GP vs DL

Sehaj Grover
Nitin Kulkarni
Xiao Zhang

Abstract

In this project we employ both Gaussian Process and Deep Learning to solve a real world time series multi-class classification problem and we tried different settings and structures of both GP and DL models. Then we further evaluate the results of various models based on accuracy and time cost. In the end, we summarize the general cases where Gaussian process and Deep Learning should apply and the best strategy to use them as well.

1. Introduction

Our aim is to compare the performance of Deep Learning and Gaussian Process classification. To this end, we employ both methods with various settings and structures to solve a real world time series multi-class classification problem.

We used The Photometric LSST Astronomical Time-Series Classification Challenge (PLAsTiCC) data set. It is a 37.37 GB data set which has the electromagnetic and optical observations of celestial bodies as features and categories of celestial bodies as labels. Therefore we are essentially helping the astronomical observatory to identify the new stars, new galaxies, etc based their observation data.

We tried various Deep Learning models with different layer numbers, layer width, regularization methods, drop out rates. On the other hand, we tried various Gaussian process with different Kernels:

- various combinations by adding the following kernels in different permutations.
- SquaredExponential
- Matern12
- Matern32
- Matern52
- Exponential
- Linear
- RationalQuadratic

and finally we evaluate the performances of these models based on their accuracy and time cost.

2. Data Set

The original data set is 37.37 GB, which consists of training set, training meta set and many batches of test sets. The training set is shown as below:

```
print('First 5 entries in the data set:\n', dataset.head())
```

```
First 5 entries in the data set:
    object_id         mjd  passband        flux   flux_err  detected
0        615  59750.4229         2 -544.810303   3.622952         1
1        615  59750.4306         1 -816.434326   5.553370         1
2        615  59750.4383         3 -471.385529   3.801213         1
3        615  59750.4450         4 -388.984985  11.395031         1
4        615  59752.4070         2 -681.858887   4.041204         1
```

Figure 1. The original data set

We have the following features:

- **object_id**: Different ID refers to different celestial bodies.
- **mjd**: The time in Modified Julian Date (MJD) of the observation.
- **passband**: The specific LSST passband integer.
- **flux**: The measured flux (brightness) in the passband of observation as listed in the passband column.
- **flux_err**: The uncertainty on the measurement of the flux listed above.
- **detected**: It is a Boolean number. If 1, the object's brightness is significantly different at the 3-sigma level relative to the reference template.

The passband, which may cause some confusion, refers to a set of various spectra properties including:

- 0: ultraviolet
- 1: green
- 2: red
- 3: infra red
- 4: spectro-scopic red shift
- 5: photometric red shift

The data of different passbands characterises the celestial objects, and since they are listed in chronological order, we essentially have at most 6 time series for each object. Let's use the object 615 as an example. We first extract all the rows which has ID 615 in the data set.

```
test1 = pd.DataFrame(dataset.where(dataset['object_id'] == 615))

test1 = test1.dropna()

print(test1)
```

	object_id	mjd	passband	flux	flux_err	detected
0	615.0	59750.4229	2.0	-544.810303	3.622952	1.0
1	615.0	59750.4306	1.0	-816.434326	5.553370	1.0
2	615.0	59750.4383	3.0	-471.385529	3.801213	1.0
3	615.0	59750.4450	4.0	-388.984985	11.395031	1.0
4	615.0	59752.4070	2.0	-681.858887	4.041204	1.0
..	...	...	...	...	...	...
347	615.0	60624.1760	2.0	552.150269	3.917989	1.0
348	615.0	60624.1836	1.0	607.047668	5.140991	1.0
349	615.0	60624.1913	3.0	296.946533	3.475000	1.0
350	615.0	60624.2022	4.0	235.489929	3.926538	1.0
351	615.0	60624.2132	5.0	157.080200	8.453112	1.0

[352 rows x 6 columns]

Figure 2. Object 615

Obviously object 615 has data from passband 0 to 5, and if we extract all the rows which have passband integer equals 1 then we have a time series.

```
In [24]: test2 = pd.DataFrame(test1.where(dataset['passband'] == 1))

In [25]: test2 = test2.dropna()

In [26]: print(test2)
```

	object_id	mjd	passband	flux	flux_err	detected
1	615.0	59750.4306	1.0	-816.434326	5.553370	1.0
5	615.0	59752.4147	1.0	-1061.457031	6.472994	1.0
10	615.0	59767.3045	1.0	-815.188599	5.293019	1.0
15	615.0	59770.2256	1.0	-820.042786	5.875329	1.0
20	615.0	59779.3265	1.0	-921.002502	6.306800	1.0
25	615.0	59782.1974	1.0	-449.095612	4.028310	1.0
30	615.0	59797.2938	1.0	35.511822	3.163646	1.0
35	615.0	59800.3244	1.0	129.541901	4.358776	1.0
40	615.0	59807.1815	1.0	-420.796417	4.037735	1.0
45	615.0	59810.1122	1.0	-527.020325	4.204173	1.0
50	615.0	59813.1122	1.0	-1100.440063	6.709106	1.0
60	615.0	59835.0678	1.0	-178.149399	3.809858	1.0
65	615.0	59839.0384	1.0	-953.883728	6.035410	1.0
70	615.0	59842.0285	1.0	-1003.971497	6.256784	1.0
76	615.0	59854.0873	1.0	217.894211	3.332742	1.0
81	615.0	59857.0531	1.0	646.523193	5.291624	1.0
86	615.0	59864.0239	1.0	-942.167908	5.916004	1.0
91	615.0	59867.0255	1.0	-910.677734	5.852032	1.0
96	615.0	59870.0272	1.0	659.486694	4.991051	1.0

Figure 3. Extract passband 1

And the time series of flux values from passband 1 looks like the following:

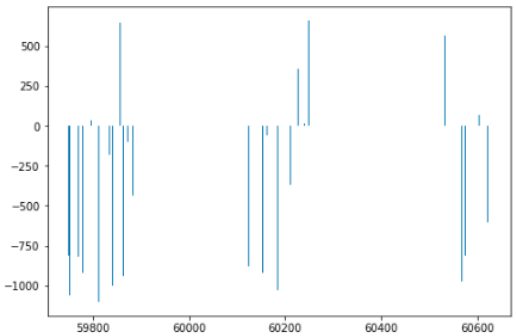

Figure 4. Time series (passband 1)

There are 5 other time series which are of their corresponding passbands as well. These time series contribute most parts of the information in this data set.

We also have another data set used in training which is training meta data set. It provides more information about the celestial bodies but we only care about the **target** column. It refers to different astronomical classes from red dwarf to galaxy.

The labels included in meta data set are[1]:

- **object_id**: Unique object identifier.
- **ra**: Right ascension, sky coordinate.
- **decl**: Declination, sky coordinate.
- **gal_l**: Galactic longitude in degrees.
- **gal_b**:Galactic latitude in degrees.
- **ddf**: A flag to identify the object as coming from the DDF survey area. Boolean
- **hostgal_specz**: Spectroscopic redshift of the source.
- **hostgal_photoz**: The photometric redshift of the host galaxy of the astronomical source.
- **hostgal_photoz_err**: The uncertainty on the host-gal_photoz based on LSST survey projections.
- **distmod**: The distance to the source calculated from hostgal_photoz and using general relativity.
- **mwebv**: MW E(B-V). this 'extinction' of light is a property of the Milky Way (MW) dust along the line of sight to the astronomical source, and is thus a function of the sky coordinates of the source ra, decl.
- **target**: The class of the astronomical source.

```
# Description of the data set.
print('MetaData Dataset description:\n', dataset_metadata.describe())
```

MetaData Dataset description:

	object_id	ra	decl	gal_l	gal_b
count	7.848000e+03	7848.000000	7848.000000	7848.000000	7848.000000
mean	4.810375e+07	169.719379	-27.254846	230.466784	-20.115526
std	4.347364e+07	114.212761	20.248556	97.416392	40.390899
min	6.150000e+02	0.175781	-64.760857	0.107681	-89.615573
25%	3.180865e+05	62.402344	-44.597992	207.248509	-54.509752
50%	4.131719e+07	150.996094	-27.279613	244.127187	-30.217582
75%	8.665901e+07	282.875976	-8.838267	313.224277	16.913859
max	1.307798e+08	359.824219	4.181528	359.943827	65.931317

	ddf	hostgal_specz	hostgal_photoz	hostgal_photoz_err
count	7848.000000	7848.000000	7848.000000	7848.000000
mean	0.269623	0.254436	0.357885	0.155695
std	0.443792	0.330053	0.545552	0.300367
min	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	0.000000	0.182750	0.210300	0.018000
75%	1.000000	0.350125	0.431200	0.122375
max	1.000000	3.445100	2.999400	1.734800

	distmod	mwebv	target
count	5523.000000	7848.000000	7848.000000
mean	41.263961	0.081953	60.895387
std	2.262711	0.150598	28.767984
min	31.996100	0.003000	6.000000
25%	39.845250	0.018000	42.000000
50%	41.167900	0.032000	65.000000
75%	42.398550	0.076000	90.000000
max	47.025600	2.747000	95.000000

Figure 5. Meta data

The mapping between the class numbers and their corresponding descriptions is shown below:

model class num[a]: name	model description	contributor(s)[b]	Nevent Gen[c]	Nevent train[d]	Nevent test[e]	redshift range[f]
90: SNIa	WD detonation, Type Ia SN	RK	16,353,270	2,313	1,659,831	< 1.6
67: SNIa-91bg	Peculiar type Ia: 91bg	SG,LG	1,329,510	208	40,193	< 0.9
52: SNIax	Peculiar SNIax	SJ,MD	8,660,920	183	63,664	< 1.3
42: SNII	Core Collapse, Type II SN	SG,LG:RK,JRP:VAV	59,198,660	1,193	1,000,150	< 2.0
62: SNIbc	Core Collapse, Type Ibc SN	VAV:RK,JRP	22,599,840	484	175,094	< 1.3
95: SLSN-I	Super-Lum. SN (magnetar)	VAV	90,640	175	35,782	< 3.4
15: TDE	Tidal Disruption Event	VAV	58,550	495	13,555	< 2.6
64: KN	Kilonova (NS-NS merger)	DK,GN	43,150	100	131	< 0.3
88: AGN	Active Galactic Nuclei	SD	175,500	370	101,424	< 3.4
92: RRL	RR lyrae	SD	200,200	239	197,155	0
65: M-dwarf	M-dwarf stellar flare	SD	800,800	981	93,494	0
16: EB	Eclipsing Binary stars	AP	220,200	924	96,572	0
53: Mira	Pulsating variable stars	RH	1,490	30	1,453	0
6: μLens-Single	μ-lens from single lens	RD,AA:EB,GN	2,820	151	1,303	0
991: μLens-Binary	μ-lens from binary lens	RD,AA	1,010	0	533	0
992: ILOT	Intermed. Lum. Optical Trans.	VAV	4,521,970	0	1,702	< 0.4
993: CaRT	Calcium Rich Transient	VAV	2,834,500	0	9,680	< 0.9
994: PISN	Pair Instability SN	VAV	5,650	0	1,172	< 1.9
995: μLens-String	μ-lens from cosmic strings	DC	30,020	0	0	0
TOTAL	Sum of all models		117,128,700	7,846	3,492,888	—

Figure 6. Description

Therefore we just extract the **target** column and attach it to the training data set.

3. Challenges

1. **Data Size**: The original data 37.37 GB and it is split into 12 files. Therefore, it becomes a challenge to go through all the separate files to extract all the data we want.

2. **Imbalanced Data**: The data is imbalanced, namely, the classes have different numbers of objects. We cannot use all objects from every class because it is inevitable that the classification on some classes performs better simply because those classes have more data.

3. **Conflict between GP and DL**: DL requires fairly large data set to perform well. Meanwhile, GP cannot handle big data set due to the inherent high time complexity. Thus, we have to figure out a balance point or even employ feature engineering to compare the performance of GP and DL on the same data set.

4. **Time requirement**: No matter how trivial the experiment on this data is, it always takes several hours to finish. For example it took us 6 to 7 hours to run the GP vs DL competition. However, this is the only way for us to know their real performances on big data set.

4. Feature Engineering

According to all the difficulties mentioned in the last section, we have to do some feature engineerings. It is essential because of the nature of the data set. This data set is huge, which has 37.37 GB size. However most parts of this big data set consist of time series data of different passbands. For example as Figure 2 shows the object 615 has 351 rows of time series data. Hence here comes the problem, if we reduce the data set by removing objects to a size that GP can handle, there will only be dozens of objects per class left for DL, which is impossible for DL to even start training. If we reduce the data set size by selecting fewer classes, we can barely perform a 3 classes classification, which ends up a toy model contributing nothing to this astronomic project. Hence we come up with the idea to use a set of critical values to represent the information of the time series as much as possible. Hence we can reduce, let's say the 351 rows data of object 651, to 1 row.

The features we pick to represent the time series including:

- **flux_min_passband**:
  The min flux value
- **flux_max_passband**:
  The max flux value
- **flux_median_passband**:
  The median flux value
- **flux_std_passband**:
  STD of flux value
- **flux_skew_passband**:
  Skewness of flux value
- **flux_diff_passband**:
  Range of flux value
- **flux_err_min_passband**:
  The min flux error value
- **flux_err_max_passband**:
  The max flux error value
- **flux_err_median_passband**:
  The median flux error value
- **flux_err_std_passband**:
  STD of flux error value
- **flux_err_skew_passband**:
  Skewness of flux error value
- **flux_err_diff_passband**:
  Range of flux error value
- **flux_per_err_sq_sum_passband**:
  Sum of squared (flue_error/mean)
- **flux_per_err_sq_skew_passband**:
  Skewness/mean
- **flux_by_flux_per_err_sq_sum_passband**:
  Sum of squared (flue/mean)
- **flux_by_flux_per_err_sq_skew_passband**:
  Sum of squared (flue/mean)

The flux and flux error data of each passband have very different statistical features. Hence we should calculate the above data for all 6 different passbands for every object.

The following picture shows what information contained in the first row (object 615) of our created new data set. Note that we only output the first 30 features.

```
In [12]: batch2.iloc[0, :30]

Out[12]: object_id                                615.000000
         number_of_timesteps                      352.000000
         detected_mean                              0.946023
         mjd_diff                                 873.790300
         flux_min_passband_0                     -116.913223
         flux_max_passband_0                      125.182808
         flux_median_passband_0                   -10.015225
         flux_std_passband_0                       83.944735
         flux_skew_passband_0                       0.128917
         flux_diff_passband_0                     242.096031
         flux_err_min_passband_0                    2.844200
         flux_err_max_passband_0                    4.737393
         flux_err_median_passband_0                 3.866380
         flux_err_std_passband_0                    0.493621
         flux_err_skew_passband_0                  -0.038907
         flux_err_diff_passband_0                   1.893193
         flux_per_err_sq_sum_passband_0         32478.068810
         flux_per_err_sq_skew_passband_0            0.438299
         flux_by_flux_per_err_sq_sum_passband_0  -626381.909593
         flux_by_flux_per_err_sq_skew_passband_0      -0.213985
         flux_min_passband_1                    -1100.440063
         flux_max_passband_1                      660.626343
         flux_median_passband_1                  -488.057968
         flux_std_passband_1                      601.787302
         flux_skew_passband_1                       0.415580
         flux_diff_passband_1                    1761.066406
         flux_err_min_passband_1                    3.034533
         flux_err_max_passband_1                    6.952323
         flux_err_median_passband_1                 4.980931
         flux_err_std_passband_1                    1.167671
         Name: 0, dtype: float64
```

Figure 7. New Data

With this new data set we can solve the conflict between GP and DL fairly well. The experiment showed that the DL and GP all recorded nearly 100% accuracy in 3 classes classification task with this new data set. It seems that the above critical values can replace the time series data well in practice.

However, as the second point in the Challenges section mentioned, the data set (also the new data set) is imbalanced. We have to extract the same amount of objects for each class. If we extract 2000 objects per class, then we will find some classes doesn't even have that many objects. To solve this problem we further created 3 smaller but balanced data sets based on our reformed new data set.

- 13 classes and 500 objects per class.
- 12 classes and 1000 objects per class.
- 7 classes and 2000 objects per class.

All the following evaluations based on these data sets.

5. Models

As for the Deep Learning models, we have tested 4 different models for Deep Learning.
DL model 1:

- Model type: Fully connected MLP
- Layers number: 2
- Layer width: 512, 218
- Regularization: L2
- Optimizer: Adam

```
Model summary:
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               54272
_________________________________________________________________
dense_1 (Dense)              (None, 128)               65664
_________________________________________________________________
dense_2 (Dense)              (None, 7)                 903
_________________________________________________________________
activation (Activation)      (None, 7)                 0
=================================================================
Total params: 120,839
Trainable params: 120,839
Non-trainable params: 0
```

Figure 8. DL1

DL model 2:

- Model type: Fully connected MLP
- Layers number: 2
- Layer width: 256, 64
- Regularization: None
- Optimizer: Adam

```
Model summary:
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 256)               27136
_________________________________________________________________
dense_4 (Dense)              (None, 64)                16448
_________________________________________________________________
dense_5 (Dense)              (None, 7)                 455
_________________________________________________________________
activation_1 (Activation)    (None, 7)                 0
=================================================================
Total params: 44,039
Trainable params: 44,039
Non-trainable params: 0
```

Figure 9. DL2

DL model 3:

- Model type: Fully connected MLP
- Layers number: 4
- Layer width: 512, 256, 128, 64
- Regularization: L2
- Optimizer: Adam

```
Model summary:
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 512)               54272
_________________________________________________________________
dense_7 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_8 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_9 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_10 (Dense)             (None, 7)                 455
_________________________________________________________________
activation_2 (Activation)    (None, 7)                 0
=================================================================
Total params: 227,207
Trainable params: 227,207
Non-trainable params: 0
```

Figure 10. DL3

DL model 4:

- Model type: Fully connected MLP
- Layers number: 4
- Layer width: 512, 256, 128, 64
- Regularization: L1, L2
- Optimizer: Adam

```
Model summary:
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_11 (Dense)             (None, 512)               54272
_________________________________________________________________
dense_12 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_13 (Dense)             (None, 128)               32896
_________________________________________________________________
dense_14 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_15 (Dense)             (None, 7)                 455
_________________________________________________________________
activation_3 (Activation)    (None, 7)                 0
=================================================================
Total params: 227,207
Trainable params: 227,207
Non-trainable params: 0
```

Figure 11. DL4

As for GP we tried the following kernels or combination of kernels on the data set.

- SquaredExponential
- Matern12

- Matern32
- Matern52
- Exponential
- Linear
- RationalQuadratic
- SquaredExponential + Matern12
- SquaredExponential + Matern32
- SquaredExponential + Matern52
- SquaredExponential + Matern12 + Exponential
- SquaredExponential + Matern32 + Exponential
- SquaredExponential + Matern52 + Exponential
- RationalQuadratic + Matern12
- RationalQuadratic + Matern32
- RationalQuadratic + Matern52
- SquaredExponential + White
- Matern12 + White
- Matern32 + White
- Matern52 + White
- Exponential + White
- Linear + White
- RationalQuadratic + White
- SquaredExponential + Matern12 + White
- SquaredExponential + Matern32 + White
- SquaredExponential + Matern52 + White
- SquaredExponential + Matern12 + Exponential + White
- SquaredExponential + Matern32 + Exponential + White
- SquaredExponential + Matern52 + Exponential + White
- RationalQuadratic + Matern12 + White
- RationalQuadratic + Matern32 + White
- RationalQuadratic + Matern52 + White

6. Results

After applying these models on the 2000 objects per class data set for 3 times we found that for the deep learning model, the Deep Learning **model 2** performed better than the other 3 models stably.

For **model 1** we record:

```
Evaluation dl Model results:
Precision:  0.78
Recall:  0.79
F1-Score:  0.78
Accuracy:  0.79

Per Class Results:
              precision    recall  f1-score   support

           0       0.75      0.88      0.81       392
           1       0.99      0.97      0.98       366
           2       0.85      0.93      0.89       406
           3       0.55      0.33      0.42       410
           4       0.63      0.79      0.70       411
           5       0.98      1.00      0.99       420
           6       0.71      0.62      0.66       395

    accuracy                           0.79      2800
   macro avg       0.78      0.79      0.78      2800
weighted avg       0.78      0.79      0.78      2800
```

Figure 12. DL1

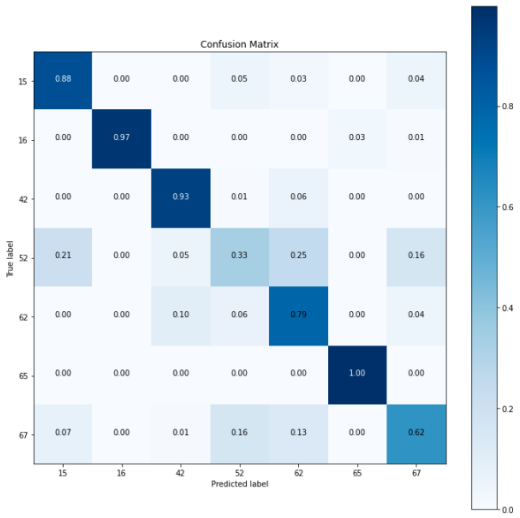

Figure 13. DL1

For **model 2** we record:

```
Evaluation dl Model results:
Precision:  0.82
Recall:  0.81
F1-Score:  0.81
Accuracy:  0.81

Per Class Results:
              precision    recall  f1-score   support

           0       0.88      0.89      0.89       406
           1       0.99      1.00      0.99       382
           2       0.84      0.93      0.88       359
           3       0.58      0.52      0.55       433
           4       0.63      0.83      0.72       392
           5       0.99      0.99      0.99       429
           6       0.80      0.55      0.65       399

    accuracy                           0.81      2800
   macro avg       0.82      0.82      0.81      2800
weighted avg       0.82      0.81      0.81      2800
```

Figure 14. DL2

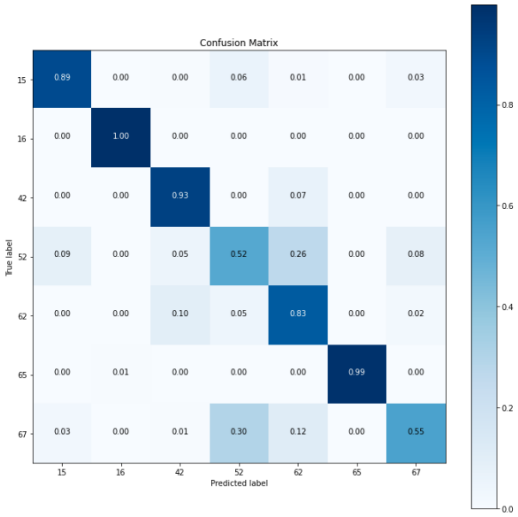

Figure 15. DL2

For **model 3** we record:

```
Evaluation dl Model results:
Precision:  0.78
Recall:  0.78
F1-Score:  0.77
Accuracy:  0.78

Per Class Results:
              precision    recall  f1-score   support

           0       0.84      0.81      0.83       386
           1       0.99      0.97      0.98       414
           2       0.78      0.97      0.87       402
           3       0.55      0.36      0.43       419
           4       0.55      0.78      0.65       377
           5       0.97      0.98      0.97       414
           6       0.74      0.58      0.65       388

    accuracy                           0.78      2800
   macro avg       0.78      0.78      0.77      2800
weighted avg       0.78      0.78      0.77      2800
```

Figure 16. DL3

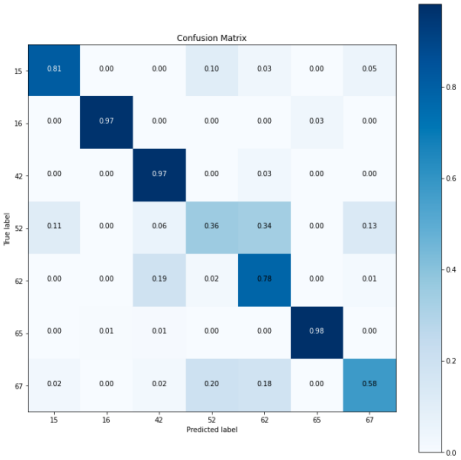

Figure 17. DL3

For **model 4** we record:

```
Evaluation dl Model results:
Precision:  0.02
Recall:  0.15
F1-Score:  0.04
Accuracy:  0.15

Per Class Results:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       389
           1       0.00      0.00      0.00       419
           2       0.00      0.00      0.00       404
           3       0.00      0.00      0.00       390
           4       0.15      1.00      0.26       415
           5       0.00      0.00      0.00       391
           6       0.00      0.00      0.00       392

    accuracy                           0.15      2800
   macro avg       0.02      0.14      0.04      2800
weighted avg       0.02      0.15      0.04      2800
```

Figure 18. DL4

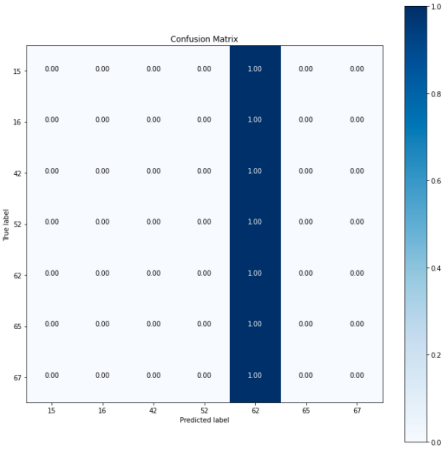

Figure 19. DL4

Hence we can clearly see that Deep Learning **model 2**, which has the following structure, performs the best.

- Model type: Fully connected MLP
- Layers number: 2
- Layer width: 256, 64
- Regularization: None
- Optimizer: Adam

As for the GP models, we also applied all the models with different kernels to the 2000 objects per class data set 3 times and recorded the following results(in the same order with the list in model section). In fact most of the models perform well and the differences only fall into a very small range. We pick the best three models based on their overall performance and stability.

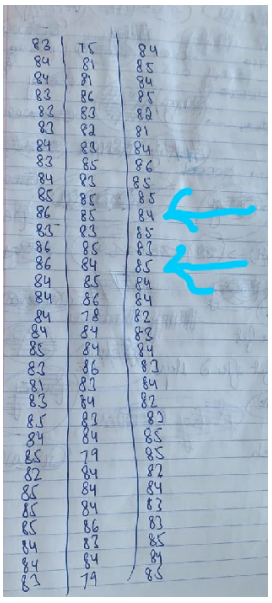

Figure 20. GP results

The following models performed the top:

- SquaredExponential + Matern52
  (average accuracy observed: 0.85)

```
Optimization took 210.73 seconds

Evaluation GP Model results:
Precision:  0.85
Recall:  0.85
F1-Score:  0.85
Accuracy:  0.85

Per Class Results:
              precision    recall  f1-score   support

           0       0.92      0.87      0.89       104
           1       0.91      0.85      0.88       101
           2       0.96      0.93      0.94        91
           3       0.78      0.74      0.76        94
           4       0.77      0.53      0.63        97
           5       0.95      0.98      0.96        90
           6       0.59      0.75      0.66       101
           7       0.89      0.95      0.92        96
           8       0.80      0.80      0.80        93
           9       0.94      0.90      0.92       114
          10       0.79      0.86      0.83       110
          11       0.96      0.97      0.97       118
          12       0.83      0.88      0.86        91

    accuracy                           0.85      1300
   macro avg       0.85      0.85      0.85      1300
weighted avg       0.85      0.85      0.85      1300
```

Figure 21. SquaredExponential + Matern52

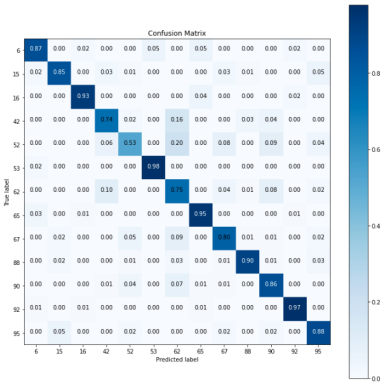

Figure 22. SquaredExponential + Matern52

- SquaredExponential + Matern12 + Exponential
  (average accuracy observed: 0.86)

```
Optimization took 232.53 seconds

Evaluation GP Model results:
Precision:  0.85
Recall:  0.84
F1-Score:  0.84
Accuracy:  0.84

Per Class Results:
              precision    recall  f1-score   support

           0       0.90      0.94      0.92        98
           1       0.83      0.84      0.84        88
           2       0.98      0.97      0.98       103
           3       0.82      0.65      0.73        95
           4       0.81      0.48      0.60        96
           5       0.96      0.97      0.96        99
           6       0.54      0.80      0.65        96
           7       0.94      0.92      0.93       119
           8       0.76      0.64      0.70       101
           9       0.91      0.91      0.91       108
          10       0.76      0.87      0.81       105
          11       0.99      1.00      0.99        93
          12       0.81      0.88      0.84        99

    accuracy                           0.84      1300
   macro avg       0.85      0.84      0.83      1300
weighted avg       0.85      0.84      0.84      1300
```

Figure 23. SquaredExponential + Matern12 + Exponential

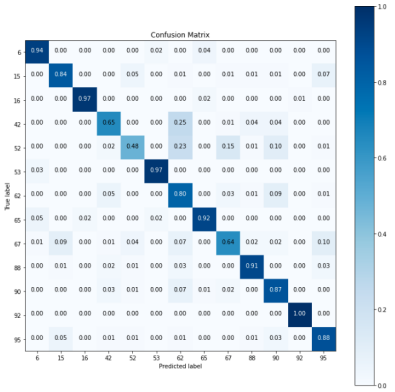

Figure 24. SquaredExponential + Matern12 + Exponential

- SquaredExponential + Matern52 + Exponential
  (average accuracy observed: 0.84)

```
Optimization took 236.91 seconds

Evaluation GP Model results:
Precision:  0.83
Recall:  0.83
F1-Score:  0.83
Accuracy:  0.83

Per Class Results:
              precision    recall  f1-score   support

           0       0.86      0.83      0.85        83
           1       0.81      0.86      0.84        86
           2       0.95      0.97      0.96        95
           3       0.81      0.72      0.76        93
           4       0.72      0.50      0.59        96
           5       0.96      0.96      0.96       109
           6       0.53      0.73      0.61       106
           7       0.90      0.93      0.92       117
           8       0.79      0.71      0.75       106
           9       0.89      0.86      0.87        99
          10       0.78      0.85      0.81       118
          11       0.99      0.97      0.98       100
          12       0.86      0.84      0.85        92

    accuracy                           0.83      1300
   macro avg       0.83      0.82      0.83      1300
weighted avg       0.83      0.83      0.83      1300
```

Figure 25. SquaredExponential + Matern52 + Exponential

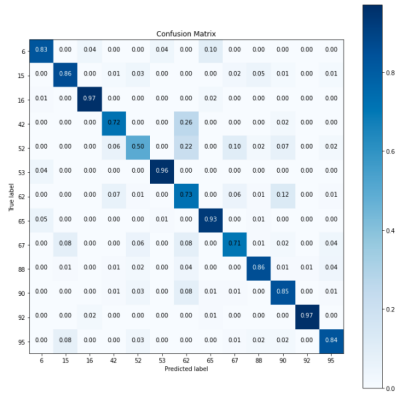

Figure 26. SquaredExponential + Matern52 + Exponential

- RationalQuadratic + Matern52
  (average accuracy observed: 0.84)

```
Optimization took 222.79 seconds

Evaluation GP Model results:
Precision:  0.85
Recall:  0.85
F1-Score:  0.85
Accuracy:  0.85

Per Class Results:
              precision    recall  f1-score   support

           0       0.91      0.90      0.91       104
           1       0.88      0.82      0.85       102
           2       0.96      0.99      0.98       109
           3       0.82      0.74      0.78        86
           4       0.79      0.58      0.67        90
           5       0.95      0.97      0.96       105
           6       0.62      0.75      0.67       110
           7       0.95      0.94      0.94        96
           8       0.75      0.73      0.74        86
           9       0.85      0.89      0.87        95
          10       0.72      0.88      0.79       101
          11       0.97      0.97      0.97       107
          12       0.91      0.80      0.85       109

    accuracy                           0.85      1300
   macro avg       0.85      0.84      0.85      1300
weighted avg       0.85      0.85      0.85      1300
```

Figure 27. RationalQuadratic + Matern52

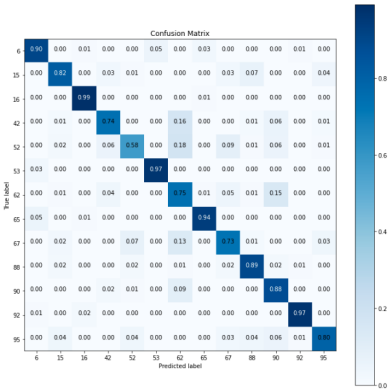

Figure 28. RationalQuadratic + Matern52

- Matern32 + White(variance=0.01)
  (average accuracy observed: 0.84)

```
Optimization took 188.43 seconds

Evaluation GP Model results:
Precision:  0.84
Recall:  0.84
F1-Score:  0.84
Accuracy:  0.84

Per Class Results:
              precision    recall  f1-score   support

           0       0.89      0.91      0.90       102
           1       0.84      0.86      0.85        92
           2       0.97      0.99      0.98       102
           3       0.82      0.74      0.78       101
           4       0.65      0.44      0.53        88
           5       0.97      0.96      0.97       110
           6       0.58      0.72      0.64       106
           7       0.95      0.93      0.94        92
           8       0.76      0.66      0.70       103
           9       0.92      0.89      0.90        85
          10       0.72      0.89      0.80       102
          11       0.99      0.98      0.99       109
          12       0.85      0.86      0.85       108

    accuracy                           0.84      1300
   macro avg       0.84      0.83      0.83      1300
weighted avg       0.84      0.84      0.84      1300
```

Figure 29. Matern32 + White

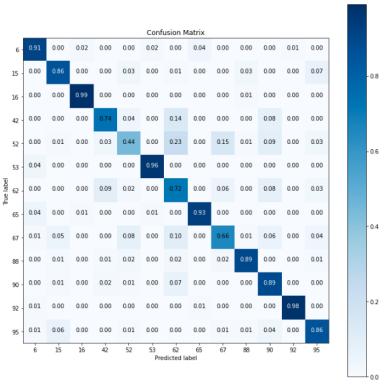

Figure 30. Matern32 + White

We finally picked kernel **SquaredExponential + Matern12 + Exponential** and applied it to all the 500 objects per class, 1000 objects per class, and 2000 objects per class data set to compete with Deep Learning **model 2**.

We recorded the following results about the competition: **For 500 objects per class data set**, we record the F1 scores of selected Deep Learning model and Gaussian process models:

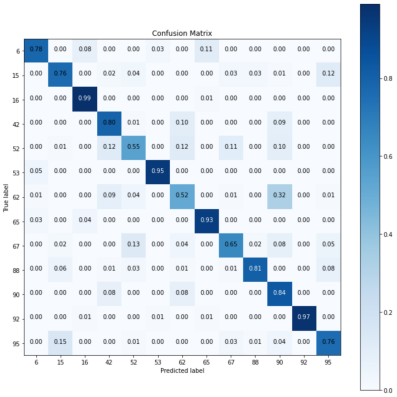

Figure 31. DL 500 Accuracy

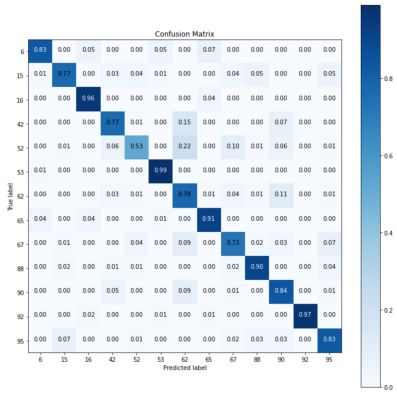

Figure 32. GP 500 Accuracy

Their corresponding (validation) accuracy, F1-score and time taken training history are:

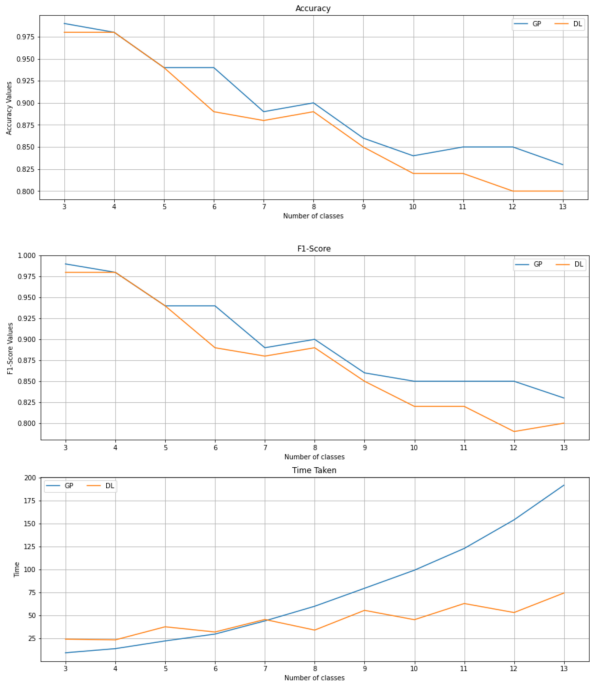

Figure 33. history 500

For 1000 objects per class data set, we record the F1 scores of selected Deep Learning model and Gaussian process model:

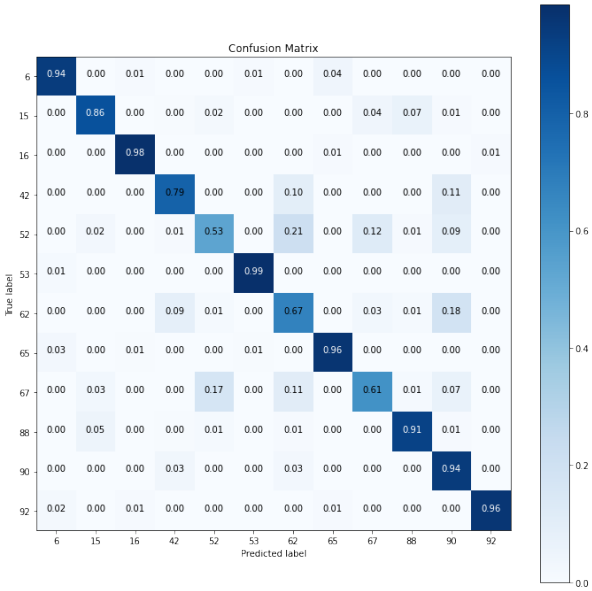

Figure 34. DL 1000 Accuracy

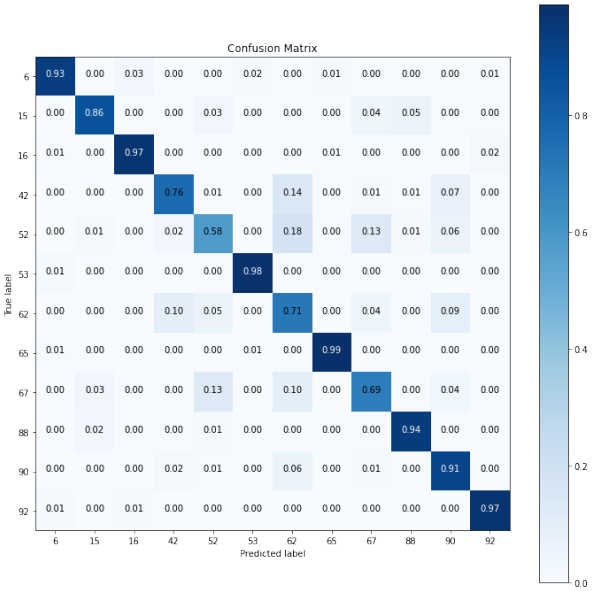

Figure 35. GP 1000 Accuracy

Their corresponding (validation) accuracy, F1-score and time taken training history are:

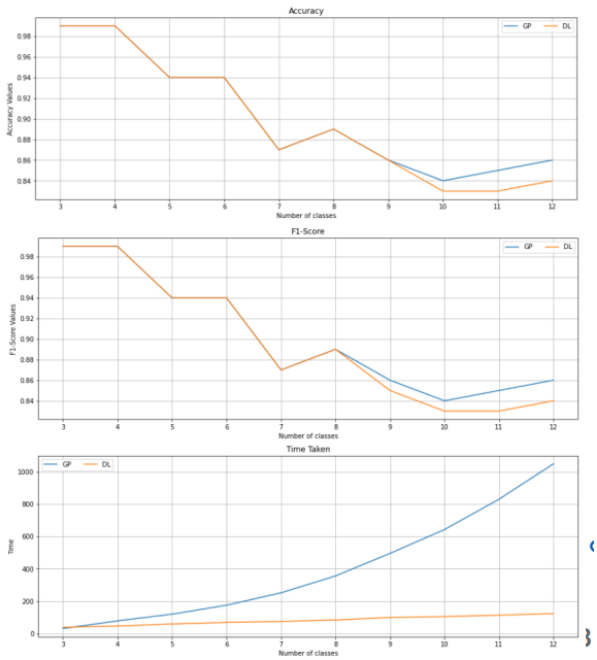

Figure 36. history 1000

For 2000 objects per class data set, we record the Accuracy scores of selected Deep Learning model and Gaussian process model:

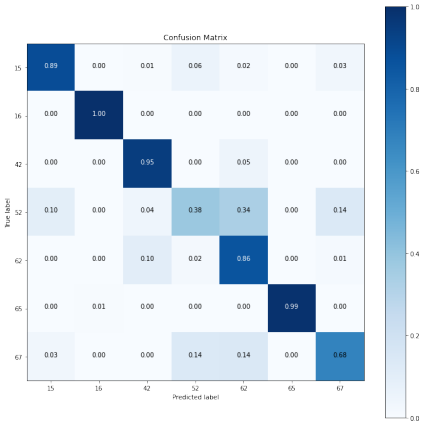

Figure 37. DL 2000 Accuracy

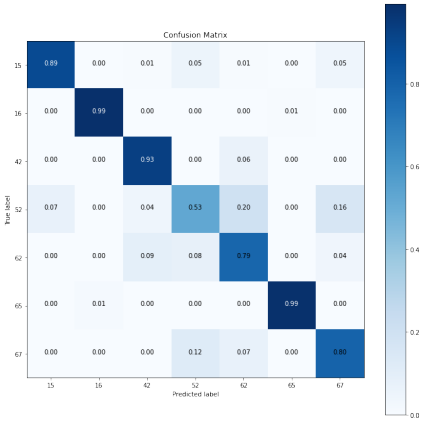

Figure 38. GP 2000 Accuracy

Their corresponding (validation) accuracy, F1-score and time taken training history are:

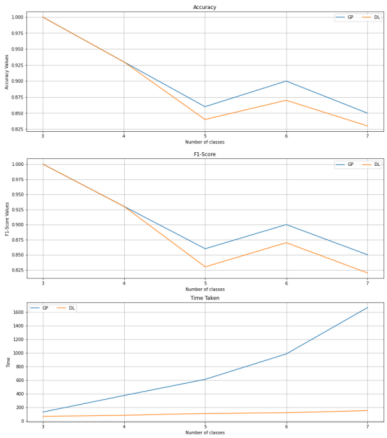

Figure 39. history 2000

7. Comparison and Reasoning

First we found that the results are not always the same so the following summary is applied to general cases. There may exist outliers in some attempts.

As we can see in competition results of Deep learning and Gaussian process, when the data set is simple, like we have only 3 categories in 2000, 1000, 500 objects per class data sets (see the starting point of figure 22, 25, 28), both methods perform perfectly. The accuracy is almost 100 percents. However when we increase the size of data set by including more categories, the accuracy decrease for both models. (The three accuracy graphs 22, 25, 28 all shows a decreasing trend)

As we can see in the three graphs, the accuracy of GP and DL both decrease when there are more categories involved. However the GP persists the trend and performs better than DL and the performance gap between GP and DL even also increases. Hence we can conclude that GP is better to handle more complex classification problem given a small data set. It is reasonable because given a small data set, the more complex the classification problem is, the fewer data points per class that DL can get, and it is known that DL performs poorly without enough data.

There is another one import thing we should notice in the 3 pair of comparisons. We can notice time used by Deep Learning really increases roughly linearly. However the time used by GP increases roughly exponentially.

Further, if we compare the performances of data set with 500 objects per class and 1000 objects per class, we can find that as the number of objects per class increases, the gap between GP and DL decrease.

So here is summary. From the above three pairs of comparisons, we can conclude that:

1. For this data set, The GP process performs better when more categories involved in the classification.

2. When the object number per class increase, although GP still perform better then DL, the performance gap actually decrease. This may be due to DL generally perform better with more data.

3. We can also notice that the time taken by GP increase roughly exponentially. And for DL it increase roughly linearly.

Therefore, the best strategy is. We can use GP when the data set is small because it performs better and it also doesn't take too much time. But when the size of data set increase, or more categories are involved, DL becomes a more and more attractive option.

References

[1] https://www.kaggle.com/c/PLAsTiCC-2018/data. 2