svd

May 30, 2022

[1]:
```
import os

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import plotly.express as px
import plotly.graph_objects as go
```

[2]:
```
import plotly.io as pio
pio.renderers.default = "plotly_mimetype+notebook+vscode"
```

[3]:
```
# function used in case we would like to select desired timestamp
def trunc_df(df):
    start = pd.to_datetime('2020-07-01 19:30:00')
    end = pd.to_datetime('2020-08-01 19:30:00')
    ret_df = df.loc[start:end]
    return ret_df

def create_folder(folder_name):
    if not os.path.exists(folder_name):
        print(f'Creating folder {folder_name}')
        os.mkdir(folder_name)

# to always have the newest plot versions, delete file before creating new one
def remove_file_if_exists(file_path):
    if os.path.exists(file_path):
        os.remove(file_path)
```

[4]:
```
# svd global variables and creating dirs
SVD_FOLDER_PATH = 'svd_plots'
create_folder(SVD_FOLDER_PATH)

# SVD_COLUMN = 'humidity'
# SVD_COLUMN = 'tempC'
SVD_COLUMN = 'windspeedKmph'
```

```
# ANALYSIS_TYPE = 'full'
ANALYSIS_TYPE = 'partial'

SVD_FOLDER_PATH = f'{SVD_FOLDER_PATH}/{SVD_COLUMN}'
create_folder(SVD_FOLDER_PATH)

# create reconstuction dir
SVD_RECONSTUCTION_FOLDER_PATH = f'{SVD_FOLDER_PATH}/reconstuction'
create_folder(SVD_RECONSTUCTION_FOLDER_PATH)
```

```
Creating folder svd_plots/windspeedKmph
Creating folder svd_plots/windspeedKmph/reconstuction
```

0.1 Prepare Data for SVD Analysis

[5]:
```
# function to create csv file used for analysis
def create_svd_df(column_value):
    FULL_DATA_FILENAME = 'data/data.csv.gz'
    OUTPUT_FILENAME = f'data/svd/data_svd_{column_value}_{ANALYSIS_TYPE}.csv'

    columns_to_read = ['date_time', 'city', column_value]
    df = pd.read_csv(FULL_DATA_FILENAME, usecols=columns_to_read,␣
 ↪compression='gzip')
    unique_towns = sorted(list(df['city'].unique()))

    new_index = pd.date_range(
        start=df.iloc[0]['date_time'],
        end=df.iloc[-1]['date_time'],
        freq='3h'
    )
    data = np.array(df[column_value])
    data = data.reshape(len(new_index), len(unique_towns))

    ret_df = pd.DataFrame(
        data=data,
        columns=unique_towns,
        index=new_index,
        dtype=float
    )
    ret_df.to_csv(OUTPUT_FILENAME)
```

[6]:
```
GEOLOCATION_FILENAME = 'data/geo_position.csv'
df_geolocation = pd.read_csv(GEOLOCATION_FILENAME).sort_values(by='CITY')
unique_towns = sorted(list(df_geolocation['CITY'].unique()))  # get unique␣
 ↪names of towns ordered by name

print(f'Doing analisys for {SVD_COLUMN}')
```

```
SVD_DATA_FILENAME = f'data/svd/data_svd_{SVD_COLUMN}_{ANALYSIS_TYPE}.csv'

if not os.path.exists(SVD_DATA_FILENAME):
    print(f'Creating svd file for {SVD_COLUMN}')
    create_svd_df(column_value=SVD_COLUMN)
    print('File created!')
```

```
Doing analisys for windspeedKmph
Creating svd file for windspeedKmph
File created!
```

0.2 SVD

[7]:
```
print(f'Loading svd file for {SVD_COLUMN}')
df_svd = pd.read_csv(SVD_DATA_FILENAME, index_col=0)
df_svd.index = pd.to_datetime(df_svd.index)

# df_svd = trunc_df(df_svd)
svd_A = np.array(df_svd)

# build matrix U, S, V
svd_U, svd_S, svd_V = np.linalg.svd(svd_A, full_matrices=False)
```

```
Loading svd file for windspeedKmph
```

0.2.1 Precision of SVD Resconstrucion

[8]:
```
# function to calculate and plot precision of svd reconstruction
def svd_precision(svd_S):
    SVD_PRECISION_FILENAME = f'{SVD_FOLDER_PATH}/
 ↪{SVD_COLUMN}_svd_precision_{ANALYSIS_TYPE}.png'
    remove_file_if_exists(SVD_PRECISION_FILENAME)

    fig = go.Figure(
        data=[go.Bar(
                x=np.arange(np.size(svd_S)),
                y=np.cumsum(svd_S / np.sum(svd_S))
            )
        ]
    )

    fig.update_layout(
        title_text='SVD Precision',
        title_x=0.5,
        xaxis_title='Rank',
        yaxis_title='Precision'
    )
    fig.update_yaxes(range=[0.5, 1])
```

```
    fig.write_image(SVD_PRECISION_FILENAME)
    fig.show()
```

[9]:
```
# Calculating svd reconstruction precision
svd_precision(svd_S)
```

0.2.2 Full Reconstruction & Lower Rank Reconstruction

[10]:
```
# full reconstruction - matrix svd_Ar
svd_Ar = np.dot(svd_U * svd_S, svd_V)
print(f'Diff: {np.mean(np.abs(svd_A - svd_Ar))}')

# lower rank reconstruction - matrix svd_Ar
k = 5
svd_Ar = np.dot(svd_U[:,:k] * svd_S[:k], svd_V[:k, :])

print(f'Diff reconstruction: {np.mean(np.abs(svd_A - svd_Ar))}')
```

```
Diff: 4.3518192208483686e-14
Diff reconstruction: 0.8253357196115587
```

0.2.3 Average SVD Error

[11]:
```
# function to calculate and plot average error of svd for k=n
def svd_average_error(svd_A, svd_Ar, k, unique_towns):
    SVD_AVG_ERR_FILENAME = f'{SVD_FOLDER_PATH}/
 ↪{SVD_COLUMN}_svd_avg_err_{ANALYSIS_TYPE}.png'
    remove_file_if_exists(SVD_AVG_ERR_FILENAME)

    svd_err = np.average(np.abs(svd_A - svd_Ar), axis=0)
    asix_range = np.arange(0, len(unique_towns))

    fig = go.Figure(data=[go.Bar(x=asix_range,
                                 y=svd_err
                                 )])

    fig.update_layout(
        title_text='SVD Average Error', title_x=0.5,
        yaxis_title=f'Average error of reconstruction with rank k={k}',
        xaxis=dict(tickmode='array',
            tickvals=asix_range,
            ticktext=unique_towns
        )
    )
    fig.update_xaxes(tickangle=90)
    fig.write_image(SVD_AVG_ERR_FILENAME)
    fig.show()
```

[12]:
```
# Calculating average error
svd_average_error(
    svd_A=svd_A,
    svd_Ar=svd_Ar,
    k=k,
    unique_towns=unique_towns
)
```

0.2.4 Dates to Concept - SVD_U

[13]:
```
# function to plot dates to concept for k=n
def svd_dates_to_concept(k, index, svd_U):
    SVD_DTC_FILENAME = f'{SVD_FOLDER_PATH}/
 ↪{SVD_COLUMN}_svd_dates_to_concept_{ANALYSIS_TYPE}.png'
    remove_file_if_exists(SVD_DTC_FILENAME)
    all_plots = []
    for i in range(k):
        all_plots.append(go.Scatter(x=index, y=svd_U[:, i], name=f'k={i}'))

    fig = go.Figure(data=all_plots)
    fig.write_image(SVD_DTC_FILENAME)
    fig.update_layout(xaxis=dict(rangeslider=dict(visible=True)))
    fig.show()
```

[14]:
```
# dates to concept
svd_dates_to_concept(
    k=k,
    index=df_svd.index,
    svd_U=svd_U
)
```

0.2.5 Towns to Concept - SVD_V

[15]:
```
# function to plot towns to concept for k=n
def svd_towns_to_concept(k, svd_V, unique_towns):
    SVD_TTC_FILENAME = f'{SVD_FOLDER_PATH}/
 ↪{SVD_COLUMN}_svd_towns_to_concept_{ANALYSIS_TYPE}.png'
    remove_file_if_exists(SVD_TTC_FILENAME)

    asix_range = np.arange(0, len(unique_towns))
    all_plots = []
    for i in range(k):
        all_plots.append(go.Bar(x=asix_range, y=svd_V[i, :], name=f'{i}'))

    fig = go.Figure(data=all_plots)
```

```
    fig.update_layout(
        title_text=f'Towns to Concept for k={k}',
        title_x=0.5,
        xaxis=dict(
                tickmode='array',
                tickvals=asix_range,
                ticktext=unique_towns
        )
    )

    fig.update_xaxes(tickangle=90)
    fig.write_image(SVD_TTC_FILENAME)
    fig.show()
```

[16]:
```
# towns to concept
svd_towns_to_concept(
    k=k,
    svd_V=svd_V,
    unique_towns=unique_towns
)
```

0.2.6 SVD Maps

[17]:
```
# plot map with values from SVD_V (towns to concept)
def plot_svd_map(vector, k, data_geo):
    SVD_MAP_FILENAME = f'{SVD_FOLDER_PATH}/
 ↪{SVD_COLUMN}_svd_map_k{k}_{ANALYSIS_TYPE}.png'
    remove_file_if_exists(SVD_MAP_FILENAME)

    data_geo['VALUES'] = vector
    px.set_mapbox_access_token(open(".mapbox_token").read())

    fig = px.scatter_mapbox(
        data_geo,
        size = [2] * len(data_geo.index),
        lat="LAT",
        lon="LNG",
        color="VALUES",
        hover_name="CITY",
        color_continuous_scale=px.colors.cyclical.Phase,
    )

    fig.update_layout(
        height = 700,
        margin = {
            'l':5,
            'r':5,
```

```
            't':5,
            'b':5,
        },
        autosize=True,
        mapbox = {
            'style': "open-street-map",
            'zoom': 7.5
        }
    )
    fig.write_image(SVD_MAP_FILENAME)
    fig.show()
```

[18]:
```
# plot maps
for i in range(k):
    print(f'Ploting map for k = {i}')
    plot_svd_map(
        vector=svd_V[i, :],
        k=i,
        data_geo=df_geolocation.copy()
    )
```

```
Ploting map for k = 0

Ploting map for k = 1

Ploting map for k = 2

Ploting map for k = 3

Ploting map for k = 4
```

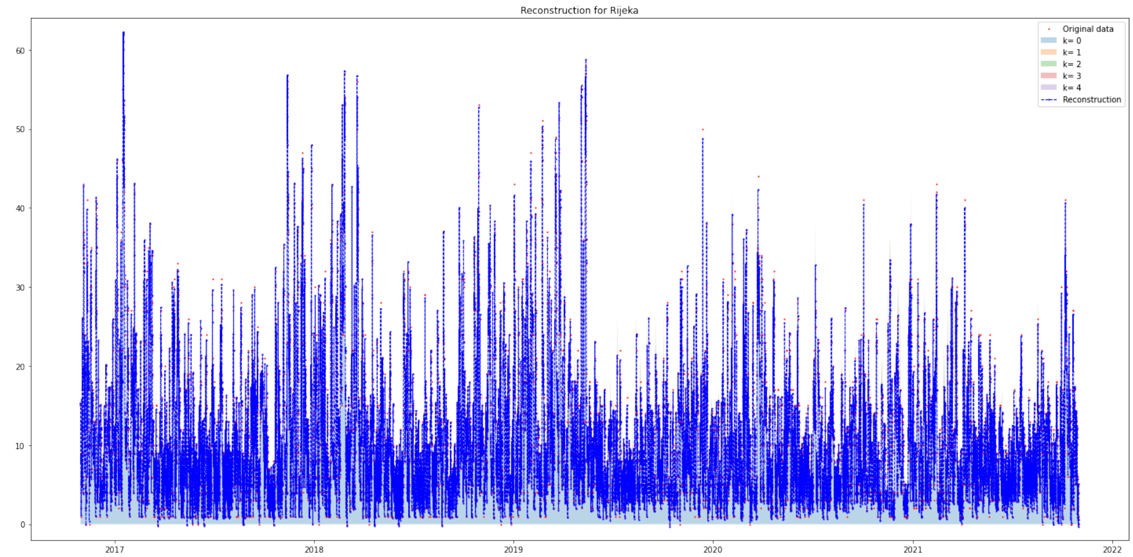
0.2.7 SVD Reconstruction

[19]:
```
# reconstruct temperatures of unique towns using svd
def svd_reconstruct(unique_towns, df_svd, svd_A, svd_U, svd_S, svd_V, k):
    print(f'Saving reconstructins to {SVD_RECONSTUCTION_FOLDER_PATH}')
    for iloc, location in enumerate(unique_towns):
        SVD_RESONSTRUCTION_FILENAME = f'{SVD_RECONSTUCTION_FOLDER_PATH}/
↪{SVD_COLUMN}_svd_reconstruction_plot_{location}_{ANALYSIS_TYPE}.png'
        remove_file_if_exists(SVD_RESONSTRUCTION_FILENAME)

        fig = plt.figure(figsize=(20, 10), tight_layout=True)
        legend_handles = []
        legend_labels = []

        plt_orig, = plt.plot(df_svd[location], marker='o', ls='', c='r', ms=1)
        legend_handles.append(plt_orig)
        legend_labels.append('Original data')
```

```
        a_cum = np.zeros(svd_A.shape[0])
        for i in range(k):
            a_k = np.dot(svd_U[:, i] * svd_S[i], svd_V[i, iloc])
            flbtw_k = plt.fill_between(df_svd.index, a_cum, a_cum + a_k,␣
↪alpha=0.3, label=f'k= {i}')
            legend_handles.append(flbtw_k)
            legend_labels.append(f'k= {i}')
            a_cum += a_k

        plt_recon, = plt.plot(df_svd.index, a_cum, marker='s', ls='--', c='b',␣
↪lw=1, ms=1)
        legend_handles.append(plt_recon)
        legend_labels.append('Reconstruction')

        plt.legend(legend_handles, legend_labels)
        plt.ylim(df_svd[location].min() - 2, df_svd[location].max() + 2)
        fig.savefig(SVD_RESONSTRUCTION_FILENAME, dpi=90)

        if location == 'Rijeka':
            plt.title(f'Reconstruction for {location}')
            plt.show()

        plt.close(fig)
```

[20]:
```
# SVD reconstruction temperature
svd_reconstruct(
    unique_towns=unique_towns,
    df_svd=df_svd,
    svd_A=svd_A,
    svd_U=svd_U,
    svd_S=svd_S,
    svd_V=svd_V,
    k=k
)
```

```
Saving reconstructins to svd_plots/windspeedKmph/reconstuction
```



1 Export to HTML

[21]:
```
# save notebook before nbconvert
import IPython
```

[22]:
```
%%javascript
IPython.notebook.save_notebook()
```

```
<IPython.core.display.Javascript object>
```

[23]:
```
# export notebook results to HTML
jupyter_out_filename = f'{SVD_COLUMN}_svd_{ANALYSIS_TYPE}'
!jupyter nbconvert --output-dir 'output' --output {jupyter_out_filename}␣
 ↪--to=HTML svd.ipynb
!jupyter nbconvert --output-dir 'output' --output {jupyter_out_filename}␣
 ↪--to=pdf svd.ipynb

jupyter_out_filename_no_code = f'{SVD_COLUMN}_svd_no_code_{ANALYSIS_TYPE}'
!jupyter nbconvert --output-dir 'output' --output␣
 ↪{jupyter_out_filename_no_code} --no-input --to=HTML svd.ipynb
!jupyter nbconvert --output-dir 'output' --output␣
 ↪{jupyter_out_filename_no_code} --no-input --to=pdf svd.ipynb
```

```
[NbConvertApp] Converting notebook svd.ipynb to HTML
[NbConvertApp] Writing 8437987 bytes to output/windspeedKmph_svd_partial.html
```