Chapter 6 - Analysis of similarity of measured data

November 10, 2020

This notebook computes all similarity measures between cadastral and tessellation layers. Generates figures 6.23, 6.24, 6.26.

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from scipy import stats
import geopandas as gpd
from tqdm import tqdm
from sklearn.metrics import mean_squared_error
import math
import scipy.stats as sp
import matplotlib
```

[2]:
```
path = 'data/'
```

[ ]:
```
# set default seaborn style
sns.set()

cadastre = gpd.read_file(path + 'cadastre/blg_cadvals.shp')
tess10 = gpd.read_file(path + 'tessellation/10_tessellation.shp')
tess15 = gpd.read_file(path + 'tessellation/15_tessellation.shp')
tess20 = gpd.read_file(path + 'tessellation/20_tessellation.shp')
tess25 = gpd.read_file(path + 'tessellation/25_tessellation.shp')
tess30 = gpd.read_file(path + 'tessellation/30_tessellation.shp')
tess40 = gpd.read_file(path + 'tessellation/40_tessellation.shp')
tess50 = gpd.read_file(path + 'tessellation/50_tessellation.shp')
tess60 = gpd.read_file(path + 'tessellation/60_tessellation.shp')
tess70 = gpd.read_file(path + 'tessellation/70_tessellation.shp')
tess80 = gpd.read_file(path + 'tessellation/80_tessellation.shp')
tess90 = gpd.read_file(path + 'tessellation/90_tessellation.shp')
tess100 = gpd.read_file(path + 'tessellation/100_tessellation.shp')
tess150 = gpd.read_file(path + 'tessellation/150_tessellation.shp')
tess200 = gpd.read_file(path + 'tessellation/200_tessellation.shp')
tess300 = gpd.read_file(path + 'tessellation/300_tessellation.shp')
```

```
[ ]: characters = ['area', 'lal', 'circom', 'shapeix', 'rectan', 'fractal',
                   'orient', 'freq', 'car', 'gini_area', 'gini_car', 'Reach']
```

```
[ ]: buffers = {10: tess10, 15: tess15, 20: tess20, 25: tess25, 30: tess30, 40:␣
     ↪tess40, 50: tess50, 60: tess60, 70: tess70, 80: tess80, 90: tess90,
                100: tess100, 150: tess150, 200: tess200, 300: tess300}
     keys = [10, 15, 20, 25, 30, 40, 50, 60, 70, 80, 90, 100, 150, 200, 300]
```

```
[ ]: cadastre.rename(index=str, columns={'uID_left': 'uID'}, inplace=True)
```

```
[ ]: singleuids = pd.read_csv('data/single_uids.csv')
     singles = singleuids['2'].to_list()
```

0.1 normality test

```
[ ]: for ch in characters:
         k2, p = sp.normaltest(cadastre[ch])
         alpha = 1e-3
         print("p = {:g}".format(p))

         if p < alpha:  # null hypothesis: x comes from a normal distribution
             print(ch + ": The null hypothesis can be rejected (non-normal␣
     ↪distribution)")
         else:
             print(ch + ": The null hypothesis cannot be rejected")
```

0.2 root mean squared deviation

`#tes = tes.loc[tes['uID'].isin(singles)]` can be used to generate analysis for single/multi building plots individually

```
[ ]: df1 = pd.DataFrame(keys, columns=['buffer'])
     rmsde = df1
     for ch in characters:
         for b in buffers:
             try:
                 key = '{b}_{ch}'.format(b=b, ch=ch)
                 tes = buffers[b]
                 #tes = tes.loc[tes['uID'].isin(singles)]
                 true = tes[ch]
                 cad = cadastre
                 #cad = cad.loc[cad['uID'].isin(singles)]
                 prediction = cad[ch]
                 min = true.min() if true.min() < prediction.min() else prediction.
     ↪min()
                 max = true.max() if true.max() > prediction.max() else prediction.
     ↪max()
```

```
            # iqr = stats.iqr(true) if stats.iqr(true) > stats.iqr(prediction)␣
↪else stats.iqr(prediction)
            value = math.sqrt(mean_squared_error(true, prediction)) / (max -␣
↪min)  # normalised
        except Exception:
            value = np.nan
        if b == 10:
            list = pd.Series(value, index=[b])
        else:
            list = list.append(pd.Series(value, index=[b]))

    rmsde[ch] = list.values
#rmsde.to_csv('Results_all_rsmd.csv')
```

[ ]:

```
sns.set_style('ticks', {'xtick.bottom': False, 'ytick.left': True})
sns.set_context(context='paper', font_scale=1, rc=None)
colors = [(72,129,185), (123,173,210), (115,109,170), (158,155,196),␣
↪(188,189,217), (218,218,234), (224,131,173), (197,57,51), (230,156,155),
          (85,160,92), (135,187,125), (142,60,33), (201,102,45), (231,155,71),␣
↪(248,218,152), (252,248,216)]

# create a color palette
# palette = plt.get_cmap('tab20')
for index, col in enumerate(colors):
    list = []
    for idx, rgb in enumerate(col):
        rgb = rgb / 255
        list.append(rgb)
    colors[index] = tuple(list)
palette = matplotlib.colors.ListedColormap(colors, name='from_list', N=None)
# multiple line plot
num = 0
for column in rmsde.drop(['buffer'], axis=1):
    plt.plot(rmsde['buffer'], rmsde[column].fillna(method='ffill'), marker='',␣
↪color=palette(num), linewidth=1, alpha=0.9, label=column)
    num += 1
sns.despine(offset=10, trim=False, left=True, bottom=True)
plt.xlim(1, 300)
plt.axvline(x=100, color='r', linestyle='--', lw=1)

# Add legend
lgd = plt.legend(bbox_to_anchor=(1.05, 1), loc=2, ncol=1,)
plt.grid(True, which='major', axis='x')
plt.ylabel("Normalised RMSD")
plt.xlabel("Buffer distance")
plt.title("Normalised root squared mean deviation")
```

```
new_labels = ['area', 'longest axis length', 'circular compactness', 'shape␣
↪index', 'rectangularity', 'fractal dimension',
              'orientation', 'CAR', 'frequency', 'Gini of area', 'Gini of CAR',␣
↪'Reach']
for t, l in zip(lgd.texts, new_labels):
    t.set_text(l)
#plt.savefig(path + 'Results_multi_rsmd.png',
#             dpi=300, bbox_extra_artists=(lgd,), bbox_inches='tight')
plt.gcf().clear()
```

0.3 spearman rho

`#tes = tes.loc[tes['uID'].isin(singles)]` can be used to generate analysis for single/multi building plots individually

[ ]:
```
# correlation spearman rho
df1 = pd.DataFrame(keys, columns=['buffer'])
spearman_rho = df1
for ch in characters:

    for b in buffers:
        try:
            key = '{b}_{ch}'.format(b=b, ch=ch)
            tes = buffers[b]
            #tes = tes.loc[~tes['uID'].isin(singles)]
            cad = cadastre
            #cad = cad.loc[~cad['uID'].isin(singles)]
            value = sp.spearmanr(cad[ch], tes[ch])[0]  # get correlation␣
↪coefficient r
            p = sp.spearmanr(cad[ch], tes[ch])[1]
        except Exception:
            value = np.nan
            p = np.nan

        if b == 10:
            list = pd.Series(value, index=[b])
            p_list = pd.Series(p, index=[b])
        else:
            list = list.append(pd.Series(value, index=[b]))
            p_list = p_list.append(pd.Series(p, index=[b]))

    p_column = 'p_{ch}'.format(ch=ch)
    spearman_rho[ch] = list.values
    # spearman_rho[p_column] = p_list.values

#spearman_rho.to_csv('Results_multi_spearman.csv')
```

```
[ ]: # plot
# style
sns.set_style('ticks', {'xtick.bottom': False, 'ytick.left': True,})
sns.set_context(context='paper', font_scale=1, rc=None)
colors = [(72,129,185), (123,173,210), (115,109,170), (158,155,196),␣
↪(188,189,217), (218,218,234), (224,131,173), (197,57,51), (230,156,155),
          (85,160,92), (135,187,125), (142,60,33), (201,102,45), (231,155,71),␣
↪(248,218,152), (252,248,216)]

# create a color palette
# palette = plt.get_cmap('tab20')
for index, col in enumerate(colors):
    list = []
    for idx, rgb in enumerate(col):
        rgb = rgb / 255
        list.append(rgb)
    colors[index] = tuple(list)
palette = matplotlib.colors.ListedColormap(colors, name='from_list', N=None)

# multiple line plot
num = 0
for column in spearman_rho.drop(['buffer'], axis=1):
    plt.plot(spearman_rho['buffer'], spearman_rho[column].
↪fillna(method='ffill'), marker='', color=palette(num), linewidth=1, alpha=0.
↪9, label=column)
    num += 1
sns.despine(offset=10, trim=False, left=True, bottom=True)
plt.xlim(1, 300)
plt.axvline(x=100, color='r', linestyle='--', lw=1)


# Add legend
lgd = plt.legend(bbox_to_anchor=(1.05, 1), loc=2, ncol=1,)
plt.grid(True, which='major', axis='x')
plt.ylim(0, 1.05)
# plt.xlim(10)
plt.ylabel("Spearman's rho")
plt.xlabel("Buffer distance")
plt.title("Correlations")
new_labels = ['area', 'longest axis length', 'circular compactness', 'shape␣
↪index', 'rectangularity', 'fractal dimension',
              'orientation', 'CAR', 'frequency', 'Gini of area', 'Gini of CAR',␣
↪'Reach']
for t, l in zip(lgd.texts, new_labels):
    t.set_text(l)
```

```
#plt.savefig(path + 'Results_multi_spearman.png', dpi=300,␣
↪bbox_extra_artists=(lgd,), bbox_inches='tight')
plt.gcf().clear()
```

0.4 moran deviation

#tes = tes.loc[tes['uID'].isin(singles)] can be used to generate analysis for single/multi building plots individually

[ ]:
```
files = [
    tess10,
    tess15,
    tess20,
    tess25,
    tess30,
    tess40,
    tess50,
    tess60,
    tess70,
    tess80,
    tess90,
    tess100,
    tess150,
    tess200,
    tess300,
    cadastre,
]

for f in files:
    f.rename(
        index=str,
        columns={"m_gini_are": "m_gini_area", "p_gini_are": "p_gini_area"},
        inplace=True,
    )
```

[ ]:
```
# moran deviation
moran = pd.DataFrame(keys, columns=["buffer"])
for ch in tqdm(characters):
    max = 0
    cadlist = []
    rch = "p_{}".format(ch)
    mch = "m_{}".format(ch)
    for idx, row in cadastre.iterrows():
        if row[rch] <= 0.01:
            max = max + 1
            cadlist.append(row[mch])
        else:
```

```
            cadlist.append(None)
    cadastre["Moran_sig"] = cadlist
    for b in tqdm(buffers):
        try:
            buflist = []
            for idx, row in buffers[b].iterrows():
                if row[rch] <= 0.01:
                    buflist.append(row[mch])
                else:
                    buflist.append(None)
            buffers[b]["Moran_sig"] = buflist

            compare = cadastre["Moran_sig"] == buffers[b]["Moran_sig"]
            value = sum(compare) / max
        except Exception:
            value = np.nan
        if b == 10:
            list = pd.Series(value, index=[b])
        else:
            list = list.append(pd.Series(value, index=[b]))

    moran[ch] = list.values
#moran.to_csv(
#    "Results2_all_accu.csv"
#)
```

[ ]:
```
#cad_s = cadastre.loc[~cadastre["uID_left"].isin(singles)]  # used to filter␣
 ↪analysis for single-building, multt-building and all plots
for ch in tqdm(characters):
    max = 0
    cadlist = []
    rch = "p_{}".format(ch)
    mch = "m_{}".format(ch)
    for idx, row in cad_s.iterrows():
        if row[rch] <= 0.01:
            max = max + 1
            cadlist.append(row[mch])
        else:
            cadlist.append(None)
    cad_s["Moran_sig"] = cadlist
    for b in tqdm(buffers):
        try:
            buflist = []
            tes = buffers[b]
            tes_s = tes.loc[~tes["uID"].isin(singles)]
            for idx, row in tes_s.iterrows():
                if row[rch] <= 0.01:
```

```
                    buflist.append(row[mch])
                else:
                    buflist.append(None)
            tes_s["Moran_sig"] = buflist

            compare = cad_s["Moran_sig"] == tes_s["Moran_sig"]
            value = sum(compare) / max
        except Exception:
            value = np.nan
        if b == 10:
            list = pd.Series(value, index=[b])
        else:
            list = list.append(pd.Series(value, index=[b]))

    moran[ch] = list.values
#moran.to_csv(
#     "Results2_m_accu.csv"
#)
```

[ ]:
```
sns.set_style("ticks", {"xtick.bottom": False, "ytick.left": True})
sns.set_context(context="paper", font_scale=1, rc=None)
colors = [
    (72, 129, 185),
    (123, 173, 210),
    (115, 109, 170),
    (158, 155, 196),
    (188, 189, 217),
    (218, 218, 234),
    (224, 131, 173),
    (197, 57, 51),
    (230, 156, 155),
    (85, 160, 92),
    (135, 187, 125),
    (142, 60, 33),
    (201, 102, 45),
    (231, 155, 71),
    (248, 218, 152),
    (252, 248, 216),
]

# create a color palette
# palette = plt.get_cmap('tab20')
for index, col in enumerate(colors):
    list = []
    for idx, rgb in enumerate(col):
        rgb = rgb / 255
        list.append(rgb)
```

```
    colors[index] = tuple(list)
palette = matplotlib.colors.ListedColormap(colors, name="from_list", N=None)


# multiple line plot
num = 0
# moran[[col for col in moran.columns if '_m' in col]]
for column in moran.drop(["buffer"], axis=1):
    plt.plot(
        moran["buffer"],
        moran[column].fillna(method="ffill"),
        marker="",
        color=palette(num),
        linewidth=1,
        alpha=0.9,
        label=column,
    )
    num += 1
# Add legend
lgd = plt.legend(bbox_to_anchor=(1.05, 1), loc=2, ncol=1)
plt.grid(True, which='major', axis='x')
sns.despine(offset=10, trim=False, left=True, bottom=True)
plt.xlim(1, 300)
plt.ylim(0, 1.05)
plt.axvline(x=100, color="r", linestyle="--", lw=1)

plt.ylabel("Accuracy score")
plt.xlabel("Buffer distance")
plt.title("Local spatial autocorrelation accuracy")
new_labels = [
    "area",
    "longest axis length",
    "circular compactness",
    "shape index",
    "rectangularity",
    "fractal dimension",
    "orientation",
    "CAR",
    "frequency",
    "Gini of area",
    "Gini of CAR",
    "Reach",
]
for t, l in zip(lgd.texts, new_labels):
    t.set_text(l)
plt.savefig(
    path + "Results2_m_accu.png",
```

```
    dpi=300,
    bbox_extra_artists=(lgd,),
    bbox_inches="tight",
)
plt.gcf().clear()
```