Flood Tool

This package implements a flood risk prediction and visualization tool.

Installation Instructions

The flood tool package can be downloaded and installed as follows:

First the relevant github repoistry should be cloned:

```
git glone https://github.com/ese-msc-2021/ads-deluge-ouse.git
```

After moving to the relevant directory associated with the repository the following command can be run to install the flood tool package:

```
python setup.py install --user
```

Usage guide

The flood tool package can be used for both calcuation and visualisation of the annual flood risk for a set of user defined locations specified by postcode.

```
import flood_tool

tool = Tool()

risk_labels=tool.get_flood_class(self, postcodes, method=1, update=False)
```

This generates a series predicting flood probability classification for a collection of postcodes. the method argument (shown above) choses the classification alogorithm with which to train flood risk data with the following conversion: KNN: method=1, RandomForest: method=2, Neural Network : method=3)

```
annual_flood_risk= tool.get_annual_flood_risk(postcodes,risk_labels)
```

where the user can define postcodes to be a list of postcodes they wish to find corresponding data for. This generates a series of total annual flood risk estimates indexed by locations based on the following formula:

```
annual flood risk= 0.05x(total postcode property value)x(postcode flood probability)
```

The visualisations can be viewed by viewing the DataVisualisation.ipynd notebook which, when run shows maps visualising annual flood risk of postcodes.

Geodetic Transformations

For historical reasons, multiple coordinate systems exist in in current use in British mapping circles. The Ordnance Survey has been mapping the British Isles since the 18th Century and the last major retriangulation from 1936-1962 produced the Ordance Survey National Grid (otherwise known as **OSGB36**), which defined latitude and longitude for all points across the island of Great Britain [1]. For convenience, a standard Transverse Mercator projection [2] was also defined, producing a notionally flat 2D gridded surface, with gradations called eastings and northings. The scale for these gradations was identified with metres, which allowed local distances to be defined with a fair degree of accuracy.

The OSGB36 datum is based on the Airy Ellipsoid of 1830, which defines semimajor axes for its model of the earth, a and b, a scaling factor F_0 and ellipsoid height, H.

$$\begin{aligned} a &= 6377563.396, \\ b &= 6356256.910, \\ F_0 &= 0.9996012717, \\ H &= 24.7. \end{aligned}$$

The point of origin for the transverse Mercator projection is defined in the Ordnance Survey longitude-latitude and easting-northing coordinates as

$$\begin{aligned} \phi_0^{OS} &= 49^\circ \text{ north}, \\ \lambda_0^{OS} &= 2^\circ \text{ west}, \\ E_0^{OS} &= 400000m, \\ N_0^{OS} &= -100000m. \end{aligned}$$

More recently, the world has gravitated towards the use of satellite based GPS equipment, which uses the (globally more appropriate) World Geodetic System 1984 (also known as **WGS84**). This datum uses a different ellipsoid, which offers a better fit for a global coordinate system (as well as North America). Its key properties are:

$$\begin{aligned} a_{WGS} &= 6378137, , \\ b_{WGS} &= 6356752.314, \\ F_0 &= 0.9996. \end{aligned}$$

For a given point on the WGS84 ellipsoid, an approximate mapping to the OSGB36 datum can be found using a Helmert transformation [3],

$$\mathbf{x}^{OS} = \mathbf{t} + \mathbf{M}\mathbf{x}^{WGS}.$$

Here $\mathbf{x}$ denotes a coordinate in Cartesian space (i.e in 3D) as given by the (invertible) transformation

$$\begin{aligned} \nu &= \frac{aF_0}{\sqrt{1 - e^2 \sin^2(\phi^{OS})}} \\ x &= (\nu + H)\sin(\lambda)\cos(\phi) \\ y &= (\nu + H)\cos(\lambda)\cos(\phi) \\ z &= ((1 - e^2)\nu + H)\sin(\phi) \end{aligned}$$

and the transformation parameters are

$$\begin{aligned} \mathbf{t} &= \begin{pmatrix} -446.448 \\ 125.157 \\ -542.060 \end{pmatrix}, \\ \mathbf{M} &= \begin{bmatrix} 1+s & -r_3 & r_2 \\ r_3 & 1+s & -r_1 \\ -r_2 & r_1 & 1+s \end{bmatrix}, \\ s &= 20.4894 \times 10^{-6}, \\ \mathbf{r} &= [0.1502'', 0.2470'', 0.8421'']. \end{aligned}$$

Given a latitude, ϕ^{OS} and longitude, λ^{OS} in the OSGB36 datum, easting and northing coordinates, E^{OS} & N^{OS} can then be calculated using the following formulae (see “A guide to coordinate systems in Great Britain, Appendix C1):

$$
\begin{aligned}
\rho &= \frac{aF_0(1-e^2)}{\left(1-e^2\sin^2(\phi^{OS})\right)^{\frac{3}{2}}} \\
\eta &= \sqrt{\frac{\nu}{\rho}-1} \\
M &= bF_0\left[\left(1+n+\frac{5}{4}n^2+\frac{5}{4}n^3\right)(\phi^{OS}-\phi_0^{OS})\right. \\
&\quad -\left(3n+3n^2+\frac{21}{8}n^3\right)\sin(\phi-\phi_0)\cos(\phi^{OS}+\phi_0^{OS}) \\
&\quad +\left(\frac{15}{8}n^2+\frac{15}{8}n^3\right)\sin(2(\phi^{OS}-\phi_0^{OS}))\cos(2(\phi^{OS}+\phi_0^{OS})) \\
&\quad \left.-\frac{35}{24}n^3\sin(3(\phi-\phi_0))\cos(3(\phi^{OS}+\phi_0^{OS}))\right] \\
I &= M+N_0^{OS} \\
II &= \frac{\nu}{2}\sin(\phi^{OS})\cos(\phi^{OS}) \\
III &= \frac{\nu}{24}\sin(\phi^{OS})cos^3(\phi^{OS})(5-\tan^2(phi^{OS})+9\eta^2) \\
IIIA &= \frac{\nu}{720}\sin(\phi^{OS})cos^5(\phi^{OS})(61-58\tan^2(\phi^{OS})+\tan^4(\phi^{OS})) \\
IV &= \nu\cos(\phi^{OS}) \\
V &= \frac{\nu}{6}\cos^3(\phi^{OS})\left(\frac{\nu}{\rho}-\tan^2(\phi^{OS})\right) \\
VI &= \frac{\nu}{120}\cos^5(\phi^{OS})(5-18\tan^2(\phi^{OS})+\tan^4(\phi^{OS}) \\
&\quad +14\eta^2-58\tan^2(\phi^{OS})\eta^2) \\
E^{OS} &= E_0^{OS}+IV(\lambda^{OS}-\lambda_0^{OS})+V(\lambda-\lambda_0^{OS})^3+VI(\lambda^{OS}-\lambda_0^{OS})^5 \\
N^{OS} &= I+II(\lambda^{OS}-\lambda_0^{OS})^2+III(\lambda-\lambda_0^{OS})^4+IIIA(\lambda^{OS}-\lambda_0^{OS})^6
\end{aligned}
$$

The inverse transformation can be generated iteratively using a fixed point process:

1. Set $M=0$ and $\phi^{OS}=\phi_0^{OS}$.
2. Update $\phi_{i+1}^{OS}=\frac{N-N_0-M}{aF_0}+\phi_i^{OS}$
3. Calculate M using the formula above.
4. If abs$(N-N_0-M)>0.01mm$ go to 2, otherwise halt.

With M calculated we now improve our estimate of ϕ^{OS}. First calculate ν, ρ and η using our previous formulae. Next

$$
\begin{aligned}
VII &= \frac{\tan(\phi^{OS})}{2\rho\nu}, \\
VIII &= \frac{\tan(\phi^{OS})}{24\rho\nu^3}\left(5+3\tan^2(\phi^{OS})+\eta^2-9\tan^2(\phi^{OS})\eta^2\right), \\
IX &= \frac{\tan(\phi^{OS})}{720\rho\nu^5}\left(61+90\tan^2(\phi^{OS})+45\tan^4(\phi^{OS})\right), \\
X &= \frac{\sec\phi^{OS}}{\nu}, \\
XI &= \frac{\sec\phi^{OS}}{6\nu^3}\left(\frac{\nu}{\rho}+2\tan^2(\phi^{OS})\right), \\
XII &= \frac{\sec\phi^{OS}}{120\nu^5}\left(5+28\tan^2(\phi^{OS})+24\tan^4(\phi^{OS})\right), \\
XIIA &= \frac{\sec\phi^{OS}}{5040\nu^5}\left(61+662\tan^2(\phi^{OS})+1320\tan^4(\phi^{OS})+720\tan^6(\phi^{OS})\right).
\end{aligned}
$$

Finally, the corrected values for ϕ^{OS} and λ^{OS} are:

$$\phi_{\text{final}}^{OS} = \phi^{OS} - VII(E-E_0)^2 + VIII(E-E_0)^4 - IX(E-E_0)^6,$$
$$\lambda_{\text{final}}^{OS} = \lambda_0^{OS} + X(E-E_0) - XI(E-E_0)^3 + XII(E-E_0)^5 - XII(E-E_0)^7.$$

Function APIs

Python flood risk analysis tool

class flood_tool.**Tool**(*postcode_file=''*, *sample_labels=''*, *household_file=''*, *balanced_data=''*, *stations=''*, *rainfall_wet=''*, *rainfall_typ=''*)

Class to interact with a postcode database file.

Parameters:
- **postcode_file** (*str*, *optional*) – Filename of a .csv file containing geographic location data for postcodes.
- **sample_labels** (*str*, *optional*) – Filename of a .csv file containing sample data on property values and flood risk labels.
- **household_file** (*str*, *optional*) – Filename of a .csv file containing information on households by postcode.

find_rainfall_average_per_postcodes(*postcodes*)

Gives the average rainfall of the closest station for given postcodes for a typical day and a wet day

Parameters:
- **data** (*DataFrame*) – a dataframe with at least the coordiantes
- **(latitude** –
- **stations** (*longitude) for every postcodes and the nearest*) –
- **rainfall** (*which measures the*) –

Returns: A table with the mean rainfall for a typical and a wet day added for every postcode

Return type: Dataframe

Examples

get_annual_flood_risk(*postcodes*, *risk_labels=None*)

Return a series of estimates of the total property values of a collection of postcodes.

Risk is defined here as a damage coefficient multiplied by the value under threat multiplied by the probability of an event.

Parameters:
- **postcodes** (*sequence of strs*) – Sequence of postcodes.
- **risk_labels** (*pandas.Series (optional)*) – Series containing flood probability classifiers, as predicted by get_flood_probability.

Returns: Series of total annual flood risk estimates indexed by locations.

Return type: pandas.Series

get_closest_stations(*postcodes*)

Find the closest stations for a series of postcodes

Parameters: **postcodes** (*pandas DataFrame or Series*) – a list of postcodes

Returns: a table with the postcodes, their coordinates (latitude/longitude) and the nearest stations which measure the rainfall.

Return type: Pandas DataFrame

Examples

```
>>> tool = Tool()
>>> tool.get_closest_stations(['tS 21 3BT', 'SE23 3HW'])
         Latitude  Longitude Rain_stationReference
postcode
```

```
TS213BT   54.655084  -1.450910                032822
SE233HW   51.437903  -0.053763              289102TP
```

get_easting_northing(*postcodes*)

Get a frame of OS eastings and northings from a collection of input postcodes.

Parameters: **postcodes** (*sequence of strs*) – Sequence of postcodes.

Returns: DataFrame containing only OSGB36 easthing and northing indexed by the input postcodes. Invalid postcodes (i.e. not in the input unlabelled postcodes file) return as NaN.

Return type: pandas.DataFrame

Examples

```
>>> tool = Tool()
>>> tool.get_easting_northing(['ch6 0 0wA', 'TQ12 6WF',             'S   S7 5
           easting  northing
postcode
CH600WA   326913.0  381988.0
TQ126WF   281468.0   74769.0
SS7 5QA   577512.0  188009.0
>>> tool.get_easting_northing(['PL14 6SD', 'EX35 6HU', 'NE10 8NX',
           easting  northing
postcode
PL146SD   221409.0   73664.0
EX356HU   271919.0  149535.0
NE108NX   428923.0  560646.0
HR1 2JT   351754.0  239791.0
NE659DZ   416722.0  603595.0
```

get_flood_class(*postcodes, method=0, update=False*)

Generate series predicting flood probability classification for a collection of poscodes.

Parameters:

- **postcodes** (*sequence of strs*) – Sequence of postcodes.
- **method** (*int (optional)*) – optionally specify (via a value in self.get_flood_probability_methods) the classification method to be used.
- **update** (*boolean (optional)*) – If set to True, tune the model with a GridsearchCV method. If False, take the implemented values for the parameters.

Returns: Series of flood risk classification labels indexed by postcodes.

Return type: pandas.Series

static get_flood_class_methods()

Get a dictionary of available flood probablity classification methods.

Returns: Dictionary mapping classification method names (which have no inate meaning) on to an identifier to be passed to the get_flood_probability method.

Return type: dict

static get_house_price_methods()

Get a dictionary of available flood house price regression methods.

Returns: Dictionary mapping regression method names (which have no inate meaning) on to an identifier to be passed to the get_median_house_price_estimate method.

Return type: dict

get_lat_long(*postcodes*)

Get a frame containing GPS latitude and longitude information for a collection of of postcodes.

Parameters: **postcodes** (*sequence of strs*) – Sequence of postcodes.

Returns: DataFrame containing only WGS84 latitude and longitude pairs for the input postcodes. Invalid postcodes (i.e. not in the input unlabelled postcodes file) return as NAN.

Return type: pandas.DataFrame

Examples

```
>>> tool = Tool()
>>> tool.get_lat_long(['p O 3 5QP', 'IG10 2QH', 'tS 21 3BT'])
           Latitude  Longitude
postcode
PO3 5QP   50.821343  -1.052293
IG102QH   51.658602   0.068585
TS213BT   54.655084  -1.450910
>>> tool.get_lat_long(['LN11 7SW', 'BS30 5RY', 'HR3 5SX'])
           Latitude  Longitude
postcode
LN117SW   53.384302   0.161671
BS305RY   51.449035  -2.400703
HR3 5SX   52.095724  -3.039906
```

get_median_house_price_estimate(*postcodes, method=0, training=False*)

Generate series predicting median house price for a collection of poscodes.

Parameters:
- **postcodes** (*sequence of strs*) – Sequence of postcodes.
- **method** (*int (optional)*) – optionally specify (via a value in self.get_house_price_methods) the regression method to be used.

Returns: Series of median house price estimates indexed by postcodes.

Return type: pandas.Series

Examples

```
>>> tool = Tool()
>>> tool.get_median_house_price_estimate(['tr  13 8 e g',             'nn 14 4l
          Median House Price Prediction
Postcode
TR138EG                   482144.398964
NN144LU                   366709.156072
YO1 1RD                   358019.270575
>>> tool.get_median_house_price_estimate(['dE 2  3DA', 'YO62 4LS',
          Median House Price Prediction
Postcode
DE2 3DA                   166138.586162
YO624LS                   494297.375989
NN144LU                   366709.156072
```

get_total_value(*locations*)

Return a series of estimates of the total property values of a collection of postcode units or sectors.

Parameters: **locations** (*sequence of strs*) – Sequence of postcode units or sectors

Returns: Series of total property value estimates indexed by locations.

Return type: pandas.Series

house_price_KNN_training()

This function fits and trains KNNRegressor in new data is inputted. If the data is not changed, then get_median_house_price_estimate function will run on pre-defined hyperparameters for computational efficiency.

Parameters:
- **pandas.core.frame.DataFrame** –

- **Prices** (*DataFrame containing easting and northing and Median House*) –
- **training.** (*for model*) –

Returns:
- *KNeighborsRegressor model with assigned n_neighbors and weights*
- *hyperparameters based on GridSearch.*

house_price_estimate_en(*en*)

Function to be used in get_total_value, taking easting and northing values. :param pandas.core.frame.DataFrame of only easting and northing values:

Returns:

Return type: numpy.ndarray of predicted values of Median House Price Estimates

set_postcodes(*postcodes*)

Cleanins the sequence of input postcodes from typos and human errors

Parameters: **postcodes** (*sequence of strs*) – Sequence of postcodes.

Returns: Series containing cleaned postcodes. Invalid postcodes (i.e. not in the input unlabelled postcodes file) return as NaN.

Return type: pandas.core.series.Series

Examples

```
>>> tool = Tool()
>>> tool.set_postcodes(['cm13 2A N', 'T n 12 6 Y T', 'ls2 8lj',             'hU
0    CM132AN
1    TN126YT
2    LS2 8LJ
3    HU179RH
Name: postcode, dtype: object
>>> tool.set_postcodes(['IP1 5AH', 'DH8 7NE', 'DY5 3NN', 'HU3 1GB'])
0    IP1 5AH
1    DH8 7NE
2    DY5 3NN
3    HU3 1GB
Name: postcode, dtype: object
```

flood_tool.average_interval(*value*)

Function that compute the average value in case the mm (rainfall) or mASD (rivers) values are presented as ‘XXX | XXX’ .

Parameters: **value** (*string*) – the raw value presented in the data

Returns: the average value and converted in float type

Return type: float

flood_tool.get_easting_northing_from_gps_lat_long(*phi, lam, rads=False*)

Get OSGB36 easting/northing from GPS latitude and longitude pairs.

Parameters:
- **phi** (*float/arraylike*) – GPS (i.e. WGS84 datum) latitude value(s)
- **lam** (*float/arraylike*) – GPS (i.e. WGS84 datum) longitude value(s).
- **rads** (*bool (optional)*) – If true, specifies input is is radians.

Returns:
- *numpy.ndarray* – Easting values (in m)
- *numpy.ndarray* – Northing values (in m)

Examples

```
>>> get_easting_northing_from_gps_lat_long([55.5], [-1.54])
(array([429157.0]), array([623009]))
```

References

Based on the formulas in “A guide to coordinate systems in Great Britain”.

See also https://webapps.bgs.ac.uk/data/webservices/convertForm.cfm

flood_tool.**get_gps_lat_long_from_easting_northing**(*east*, *north*, *rads=False*, *dms=False*)

Get GPS latitude and longitude pairs from OSGB36 easting/northing.

Parameters:
- **east** (*float/arraylike*) – OSGB36 easting value(s) (in m).
- **north** (*float/arraylike*) – OSGB36 easting value(s) (in m).
- **rads** (*bool (optional)*) – If true, specifies ouput is is radians.
- **dms** (*bool (optional)*) – If true, output is in degrees/minutes/seconds. Incompatible with rads option.

Returns:
- *numpy.ndarray* – GPS (i.e. WGS84 datum) latitude value(s).
- *numpy.ndarray* – GPS (i.e. WGS84 datum) longitude value(s).

Examples

```
>>> get_gps_lat_long_from_easting_northing([429157], [623009])
(array([55.5]), array([-1.540008]))
>>> get_gps_lat_long_from_easting_northing([429157], [623009],
                                           rads=False, dms=True)
(array([55.50001947, 30.00208694, 59.92798956]),
        array([-1.54001445, 27.69329395, 41.77411121]))
```

References

Based on the formulas in “A guide to coordinate systems in Great Britain”.

See also https://webapps.bgs.ac.uk/data/webservices/convertForm.cfm

flood_tool.**get_level_data_aggregate**(*data*, *stations*)

Compute for each stations the standard deviation and the mean of the rivers height evolution. Add the stations coordinates.

Parameters:
- **data** (*DataFrame*) – the processed rivers data set
- **stations** (*DataFrame*) – the stations dataset with their coordinates

Returns: the final data Frame with the aggregates for each stations.

Return type: DataFrame

flood_tool.**get_level_data_processed**(*data*)

Filter the data to get only the rivers height measurements. And process them to get rid of outliers, missing values or duplicates.

Parameters: **data** (*DataFrame*) – the raw data

Returns: the processed data

Return type: (DataFrame)

flood_tool.**get_rainfall_data_processed**(*data*)

Filter the data to get only the rainfall measurements. And process them to get rid of outliers, missing values or duplicates.

Parameters: **data** (*DataFrame*) – the raw data

Returns: the processed data

Return type: (DataFrame)

flood_tool.**get_rainfall_per_hour_data**(*data*, *stations*)

Generate the sum of the rainfall per hour per station. Merge the stations informations (coordinates). Classify the stations acocridng to the amount of rainfall per hour. Btw 0-2 mm : slight Btw 2-4 mm : moderate Btw 4-50 mm : heavy Over 50 mm : violent

Parameters:
- **data** (*DataFrame*) – the processed data filtered with the rainfall
- **stations** (*for each*) – dataset with the coordinates
- **stations** –

Returns: The rainfall data set with stations coordinates and rainfall classification added.

Return type: DataFrame

`flood_tool.`**`get_station_data`**`(`*`filename, station_reference`*`)`

Return readings for a specified recording station from .csv file.

Parameters:
- **filename** (*str*) – filename to read
- **station_reference** – station_reference to return.
- **get_station_data('resources/wet_day.csv)** (*>>> data =*) –

`flood_tool.`**`knn_classifier`**`(`*`x_train, y_train, data, y_data, update_parameter=False`*`)`

Create, tune and train a KNN classifier with the splitted data set (postcodes_samples.csv).

Parameters:
- **X_train** (*pandas Dataframe*) – the training part of the dataset
- **y_train** (*pandas DataFrame or Serie*) – the response label of the
- **data** (*training set*) – the complete original dataset
- **y** (*pandas DataFrame*) – the response label for the complete original
- **update_parameter** (*dataset*) – if true make a grid search cv
- **parameter** (*to find the best*) –

Returns: a trained KNN classifier with the best parameter

Return type: sklearn.neighbors._classification.KNeighborsClassifier

`flood_tool.`**`nn_classifier`**`(`*`x_train, y_train, data, y_data, update_parameter=False`*`)`

Create, tune and train a neural network multi layer perceptron classifier with the splitted data set (postcodes_samples.csv).

Parameters:
- **X_train** (*pandas Dataframe*) – the training part of the dataset
- **y_train** (*pandas DataFrame or Serie*) – the response label of the
- **data** (*training set*) – the complete original dataset
- **y** (*pandas DataFrame*) – the response label for the complete original
- **update_parameter** (*dataset*) – if true make a grid search cv
- **parameter** (*to find the best*) –

Returns: a trained KNN classifier with the best parameter

Return type: sklearn.neighbors._classification.KNeighborsClassifier

`flood_tool.`**`plot_circle`**`(`*`lat, lon, radius, map=None, **kwargs`*`)`

Plot a circle on a map (creating a new folium map instance if necessary).

Parameters:
- **lat** (*float*) – latitude of circle to plot (degrees)
- **lon** (*float*) – longitude of circle to plot (degrees)
- **radius** (*float*) – radius of circle to plot (m)
- **map** (*folium.Map*) – existing map object

Returns:

Return type: Folium map object

Examples

```
>>> import folium
>>> armageddon.plot_circle(52.79, -2.95, 1e3, map=None)
```

flood_tool.**rf_classifier**(*x_train*, *y_train*, *data*, *y_data*, *update_parameter=False*)

Create, tune and train a RandomForest classifier with the splitted data set (postcodes_samples.csv).

Parameters:
- **X_train** (*pandas Dataframe*) – the training part of the dataset
- **y_train** (*pandas DataFrame or Serie*) – the response label of the
- **data** (*training set*) – the complete original dataset
- **y** (*pandas DataFrame*) – the response label for the complete original
- **update_parameter** (*dataset*) – if true, make a gid search cv
- **model** (*to tune the*) –

Returns: a trained RF classifier with the best parameter

Return type: sklearn.ensemble._forest.RandomForestClassifier

flood_tool.**smote_func**(*raw_data_file_path*)

This function is used to deal with the imbalanced data set of the multivariate classification problem, And make each category have the same number of examples, that is, the number of examples in the largest category.

Parameters: **raw_data_file_path** (*str.*) – Filename of a .csv file containing features (geographic location) and label data(i.e. risk level).

Returns: DataFrame consists of ‘easting’, ‘northing’ and flood risk classification labels, this is a new balanced dataset.

Return type: pandas.DataFrame

References

[1] A guide to coordinate systems in Great Britain, Ordnance Survey

[2] Map projections - A Working Manual, John P. Snyder, https://doi.org/10.3133/pp1395

[3] Computing Helmert transformations, G Watson, http://www.maths.dundee.ac.uk/gawatson/helmertrev.pdf