```
import pickle
import pandas as pd
import numpy as np
from sklearn.gaussian_process.kernels import Matern, ConstantKernel
from sklearn.gaussian_process import GaussianProcessRegressor
from bayes_opt import BayesianOptimization

import sys
sys.path.insert(0, '../utils')
from plot_format import *
import seaborn as sns
from seaborn import apionly as sns


x_seq_noch = np.array([0.3752, 0.9508, 0.7323, 0.5991, 0.0010, 0.4732, 0.4671, 0.4405, 0.4592, 0.1932, 0.8384]).reshape((-1,1))
y_seq_noch = np.array([1904.44167, 4166.38284, 2886.16776, 2030.58708, 3511.18782, 1815.88346, 1793.71586, 1814.04904,
1771.51534, 2799.16994, 3407.47170])
y_seq_noch_rmseall = np.array([66.020235327172429**2,
                144.79492384667014**2,
                124.62328813252397**2,
                102.69758173952037**2,
                28.717401415334244**2,
                87.365020299648322**2,
                84.530973668499286**2,
                78.364352523865179**2,
                81.609857644938558**2,
                81.609857644938558**2,
                35.364707692321197**2,
                132.23033368723819**2])

x_single_noch = np.array([0.3752, 0.9508, 0.7323, 0.5991, 0.0010, 0.1697, 0.1043, 0.7481, 0.8186, 0.6034, 0.1125, 0.1323, 0.1366
]).reshape((-1,1))
y_single_noch = np.array([1186.80634, 3947.30585, 3073.07991, 2268.21285, 954.12632, 1065.12579, 1112.03419, 3185.04042,
3678.08562, 2337.37609, 1012.44412, 1005.95753, 1097.95087])

x_seq_ch = np.array([0.3746, 0.9507, 0.7320, 0.5987, 0.0001, 0.8499, 1.0000, 0.7918, 0.8982, 0.9132, 0.2320, 0.4835]).reshape((-
1,1))
y_seq_ch = np.array([0.77175, 0.79790, 0.79473, 0.77754, 0.61321, 0.79675, 0.80114, 0.79117, 0.80315, 0.79742, 0.73338,
0.77551])

x_last_ch = np.array([0.3746, 0.9507, 0.7320, 0.5987, 0.0001, 0.5362, 0.6414, 0.8423, 0.8101, 0.8728, 0.8289, 0.1884, 1.0000,
0.4590, 0.7801, 0.2844]).reshape((-1,1))
y_last_ch = np.array([0.80453, 0.80610, 0.80711, 0.80831, 0.79043, 0.80764, 0.80705, 0.80970, 0.80967, 0.80859, 0.80897,
0.80071, 0.80896, 0.80697, 0.81044, 0.80200])

second_dense = {
        'w_scale': np.array([0.3746, 0.9507, 0.7320, 0.5987, 0.0001, 0.1983, 0.2557, 0.2832, 0.2675, 0.0945, 0.4806,  0.3220, 0.2284,
         0.2417, 0.1628, 1.0000, 0.1757, 0.1449, 0.2384, 0.8370, 0.6628, 0.5380]),
        'rmse': np.array([
            60.137878942450072,
            77.722036745092268,
            72.879331251873978,
            67.062420522234277,
            61.657153972481723,
            57.631117787828856,
            56.713102187481006,
            56.834944670880205,
            57.488396982161127,
            65.327687191224172,
            65.182131239745033,
            58.24648629796301,
            56.54127003769475l,
            55.588719014202631,
            56.43249292417061,
            79.851982900902101,
            57.243657935485729,
            56.633961360532581,
            57.327917657711311,
            75.887397934508854,
            68.695413363368999,
            65.946070181104531]),
        'churnacc': np.array([
            0.71487603305785119,
            0.71242252066115708,
            0.71448863636363635,
            0.70945247933884292,
            0.67484504132231404,
            0.72675619834710747,
            0.71435950413223137,
            0.71655475206611574,
            0.72236570247933884,
            0.73024276859504134,
            0.71552169421487599,
            0.71694214876033058,
            0.72223657024793386,
            0.71823347107438018,
            0.72068698347107744,
```

```
        0.70803202479338845,
        0.72778925619834711,
        0.72210743801652888,
        0.72404442148760328,
        0.71280991735537191,
        0.71397210743801653,
        0.71552169421487599]),
    'churnauc': np.array([
        0.7792210720970616,
        0.76933765457035874,
        0.77342995451014707,
        0.77548862306969646,
        0.76870632137982753,
        0.7871430878039829,
        0.78436558201282924,
        0.78344705976695317,
        0.78326333370294687,
        0.79096527440496356,
        0.77696498808050141,
        0.78322482327894249,
        0.78562990553093992,
        0.78618666755431565,
        0.78890725429034592,
        0.76653687681109761,
        0.78807973048323054,
        0.7883635332150426,
        0.78450978896071499,
        0.76564011349515482,
        0.77505312024856476,
        0.7777320263853682]),
    'churnrecall': np.array([
        0.39971600993965212,
        0.45970891018814342,
        0.43876464323748671,
        0.36279730209442668,
        0.42598509052183176,
        0.46432374866879661,
        0.38729144479943201,
        0.38835640752573658,
        0.4174653887113951,
        0.56975505857294995,
        0.42988995385161521,
        0.3919062832800852,
        0.42350017749378771,
        0.39723109691160807,
        0.44302449414270501,
        0.51082712105076322,
        0.47142350017749379,
        0.44053958111466096,
        0.44905928292509761,
        0.47497337593184241,
        0.39758608448704297,
        0.41959531416400425]),
    'concordance': np.array([
        0.81052730830811615,
        0.80691224818420137,
        0.80675021193378538,
        0.80987616373148386,
        0.79853516545793668,
        0.8113337006233946,
        0.81228918365501601,
        0.81087573262151835,
        0.81147181921024902,
        0.81223363231596823,
        0.80987732804018864,
        0.81054202990631496,
        0.81200751961870032,
        0.81239448452363372,
        0.81170584525989997,
        0.80780923950394445,
        0.81162205450125102,
        0.81121975624269771,
        0.81186523714817305,
        0.80693673813509237,
        0.80930750746647495,
        0.81062870183565705])
    }

def plot_vs(res, width=1, height=None):
    w_scale = res['w_scale']
    # rmse = res['rmse'] / res['rmse'].max()
    # churn_acc = res['churn_acc'] / res['churn_acc'].max()
    # churn_auc = res['churn_auc'] / res['churn_auc'].max()
    # churn_recall = res['churn_recall'] / res['churn_recall'].max()
    # concordance = res['concordance'] / res['concordance'].max()
```

```
    fig, ax = newfig(width, height)

    res['rmse'] = -res['rmse']

    for key in ['rmse', 'churnacc', 'churnauc', 'churnrecall', 'concordance']:
        ax.scatter(w_scale, normalise(res[key]), label=key)

    box = ax.get_position()
    ax.set_position([box.x0, box.y0, box.width * 0.8, box.height])

    ax.legend(loc='upper left', bbox_to_anchor=(1,1))
    # ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))

    # fig.tight_layout()
    fig.show()

def normalise(arr):
    arr -= arr.min()
    arr /= arr.max()
    return arr



def posterior_1d(bo, x, steps):
    # gp = GaussianProcessRegressor(kernel=Matern(nu=2.5)*ConstantKernel(1), n_restarts_optimizer=25, normalize_y=True)
    gp=bo.gp
    gp.fit(bo.X[:steps], bo.Y[:steps])
    mu, sigma = gp.predict(x, return_std=True)
    return mu, sigma

def plot_gp_1d(steps=10, width=1, height=None):
    # bo = pickle.load(open(model.RESULT_PATH+'bayes_opt{}.pkl'.format(opt), 'rb'))

    bounds = {'w_scale': (.001, 1.)}
    bo = BayesianOptimization(lambda w_scale: 0, bounds)

    # bo.maximize(init_points=1, n_iter=0, acq='ucb', kernel=Matern())
    bo.X = x_last_ch
    bo.Y = y_last_ch

    # bo.gp = GaussianProcessRegressor(kernel=Matern(nu=2.5)*ConstantKernel(1), n_restarts_optimizer=25)
    # bo.gp.set_params(normalize_y=True)
    bo.gp = GaussianProcessRegressor(kernel=Matern(nu=2.5), n_restarts_optimizer=25)


    x = np.linspace(.001, 1, 10000).reshape(-1, 1)

    fig, ax = newfig(width, height)

    mu, sigma = posterior_1d(bo, x, steps)
    ax.plot(bo.X[:steps].flatten(), bo.Y[:steps], 'D', markersize=8, label=u'Observations', color='r')
    ax.plot(x, mu, '--', color='k', label='Prediction')

    # plt.fill_between(x.reshape(-1), mu+sigma, mu-sigma, alpha=0.1)
    ax.fill(np.concatenate([x, x[::-1]]),
              np.concatenate([mu - 1.9600 * sigma, (mu + 1.9600 * sigma)[::-1]]),
        alpha=.2, fc='black', ec='None', label=r'95₩% confidence interval')

    ax.set_xlim((0.001, 1))
    ax.set_ylim((None, None))
    # ax.set_ylabel(r'MSE (returning users)')
    ax.set_ylabel(r'Concordance')
    ax.set_xlabel(r'$w$')

    ax.legend(loc=4)
    fig.tight_layout()
    fig.show()


# def posterior(bo, res, grid):
def posterior(bo, grid):
    # xmin, xmax = 0, 5000
    # bo.gp.fit(bo.X[:steps], bo.Y[:steps])
    # bo.gp.fit(res['X'], res['Y'])
    bo.gp.fit(bo.X, bo.Y)
    mu, sigma = bo.gp.predict(grid, return_std=True)
    return mu, sigma

def plot_gp(steps=32, width=1, height=None):
    # bo = BayesianOptimization(lambda x: 0, bounds)
    bo = pickle.load(open('../../results/rnn/bayes_opt/bayes_opt_rnn_5.pkl', 'rb'))
    bo.X = bo.X[:,[1,0]][:steps]
    bo.Y = -bo.Y[:steps]

    x = np.arange(1, 151)
    y = np.arange(1, 101)
    grid = [(i,j) for j in y for i in x]
```

```python
    fig, ax = newfig(width, height)

    # mu, sigma = posterior(bo, res, grid)
    mu, sigma = posterior(bo, grid)
    mu = np.log(mu)
    # return x,y,mu,grid

    # cs = ax.contourf(x,y,mu.reshape((100,150)), np.geomspace(750, 2550, 20), cmap=plt.cm.viridis_r)
    cs = ax.contourf(x,y,mu.reshape((100,150)),np.geomspace(np.log(750), np.log(2550), 20), cmap=plt.cm.viridis_r)

    # samples_x = [x[0] for x in res['X']]
    # samples_y = [x[1] for x in res['X']]
    samples_x = [x[0] for x in bo.X]
    samples_y = [x[1] for x in bo.X]
    sc = ax.scatter(samples_x, samples_y, label='Samples', color='C3', s=5)
    # ax.plot(bo.X[:steps].flatten(), bo.Y[:steps], 'D', markersize=8, label=u'Observations', color='r')
    # ax.plot(x, mu, '--', color='k', label='Prediction')
    ax.set_xlim((1, 150))
    ax.set_ylim((1, 100))

    ax.set_xlabel('Number of active days')
    ax.set_ylabel('Number of LSTM cells')
    ax.legend(loc=1)
    cbar = fig.colorbar(cs)
    cbar.ax.set_ylabel('Posterior mean (MSE)')
    xs = list(map(lambda x: float(x.get_text()[1:-1]), cbar.ax.get_yticklabels()))
    cbar.ax.set_yticklabels(list(map(lambda x: int(np.ceil(np.exp(x))), xs)))
    sc.set_clip_on(False)
    fig.tight_layout()
    fig.show()

def plot_gp_var(steps=32, width=1, height=None):
    # bo = BayesianOptimization(lambda x: 0, bounds)
    bo = pickle.load(open('../../results/rnn/bayes_opt/bayes_opt_rnn_5.pkl', 'rb'))
    bo.X = bo.X[:,[1,0]][:steps]
    bo.Y = -bo.Y[:steps]

    x = np.arange(1, 151)
    y = np.arange(1, 101)
    grid = [(i,j) for j in y for i in x]

    fig, ax = newfig(width, height)

    # mu, sigma = posterior(bo, res, grid)
    mu, sigma = posterior(bo, grid)
    # return x,y,mu,grid

    # cs = ax.contourf(x,y,sigma.reshape((100,150)), 20)
    cs = ax.contourf(x,y,sigma.reshape((100,150)), 20, cmap=plt.cm.viridis_r)

    # samples_x = [x[0] for x in res['X']]
    # samples_y = [x[1] for x in res['X']]
    samples_x = [x[0] for x in bo.X]
    samples_y = [x[1] for x in bo.X]
    sc = ax.scatter(samples_x, samples_y, label='Samples', color='C3', s=5)
    # ax.plot(bo.X[:steps].flatten(), bo.Y[:steps], 'D', markersize=8, label=u'Observations', color='r')
    # ax.plot(x, mu, '--', color='k', label='Prediction')
    ax.set_xlim((1, 150))
    ax.set_ylim((1, 100))

    ax.set_xlabel('Number of active days')
    ax.set_ylabel('Number of LSTM cells')
    ax.legend(loc=1)
    cbar = fig.colorbar(cs)
    cbar.ax.set_ylabel('Posterior variance')
    sc.set_clip_on(False)
    fig.tight_layout()
    fig.show()

def plot_gp_multiple(model, opt='', steps=[2,8,16], width=1, height=None):
    bo = pickle.load(open(model.RESULT_PATH+'bayes_opt{}.pkl'.format(opt), 'rb'))

    x = np.linspace(0, 5000, 10000).reshape(-1, 1)

    ax = {}
    fig, ax[steps[0]] = newfig(width, height, 131)
    ax[steps[1]] = fig.add_subplot(132)
    ax[steps[2]] = fig.add_subplot(133)

    # p = {i: {'mu': mu, 'sigma': sigma} for mu, sigma in posterior(bo, x, i) for i in steps}
    p = {i: {'mu': mu, 'sigma': sigma} for i, (mu, sigma) in [(i, posterior(bo, x, i)) for i in steps]}
    obs = {i: ax[i].plot(
                        bo.X[:i].flatten(),
                        bo.Y[:i],
                        '.', markersize=8, label=u'Observations', color='r')[0]
        for i in steps}
```

```
    pred = {i: ax[i].plot(x, p[i]['mu'], '--', color='k', label='Prediction')[0] for i in steps}

    conf = {i: ax[i].fill(
                np.concatenate([x, x[::-1]]),
                np.concatenate(
                    [p[i]['mu'] - 1.9600 * p[i]['sigma'],
                    (p[i]['mu'] + 1.9600 * p[i]['sigma'])[::-1]]),
                alpha=.2, fc='black', ec='None', label=r'95₩% confidence interval')[0]
        for i in steps}

    ax[steps[1]].set_yticklabels([])
    ax[steps[2]].set_yticklabels([])
    ax[steps[0]].set_ylabel('Concordance')
    [ax[i].set_xlabel(r'$₩gamma$') for i in steps]
    [ax[i].set_xlim((0, 5000)) for i in steps]
    [ax[i].set_ylim((.73,.84)) for i in steps]

    fig.legend(handles=[obs[steps[0]], pred[steps[0]], conf[steps[0]]], labels=[r'Observations', r'Prediction', r'95₩% confidence
    interval'], loc='upper center', ncol=3, framealpha=1, bbox_to_anchor=(0.55, 0.91))
    fig.tight_layout()
    fig.show()


def plot_grid_search(width=1, height=None):
    # res = {'penalties': space, 'scores': {k: [d[k] for d in scores] for k in scores[0]}}
    res = pickle.load(open(CoxChurnModel.RESULT_PATH+'grid_search_21.pkl', 'rb'))

    scores = {'churn_auc': 'Churn AUC',
              'churn_acc': 'Churn Accuracy',
              'rmse_days': 'RMSE',
              'concordance': 'Concordance'}
    x = res['penalties']
    y = res['scores']

    fig, ax = newfig(width, height)

    for k in scores:
        ax.plot(x, np.array(y[k])/np.max(y[k]), label=scores[k])

    ax.legend()
    fig.tight_layout()
    fig.show()
```