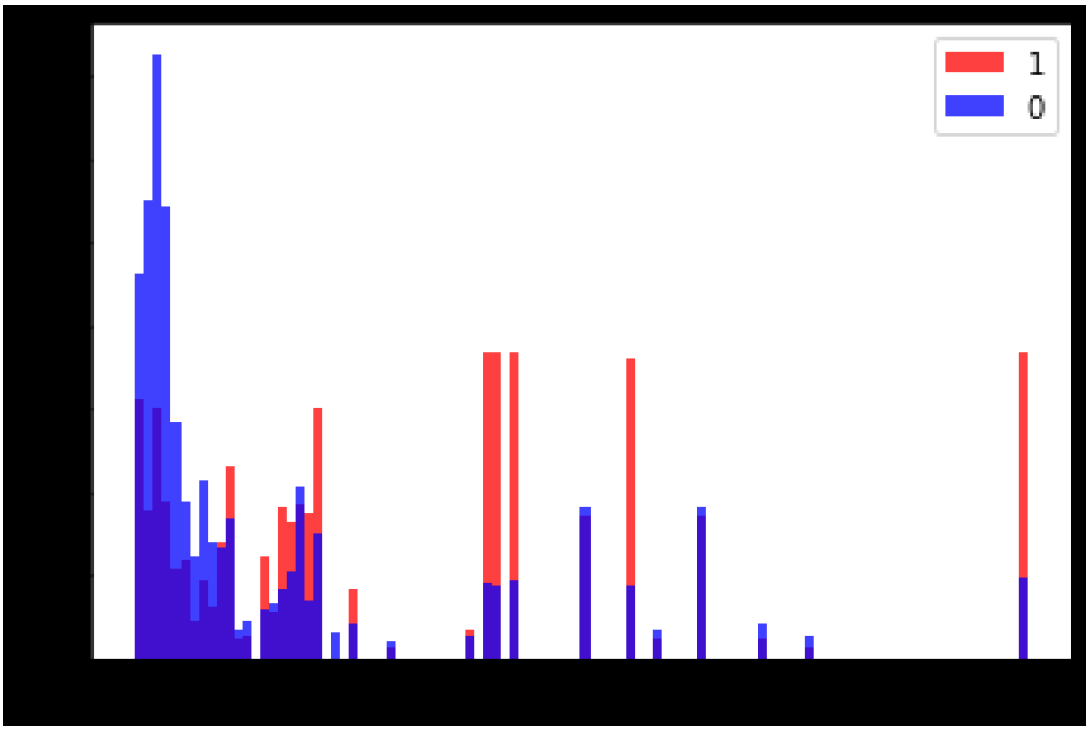
• Unbalanced towards 0 values
• Subsample 0s - we use the same approach as in De-vVries18 for consistency : Subsampling here reduces the number of samples from 6+ million to c. 200,000
• Merge 1s and 0s We do the same engineering of stress features as done in DeVries18, spliting the 6 stress components into 12 (abs(stress)and − abs(stress)) and normalizing the new features by 1e6.



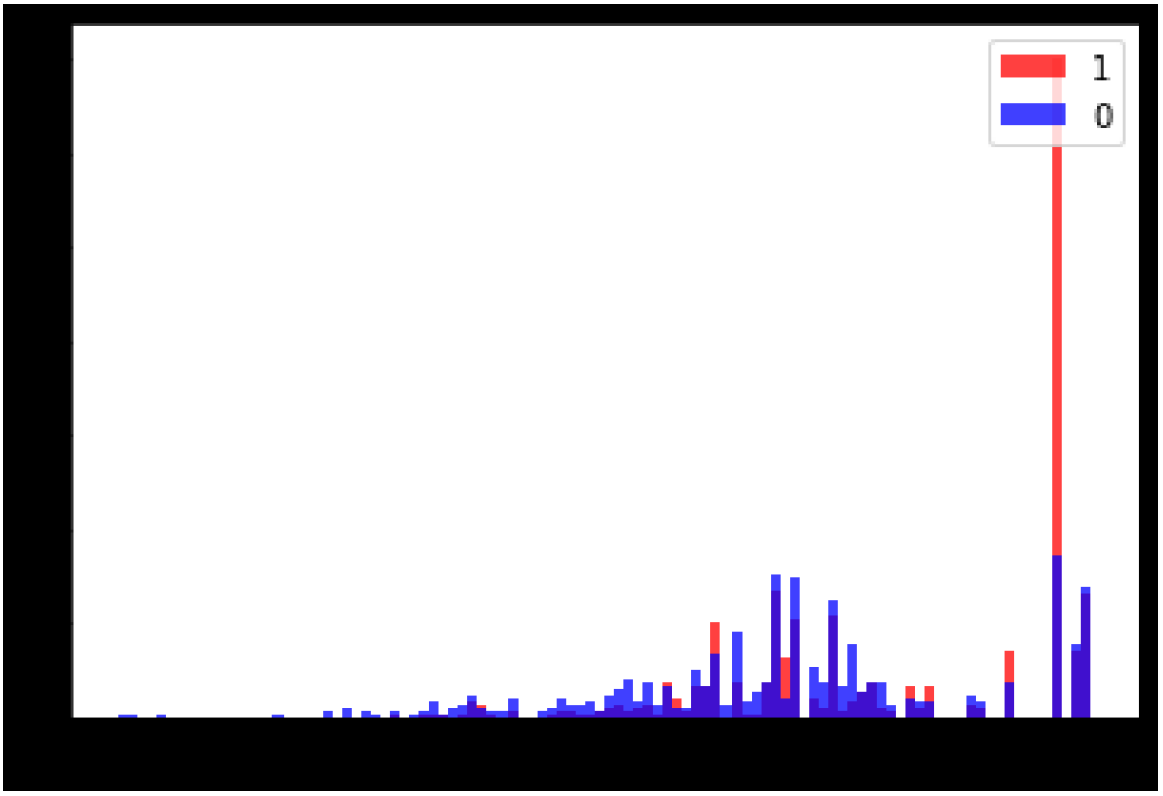
The simplest choice is to define as feature the nearest distance between aftershock cells and rupture (see animation above). Since DeVries18 found the absolute values of stress components to lead to AUC = 85%, we must identify some rupture parameters that will lead our ML algorithm to approximate, in part, those stress operations. Since stress depends on deformation, mainshock rupture slip should be a feature. Various features are defined in the next step of the process model. Mindist



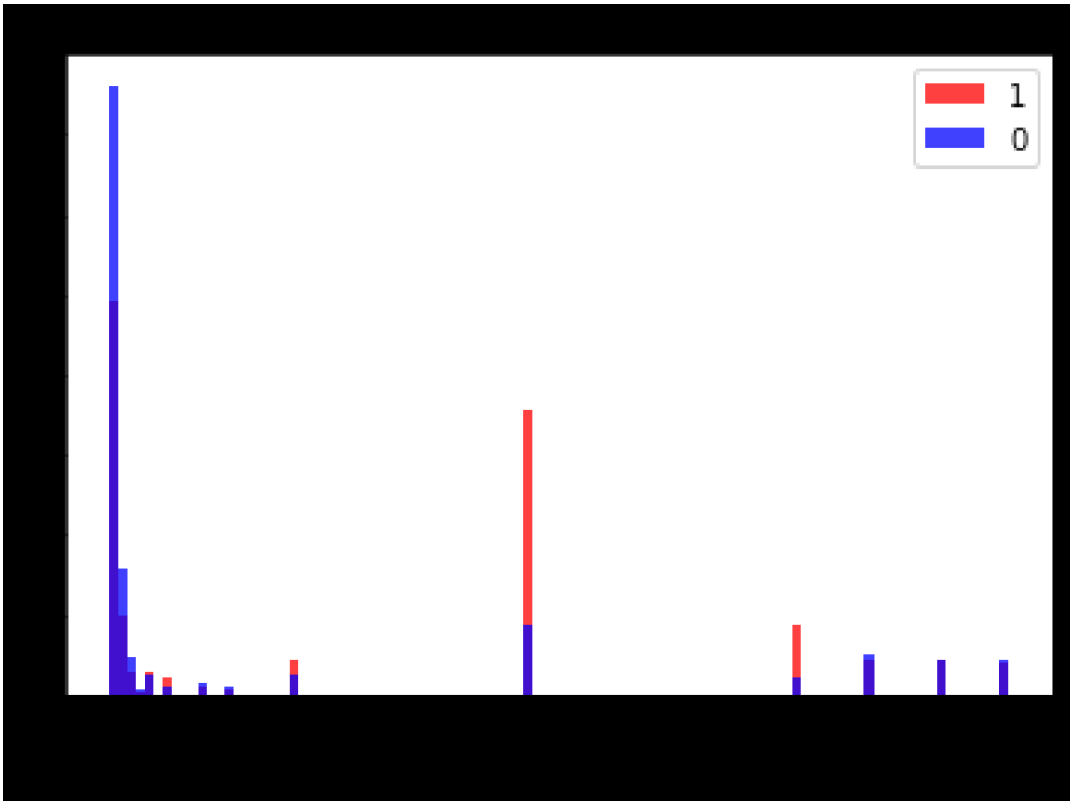
Slip-mean aftershock rupture



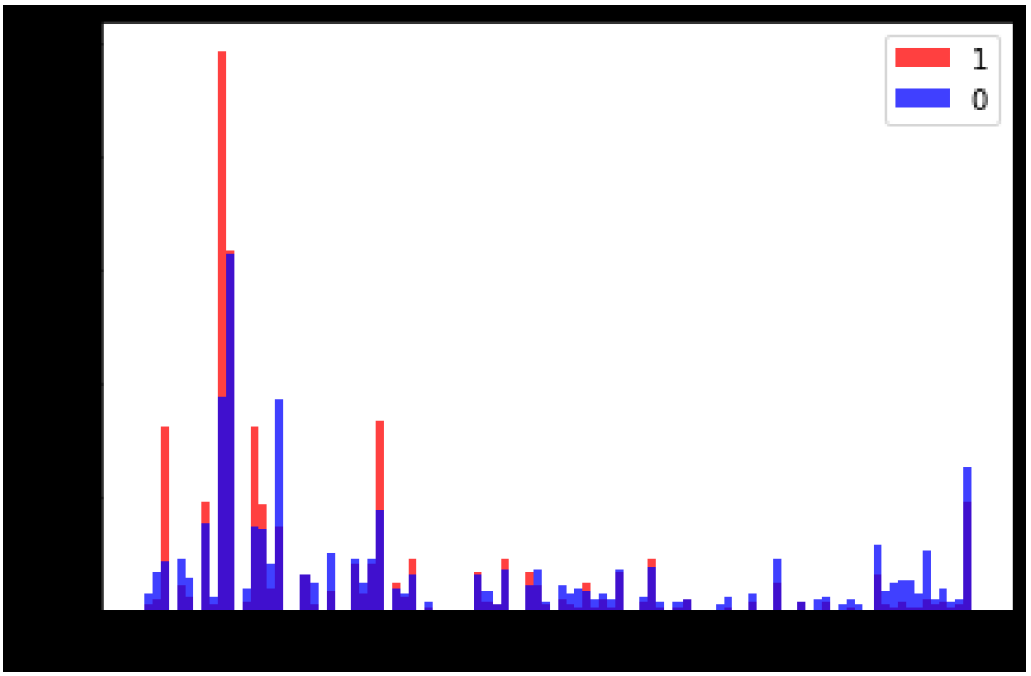
Mindist aftershock rupture


1
0

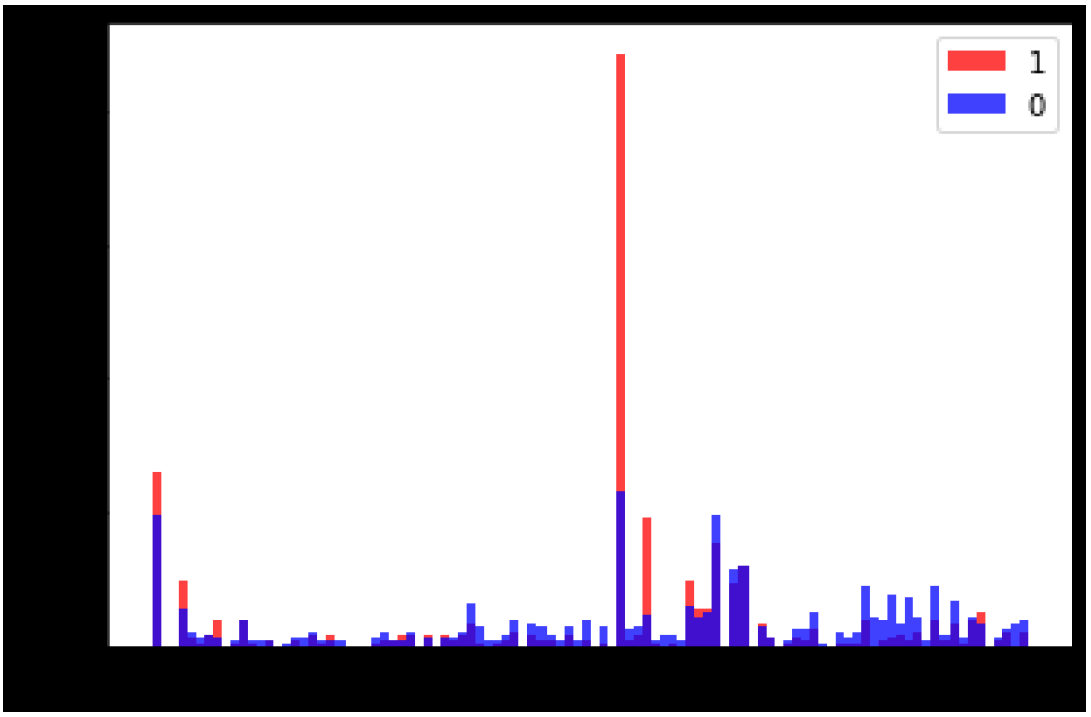
Magnitude aftershock rupture



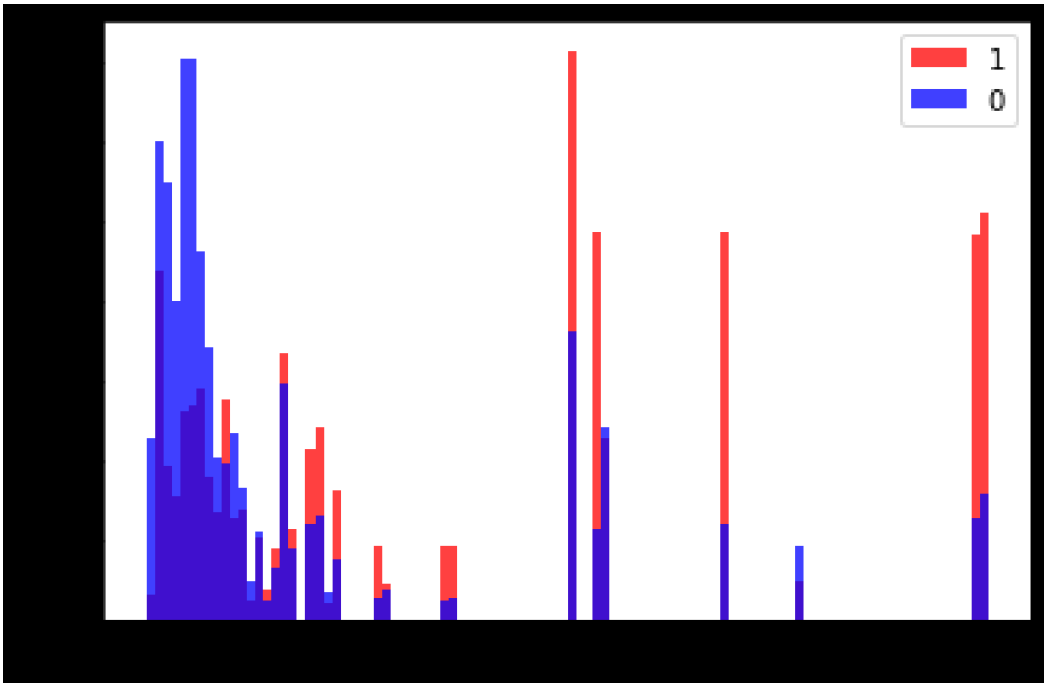
Moment aftershock rupture



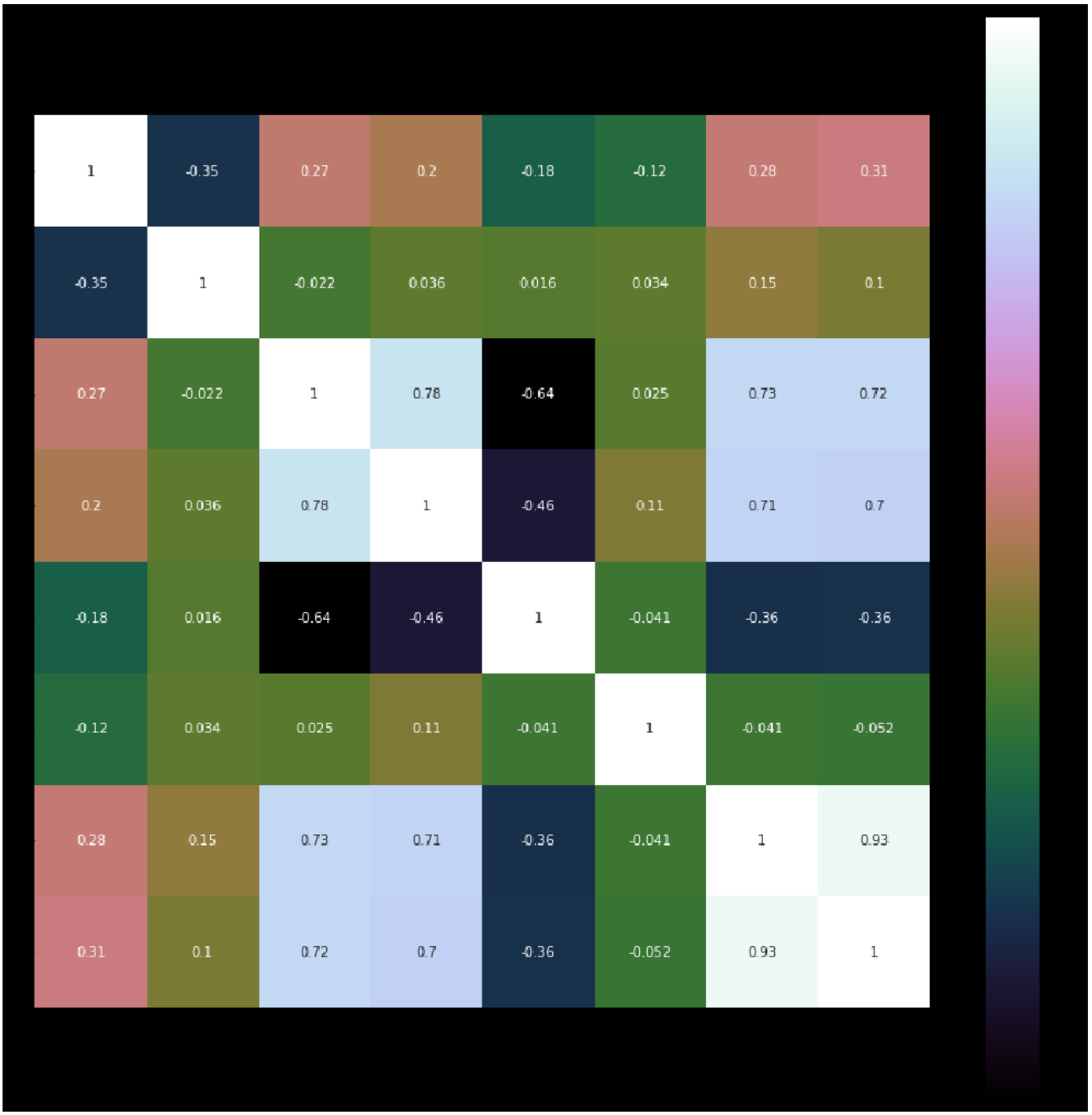
Dip-Mean aftershock rupture



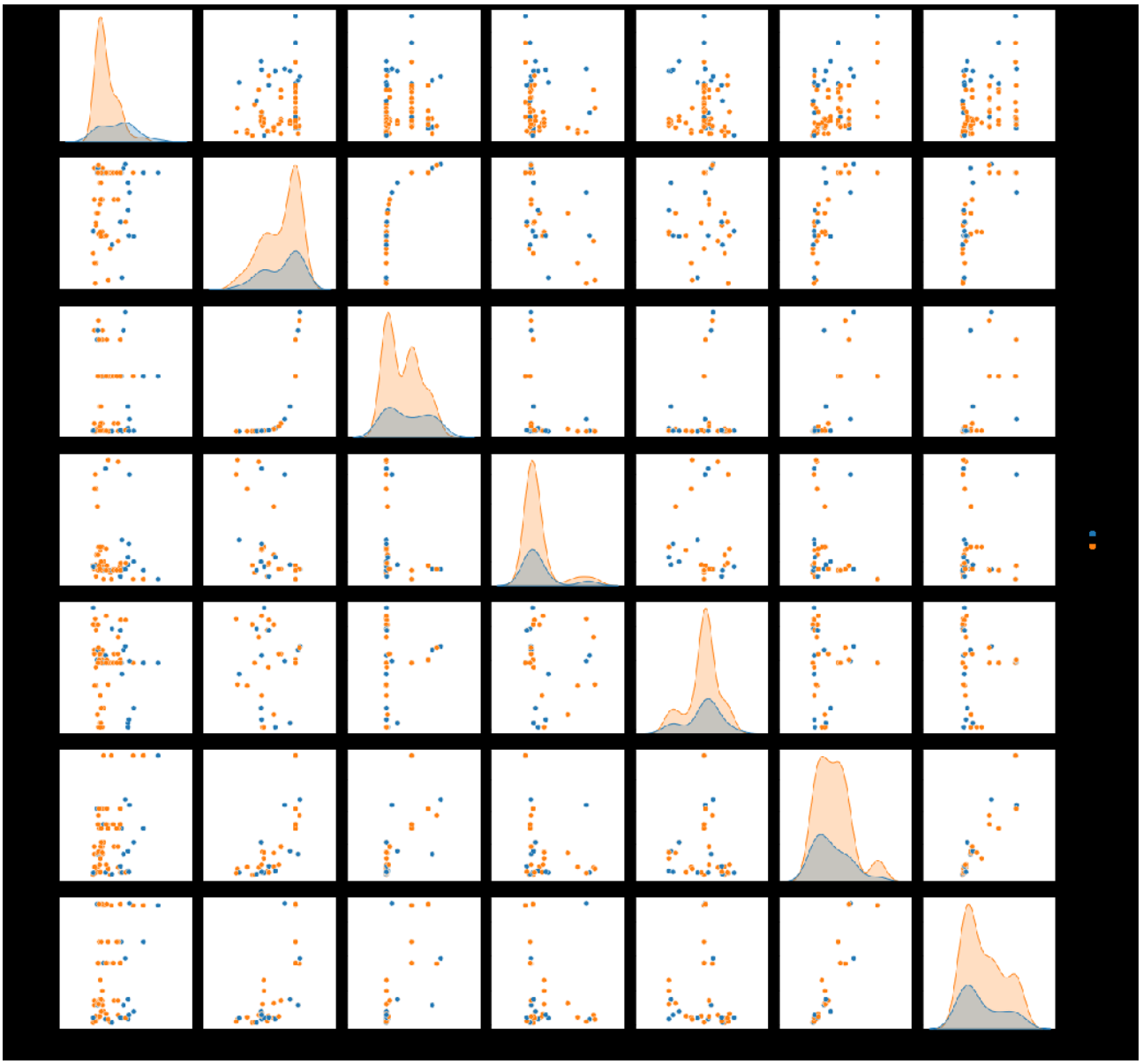
Strike-Mean aftershock rupture



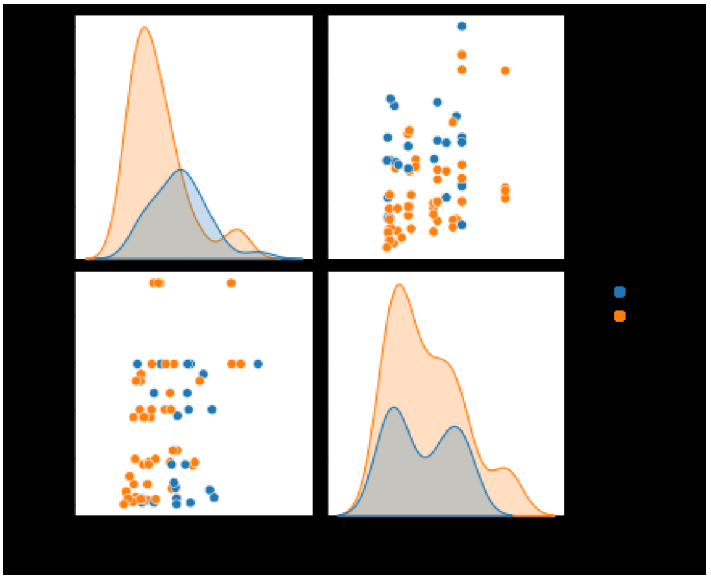
Slip-Max aftershock rupture



New Features Co-relation



New features Sample



Model Definition

• We here reproduce the DeVries18 deep learning model = six hidden layers with 50 neurons each and hyperbolic tangent activation functions (13,451 weights and biases in total).

• Since the DeVries model has a 12-neuron input layer, we first verify whether a much simpler Deep Neural Network (DNN) topology can lead to similar results. We also reduce the dropout rate. We keep everything else the same (i.e. kernelinitializer, activation, optimizer, loss). This model is for illustration purposes only. A different model with different features is proposed in this project.

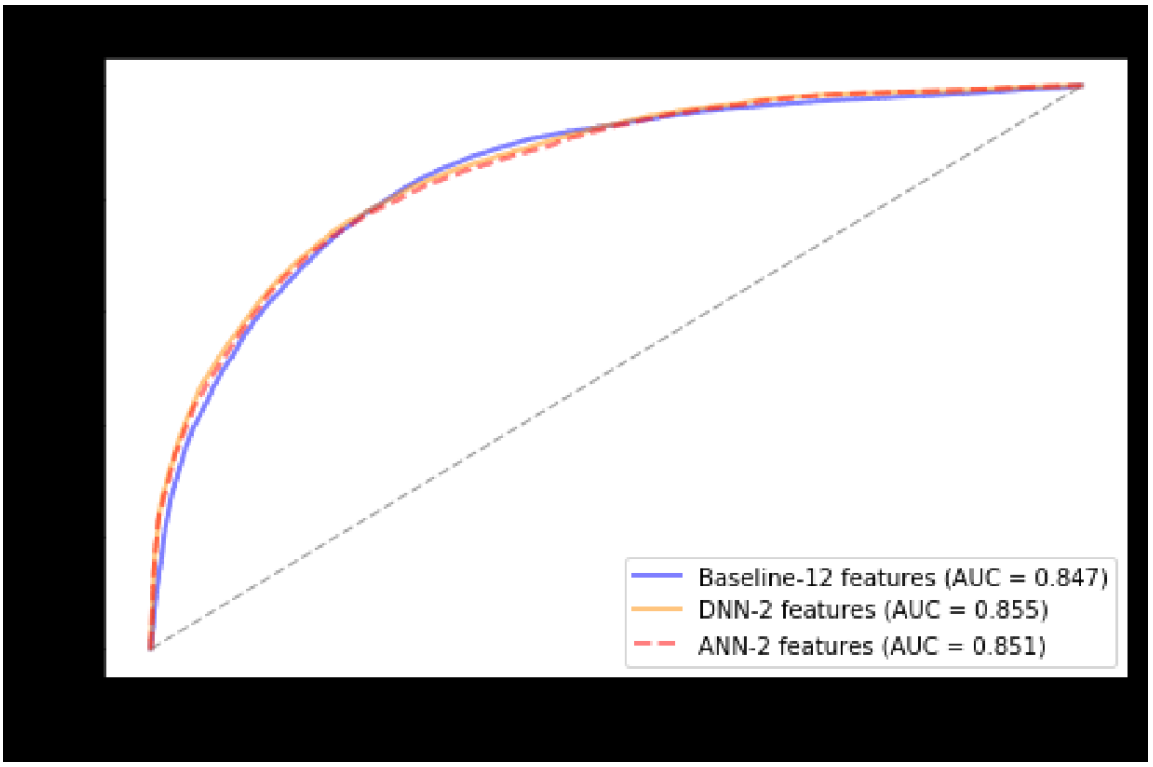
• We use more direct features than in DeVries18, here based on geometry/kinematics instead of stress. We further simplify the topology of the DNN compared to the baseline.

• We also test a shallow Artificial Neural Network (ANN) (i.e. no deep learning, only one hidden layer).

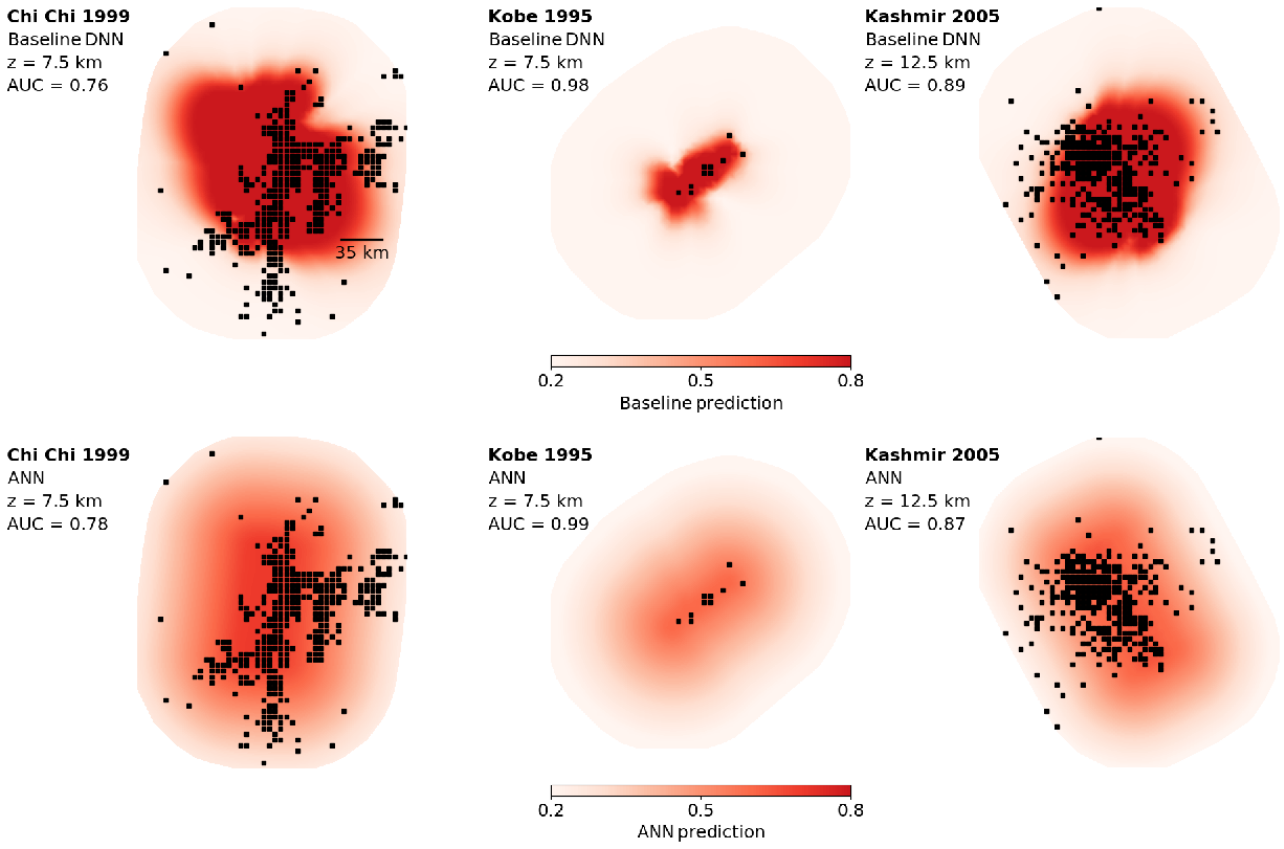
• NB: During training, we will also test another standard machine learning algorithm (XGBoost) for completeness.

Model Training

This is the baseline model CV results - we should aim at achieving similar or greater accuracy (72%) for the new model with new set of features. This will however not assure a same or better generalization than the baseline model (AUC = 85%, see next step of the process model). Since the Devries18 model uses a relatively small input layer (12 nodes), a simpler DNN topology should do just fine. We show it, only for illustrative purpose. The model proposed in this project will use another set of features. Training new models with the proposed set of features based on geometry and kinematics (minimum distance to mainshock rupture and mean slip on rupture). To not only test neural networks, we here test another standard machine learning algorithm. XGBoost has become a method of choice in recent Kaggle competitions (althought this does not mean that it will perform better than neural networks



We finally produce plots similar to Fig. 1 of DeVries18 for a last check. Most of the plot formatting comes from PlotThreeTestCases.py of DeVries18.



CHAPTER 5:

RESULTS AND DISCUSSION:

The DeVries18 results are first reproduced (AUC = 85%). This model (deep neural network (DNN) 12 - 50 - 50 - 50 - 50 - 50 -50 - 1) is then used as baseline;

• Since DeVries only used 12 stress-related features, we test simpler DNN topologies. A new DNN 12 - 8 - 8 – 1 yields a similar performance (AUC = 85%);

• Since (1) stress computation (i.e. mapping of deformation to stress via geometric operations; King et al., 1994) is computationally intensive and cumbersome, and since (2) neural networks can be interpreted as geometric operations themselves, we bypass stress computation entirely and define features based on mainshock rupture geometry and kinematics instead;

• We find that a DNN (2 - 6 - 6 - 1) with only two features (minimum distance to rupture and mean rupture slip) yields a similar performance (AUC = 85%);

• Since aftershock patterns are in fact relatively simple at first order (Mignan, 2018), an artificial neural network (ANN, 2 - 30 - 1), based on the universal approximation theorem, yields a similar performance (AUC = 85)

• This is an example of Occam's Razor in action. In view of the AI/earthquake-prediction buzz observed in the media (see inset below), one could have imagined that afterhosks represent a complex problem that only artificial intelligence can solve. Although ML helps us better predicting aftershock patterns, there is nothing magical about it. No deep learning was in fact needed with aftershock patterns already well described by only 2 features, which is in agreement with any visual inspection
of the aftershock data.

The impact of these new results is twofold:

• By bypassing stress computation entirely, using a simple ANN topology, and only 2 mainshock-rupture-based features, we can streamline the process of aftershock pattern prediction for future post-mainshock crisis management (i.e. process faster and more transparent);

• In our in-depth analysis, we were not able to obtain performances higher than AUC = 86% despite using stress, or geometry and kinematics, and this for various

hyperparameterizations. This infers that important features are missing. Observing past aftershock patterns, we suggest that performance could be further improved if the location of nearby faults were included. Testing this hypothesis is however out of the scope of the present project.

CHAPTER 6:

CONCLUSIONS AND FUTURE STUDY:

The present project aims at forecasting the location of aftershocks that follow the occurrence of any large earthquake. While mainshocks are not (yet) predictable, aftershock statistics are sound enough for us to try predict them in reasonably small magnitude-space-time windows. For any mainshock of magnitude M occurring, Båth's law combined to the Gutenberg-Richter law tells us that there is a c. 10% chance that another earthquake (i.e. an aftershock) of the same magnitude or greater will occur (Båth, 1965; Gutenberg & Richter, 1944). Aftershocks are also most likely to occur just after the mainshock with their likelihood decreasing with time (Mignan, 2015 and references therein). Predicting where they are most likely to occur represents an important endeavour to improve time-dependent seismic hazard and risk assessment (e.g. Mignan et al., 2018). So far, the preferred model to predict aftershock spatial patterns is the Coulomb stress model (e.g. King et al., 1994). Recent results (De Vries et al., 2018) as well as the present project show that machine learning (ML) can significantly improve those predictions.

Our work is based on the recent article 'Deep learning of aftershock patterns following large earthquakes' by Phoebe M. R. DeVries, Fernanda Viégas, Martin Wattenberg & Brendan J. Meade, published in Nature in 2018 (https://www.nature.com/articles/s41586-018-0438-y). We will refer to that study as DeVries18 hereafter. To the best of our knowledge, this is the first study to use ML (here deep learning) to predict the spatial patterns of aftershocks. As such, DeVries18 represents a milestone in earthquake ML research. It also provides us a framework upon which we can test new models and develop new hypotheses and products.

Our results can be summarized as follows:

The DeVries18 results are first reproduced (AUC = 85%). This model (deep neural network (DNN) 12 - 50 - 50 - 50 - 50 - 50 -50 - 1) is then used as baseline;

Since DeVries only used 12 stress-related features, we test simpler DNN topologies. A new DNN 12 - 8 - 8 - 1 yields a similar performance (AUC = 85%);

Since (1) stress computation (i.e. mapping of deformation to stress via geometric operations; King et al., 1994) is computationally intensive and cumbersome, and since (2) neural networks can be interpreted as geometric operations themselves, we bypass stress computation entirely and define features based on mainshock rupture geometry and kinematics instead;

We find that a DNN (2 - 6 - 6 - 1) with only two features (minimum distance to rupture and mean rupture slip) yields a similar performance (AUC = 85%);

Since aftershock patterns are in fact relatively simple at first order (Mignan, 2018), an artificial neural network (ANN, 2 - 30 - 1), based on the universal approximation theorem, yields a similar performance (AUC = 85%).

This is an example of Occam's Razor in action. In view of the AI/earthquake-prediction buzz observed in the media (see inset below), one could have imagined that afterhosks represent a complex problem that only artificial intelligence can solve. Although ML helps us better predicting aftershock patterns, there is nothing magical about it. No deep learning was in fact needed with aftershock patterns already well described by only 2 features, which is in agreement with any visual inspection of the aftershock data.

The impact of these new results is twofold:

By bypassing stress computation entirely, using a simple ANN topology, and only 2 mainshock-rupture-based features, we can streamline the process of aftershock pattern prediction for future post-mainshock crisis management (i.e. process faster and more transparent);

In our in-depth analysis, we were not able to obtain performances higher than AUC = 86% despite using stress, or geometry and kinematics, and this for various hyperparameterizations. This infers that important features are missing. Observing past aftershock patterns, we suggest that performance could be further improved if the location of nearby faults were included. Testing this hypothesis is however out of the scope of the present project.

CHAPTER 7:

REFERENCES:

- https://www.blog.google/technology/ai/forecasting-earthquake-aftershock-locations-ai-assisted-science/
- https://www.nature.com/articles/s41586-018-0438-y#author-information
- https://www.nature.com/articles/d41586-018-06091-z
- https://phys.org/news/2018-08-ai-quake-aftershocks.html
- Båth, M. Lateral inhomogeneities of the upper mantle. *Tectonophysics* **2**, 483–514 (1965).
- Utsu, T. A statistical study on the occurrence of aftershocks. *Geophys. Mag*. **30**, 521–605 (1961).
- Parsons, T., Stein, R. S., Simpson, R. W. & Reasenberg, P. A. Stress sensitivity of fault seismicity: a comparison between limited-offset oblique and major strike-slip faults. *J. Geophys. Res*. **104**, 20183–20202 (1999).