COMP9418 Assignment 3

zid: z5137606 z5127440
name: Zhen YUAN YiFan ZHAO

Abstract:

In this project, the problem is that we have a multi-class problem as we have more than two classes. So we used Gaussian Process Models to classify chunk tags. Besides that, another problem is that we need to encode the targets for ease of computation. Because of the D-dimensional binary feature vector for each word/token in the sentence, we need to decrease the number of dimensional. In the step of choosing a model, we choose the gpflow.kernels.RBF() function as the kernel function and use gpflow.models.SVGP() as the model. After building the model and calculate the accuracy, we have about 80% precisely and could almost obtain the classification task.

Introduction:

In this project, we have got preprocessed and extracted features from the dataset. However, the D-dimensional binary feature vector for each word/token in the sentence is too big so that we should use the way to decrease the number of dimensional. After the dimensional is so small that we can put it in to computation, we use them to

Model:

We use Gaussian Process Model to predict the labels and calculate the accuracy. First we know that the input of the model is

Supervised Learning Problems

– Inputs: $\mathbf{x} = \{\mathbf{x}_n\}_{n=1}^N$ Labels: $\mathbf{y} = \{\mathbf{y}_n\}_{n=1}^N$

And we all know that Factorization of GP prior over Q latent functions is :

$$f_j \sim \mathcal{GP}(0, \kappa_j(\cdot,\cdot)) \rightarrow p(\mathbf{f}|\boldsymbol{\theta}_0) = \prod_{j=1}^{Q} p(\mathbf{f}_{\bullet j}|\boldsymbol{\theta}_0) = \prod_{j=1}^{Q} \mathcal{N}(\mathbf{f}_{\bullet j}; \mathbf{0}, \mathbf{K}_j)$$

- Covariance function of jth GP
- All NxQ latent function values
- Covariance Hyper-parameters
- All N latent values for function j
- Covariance matrix induced by κ_j

And the probability of each label is

$$p(\mathbf{y}|\mathbf{f},\boldsymbol{\theta}_1) = \prod_{n=1}^{N} p(\mathbf{y}_n|\mathbf{f}_{n\bullet},\boldsymbol{\theta}_1)$$

Cond. Likelihood parameters

Observations and latent functions for data-point n

For multi-class classification, we can see Q classes as Q independent GP priors $p(f_j)$, and j = 1, ….Q. For each GP can have different covariance. Also we can Softmax likelihood as – p(y=j)*α*exp(f_j) .

After we make the prediction we should evaluate the presentation and the model. We know that ELL term is a model fit. We can evaluate how well the (samples from the) posterior explains the observations.

As Expected Log Likelihood(ELL) Term, ELL and its gradients can be estimated using expectations over univariate Gaussian distributions.

$$q_{k(n)} = q_{k(n)}(\mathbf{f}_{n\bullet}|\boldsymbol{\lambda}_{k(n)})$$

$$\mathbb{E}_{q_k}[\log p(\mathbf{y}|\mathbf{f})] = \sum_{n=1}^{N} \mathbb{E}_{q_{k(n)}}[\log p(\mathbf{y}_n|\mathbf{f}_{n\bullet})]$$

$$\nabla_{\boldsymbol{\lambda}_{k(n)}} \mathbb{E}_{q_{k(n)}}[\log p(\mathbf{y}_n|\mathbf{f}_{n\bullet})] = \mathbb{E}_{q_{k(n)}} \nabla_{\boldsymbol{\lambda}_{k(n)}} \log q_{k(n)}(\mathbf{f}_{n\bullet}|\boldsymbol{\lambda}_{k(n)}) \log p(\mathbf{y}_n|\mathbf{f}_{n\bullet})$$

Inference:

First we use PCA to decrease the dimensional feature vector for each word/token. The code is below:

```
reducer = IncrementalPCA(n_components=10, batch_size=500)
reducer.fit(file_x_list)
file_x_list = reducer.transform(file_x_list)
```

After that we use training set to build the model. This is the code for building the model:

```
kenel = gpflow.kernels.RBF(file_x_list.shape[1], lengthscales=1.0, variance=2.0)
m = gpflow.models.SVGP(
    np.float64(file_x_list), np.float64(file_y_list.A.T), kern=kenel, likelihood=gpflow.likelihoods.MultiClass(23),
    Z=np.float64(file_x_list[::3]), num_latent=23, whiten=True, q_diag=True)
```

We use the gpflow.kernels.RBF() function as kernel function and use gpflow.models.SVGP() function as the model. After building the model we use ScipyOptimizer() function to make the value minimize. It is the code below:

```
opt = gpflow.train.ScipyOptimizer()
opt.minimize(m, maxiter=10000)
```

By using gpflow.models.SVGP() function, this function has including the comparing posterior inference and make the prediction.

After building the model, we should use some dataset(which we didn't use to training the model before) to make the predict.

Parameter Estimation:

However after we finish the model and make the prediction. We need to apply cross validation in training and testing(only use part of training set to training and other to test the accuracy). After prediction, we compare the predicted label with the y_test label and calculate the accuracy.

We use the part of first 70% in training set to training the model and use part of other 30% data to make the predict and compare with the truth value. If the accuracy is too lower, then we mixture the dataset and select part of another 70% to training and use part of other 30% to test until the accuracy is about 80%.

Results:

After cross validation we use:

$$\mathrm{ER} = 1 - \frac{1}{N}\sum_{i=1}^{N}\frac{1}{T_i}\sum_{k=1}^{T_i}\mathbb{I}[\hat{\mathbf{y}}_k^{(i)} = \mathbf{y}_k^{(i)}],$$

$$\mathrm{MNLP} = -\frac{1}{N}\sum_{i=1}^{N}\frac{1}{T_i}\sum_{k=1}^{T_i}\sum_{j=1}^{C}\mathbb{I}[\mathbf{y}_k^{(i)} = j]\log p_{\mathrm{model}}(\mathbf{y}_k^{(i)}|\mathbf{x}^{(i)}),$$

To calculate the ER and MNLP and accuracy.

After cross validation we finally have almost above 80% accuracy in the testing set.

This is the some testing accuracy photos:

```
predict-label [2 1 0 2 2 2 6 2 2 1 0 0 2 2 3 4 4 4 0 2 1 0 1 0 2 2 6]
predict-label [0 2 6 0 2 1 0 2 6 2 6 0 3 6 0 2 1 0 6 2 2 3 4 0 3 4 0 6]
org-label [0 2 6 0 2 1 0 2 2 2 6 3 0 3 0 2 5 0 0 2 2 3 4 0 3 4 0 6]
the acc is  0.7857142857142857
0.8317593967746915
```

The last line is the whole accuracy in testing set. In addition, by analysis the data we find that the testing set and training set should be select randomly, because may be some words only appearance in the last some sentence. So if we want to have better result we should make select the training set randomly.

References:

1. Csató, L., Opper, M.: Sparse on-line Gaussian processes. Neural Computation 14, 641–668 (2002)

2. A. J. Smola and P. Bartlett. Sparse greedy Gaussian process regression. In Advances in Neural Information Processing Systems 13. MIT Press, 2000.

3.L. Csato.´ Gaussian Processes — Iterative Sparse Approximations. PhD thesis, Aston University, UK, 2002

Appendix:

Because of the problem of memory size in the laptop, so we can just training the part of dataset to training the model. If we have the better hardware we can training the whole data set then the accuracy will be more steady and higher.