PersonalizedGP

PersonalizedGP is a package for building personalized Gaussian process (GP) models in python. The package is built upon the GPflow library, which implements Gaussian process models using TensorFlow.

This package uses a modified radial basis function kernel as the learning algorithm. At its core, PersonalizedGP applies the following formula to data points to observe their similarities.

$$\lambda e^{\frac{\left(x-x^{'}\right)^2}{2\sigma}}$$

λ is the variance parameter, and σ is the lengthscales parameter.

GP Models

PersonalizedGP outputs predictions from four models.

Source Model (sGP)

sGP uses the full training dataset on an auto-regressive GP to learn a forecasting function (Eq. (1)). The model initially places a prior on functions to give rise to a joint prior, then the model is optimized such that kernel parameters minimize the negative log-marginal likelihood.

After training, given an input feature vector, sGP outputs mean and variance of the output distribution (Eqs. (2–3)).

Target Model (tGP)

tGP builds a GP model from observed data of a target subgroup. Since the dataset is limited in size, it cannot effectively learn GP parameters due to risk of overfitting. Thus, tGP uses parameters from sGP, which were learned from the full training dataset. Here, the covariance matrix is constantly updated as more observed data becomes available.

After training, given an input feature vector, tGP outputs mean and variance of the output distribution (Eqs. (6–7)).

Personalized Model (pGP)

pGP extends the approach of domain adaptive GPs to personalize sGP to a target subgroup. Using the full training dataset and observed data of the target subgroup, the posterior distribution is obtained, which is then used to obtain the prior of future data. This prior corrects the posterior distribution to account for observed data of the target subgroup.

After training, given an input feature vector, pGP outputs mean and variance of the output distribution, which is a combination of the sGP prediction and a correction term (Eqs. (4–5)). The correction term shifts the mean towards the distribution of the target subgroup and improves the model's confidence by reducing predictive variance.

Joint Model (pGP+tGP)

pGP+tGP computes the geometric mean of the results outputted by pGP and tGP. This model inherits the lower error rate of tGP and the trend of decreasing error with visit number of pGP, resulting in best overall performance.

Getting Started

PersonalizedGP allows for prediction of mean and variance based on source, target, and personalized models upon building, training, and optimization of a source model with GPflow. Both the input feature vector and output labels can be multi-dimensional.

For example, mean and variance predictions can be made from source, target, personalized, and joint models in a few lines of code:

```
from call_pgp import *
from evaluation_metrics import *

import gpflow
import numpy as np

#generate random X and Y values
X = np.random.uniform(low = 0, high = 1, size = (1000, 4))
Y = np.random.uniform(low = 0, high = 1, size = (1000, 1))

#generate list of training and adaptation data indices
tr_ind_source = list(range(0, 800))
adapt_ind = list(range(800, 1000))
```

```
#extract source features and labels
x_s = X[[tr_ind_source]]
y_s = Y[[tr_ind_source]]

#extract adaptation data
x_a = X[[adapt_ind]][:-1, :]
y_a = Y[[adapt_ind]][:-1, :]

#extract test data
xtest = X[[adapt_ind]]
ytest = Y[[adapt_ind]]

#create RBF kernel
d = np.shape(X)[1]
k = gpflow.kernels.RBF(d)

#TRAINING AND PREDICTING SOURCE MODEL

#create GP model m with x_s, y_s variables and kernel k
m = gpflow.models.GPR(x_s, y_s, kern = k)

#initialize hyperparameters
m.likelihood.variance = np.exp(2*np.log(np.sqrt(0.1*np.var(y_s))))
max_x = np.amax(x_s, axis=0)
min_x = np.amin(x_s, axis=0)
m.kern.lengthscales = np.array(np.median(max_x - min_x))
m.kern.variance = np.var(y_s)

#optimize model
m.compile()
opt = gpflow.train.ScipyOptimizer()
opt.minimize(m)

#call personalized GP method
out = call_pgp(m, x_s, y_s, x_a, y_a, xtest, k)

g_t = ytest

m_s = out['source model mu']
m_a = out['adapted model mu']
m_t = out['target model mu']
m_j = out['joint model mu']

s_s = out['source model sigma']
s_a = out['adapted model sigma']
s_t = out['target model sigma']
s_j = out['joint model sigma']

#error - source model
e_s = calcMAE(g_t, m_s)
w_s = calcWES(g_t, m_s, s_s)
print('SOURCE MODEL - MEAN ABSOLUTE ERROR:', e_s)
print('SOURCE MODEL - WEIGHTED ERROR SCORE:', w_s)
```

```
#error - adaptation model
e_a = calcMAE(g_t, m_a)
w_a = calcWES(g_t, m_a, s_a)
print('ADAPTATION MODEL - MEAN ABSOLUTE ERROR:', e_a)
print('ADAPTATION MODEL - WEIGHTED ERROR SCORE:', w_a)

#error - target model
e_t = calcMAE(g_t, m_t)
w_t = calcWES(g_t, m_t, s_t)
print('TARGET MODEL - MEAN ABSOLUTE ERROR:', e_t)
print('TARGET MODEL - WEIGHTED ERROR SCORE:', w_t)

#error - joint model
e_j = calcMAE(g_t, m_j)
w_j = calcWES(g_t, m_j, s_j)
print('JOINT MODEL - MEAN ABSOLUTE ERROR:', e_j)
print('JOINT MODEL - WEIGHTED ERROR SCORE:', w_j)
```

Installation

PersonalizedGP depends on GPflow. Instructions for installation can be found here.

Comment [Office1]: Update link

To install PersonalizedGP, clone the repository:

```
$ git clone <insert link here>
```

Comment [Office2]: Insert link

The requirements can be installed via pip as follows (use sudo if necessary):

```
$ pip install -r requirements.txt
```

To demo, run the `pgp_demo.py` script.

Running Tests

To use PersonalizedGP, save training data as a CSV file with input data in the first n columns and label in the last column. The data extraction code in pgp_demo.py may be used to extract CSV data as numpy arrays.

Citations

- GPflow
- GPy