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.

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

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

Citation

If you use this code in your research, please cite the following publications:

Utsumi, Y., Rudovic, O., Peterson, K., Guerrero, R., Picard, R. "Personalized Gaussian Processes for Forecasting of Alzheimer's Disease Assessment Scale-Cognition Sub-Scale (ADAS-Cog13)." The 40th International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). May 2018.

Available via arXiv.

Peterson, K., Rudovic, O., Guerrero, R., Picard, R. "Personalized Gaussian Processes for Future Prediction of Alzheimer's Disease Progression," NIPS Workshop on Machine Learning for Healthcare, Long Beach, CA, December 2017.

Available via arXiv.

GP Models

PersonalizedGP outputs predictions from four models. Reference this paper for corresponding equations.

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, a simple personalized Gaussian process model can be built and trained in just a few lines of code:

```
from data.data_generator import *
from utils import process_data
from GP import personalizedGP

import numpy as np
import os
import gpflow

# Generate and process data
generate_demo_data()

CURRENT_DIR = os.path.dirname(os.path.abspath(__file__))
DATA_CSV_DIR = os.path.join(CURRENT_DIR, 'data/ts_demo_data.csv')

data = np.genfromtxt(DATA_CSV_DIR, delimiter=',')
IDs, X, Y, indicators = data[:, :1], data[:, 1:-8], data[:, -8:-4],
data[:, -4:]
unique_IDs = np.unique(list(map(lambda x:int(x[0]), IDs)))

for i in range(10):
    te_IDs = [unique_IDs[i]]
    tr_IDs = np.setdiff1d(unique_IDs, te_IDs)

    X_tr, Y_tr, X_te, Y_te, ind_te, ID_te = process_data(X, Y,
        indicators, IDs, tr_IDs, te_IDs)

    # Create RBF kernel and GP model instance
    k = gpflow.kernels.RBF(input_dim=X_tr.shape[1])
    pGP = personalizedGP(X=X, Y=Y, kernel=k)

    # Train and predict on test patient
    for te in te_IDs:
        te_rows = np.where(ID_te == te)[0]

        X_ad_patient = X_te[te_rows][:-1, :]
        Y_ad_patient = Y_te[te_rows][:-1, :]
        X_te_patient = X_te[te_rows]

        pGP.train(X_tr=X_tr, Y_tr=Y_tr, X_ad=X_ad_patient,
                Y_ad=Y_ad_patient, new_patient=True)
        m_ad_patient, s_ad_patient = pGP.predict(X_te=X_te_patient)
```

Installation

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

To install PersonalizedGP, clone the repository:

```
$ git clone https://github.com/yuriautsumi/PersonalizedGP.git
```

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, format and process data using our helper functions.

Data Format

Data should be in a CSV file and formatted with columns as follows: unique patient ID, features, label (corresponding to same visit as features), and indicator (1 – true label, 0 – missing label). Each patient data should be in chronological order, with earliest visits at the top.

To reformat the data for predicting T time steps ahead, given the previous visit label, pass the file through `ts_formatter`, which can be imported from `data.ts_formatter`.

Data Processing

Data can be processed into training and testing data using our `process_data` function, which can be imported from `utils`.

Citations

- GPflow
- GPy