Finding optimal Kernel and Hyper-parameters for regression problem

Amit Upadhyay
University at Buffalo
Buffalo,New York
amitupad@buffalo.edu

Sai Bharat Kumar Konkalla
University at Buffalo
Buffalo,New York
skonakal@buffalo.edu

Abstract

Non-parametric approach of Machine learning seems to be very promising. Its performance in terms of accuracy on small data-sets shows it can easily replace parametric or deep learning models in these scenarios. Non-parametric approaches, unlike their parametric counterparts, don't need parameters. Instead, they depend on the training data. In fact, training data can informally be called as their parameters. Thus, any increase or decrease in data set can bring big changes in model's performance. In order to check the interaction between the results of function applied on these data-sets, they use co variance or kernel function. The "training-phase" is used to find the optimal values of hyper-parameters used in these kernels. However, the big question is how to find these kernels? How to decide which kernel will suit for any real world data-set? There can be many solutions to a given problem.In same way,many kernels or combination of kernels can be used for a given problem. So what are the deciding factors to choose one from the lot? Further, comparative study is one of the best form of study.Can we compare performance of a non-parametric model with everyone's favorite, deep learning model? How can we explain the results of the proposed model? How can we back our model and explain what all it is considering before making any prediction or classification? Further, can we do something that will reduce the time taken to optimize our model? This project tries to answer all these questions. Apart from finding a really good kernel for a real world data-set, we will try to answer all the above raised questions.Also," developing kernel is like an art. You need time,practise and patience to develop one which best fits the given data-set". Our kernel may not be the best one. There may be other kernels that can fit even better on the used data-set and give far better performance,however, this is the start of a research where the aim is to not only understand how a kernel can be built, but also many basic concepts ,which are questioned above.

1. Introduction

What is a kernel function? This is a basic question, however its interpretation is often wrong. Kernels are often thought to be specifying the similarity between two objects. However this definition is bit misleading. In our opinion, we don't need specialised functions to find similarity between two objects. It can be found by simple methods like dot products or distance between them etc.*Kernel functions or co variance functions or co variance kernels (a term sometimes used in place of kernels) can be defined as a function that specifies how similar are the values obtained after a function is applied on two different objects.*[1] *This in turn specifies which functions are likely under a prior*. Please note that a kernel is a function with two arguments that maps a pair of inputs, $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ into $\mathbb{R}$. A kernel can be any function. However, for a kernel to be a valid co variance function, it needs to produce a positive semi-definite (PSD) Gram matrix. However, in this report, we are using the terms kernel and co variance functions interchangeably. The definition and importance of kernels can be best understood from the perspective of Basis-function expansion which we feel is the reason why Support Vector Machine became such powerful non-linear machine learning classifier. If there seems to be no linear boundary to classify the given data-set, basic function expansion takes the data-set to higher dimension where there are sufficient dimensions to convert the non-linear equation of the boundary into a linear form. However the issue is how the data-set are spread in that high dimension? Which points are nearer and which ones are far? Dot product seems to be the first choice to solve this problem. However, the number of operations in dot product is directly proportional to the number of dimensions. So what if the the data-points are transferred to a nearly infinite dimension? The functions used to transfer these points are called basis functions. The kernels or co variance functions are specialzed functions that can estimate the similarity of the outcomes of these basis functions

[1] https://www.cs.toronto.edu/ duvenaud/thesis.pdf

when applied on data-points and save us from extremely lengthy calculations.However, not every function can be defined as a kernel function. In fact they need to satisfy some properties before becoming one. These properties are 1. the function must produce symmetric and positive semi definite gram matrix.The formula used to define kernels is :

$$Cov[f(x), f(x^{"})] = k(x, x^{'})$$

2. Motivation

The report starts with a small description about different types of kernels available. Then we will discuss about the the different ways by which kernels can be brought together to form complex kernels. The advantages and disadvantages of these methods will be discussed. Then will be talk about the data-set which is used in this project.We will talk about the features of independent variables and what the dependent variable is predicting. Then, we will start building a personalized kernel for the data. We will show how this kernel is made . Upon Completion, we will compare its performance with other kernels, which we found can be used for this data-set. We will explain the scenarios where user can use our developed kernels and where he can go with other kernels, because, at the end, its the requirement of user which matters.Then we will show the importance of a proper initialization of hyper-parameters used in kernel as this can be a deciding factor to discriminate a good model from non-so-good ones.We will then move to the concept of interpretability.This is an important concept and serves as a link between developer and end user. Its the interpretability which is used by developer to explain the user about which feature or set of features, the developed model is giving importance to while delivering the results. This concept is missing in deep learning and that's one of the reasons, deep learning seems tough to understand. The absence of interpretability makes deep learning models like black-box. After the concept of interpretability, we try to compare the performance of our model with that of a deep learning model and thus set the norm regarding where deep learning can be used and where we can use non-parametric models. Finally, we will explore the concept of spare GP and try to further improve our model by tackling the main issue of any non- parametric model, i.e, the time taken to optimize the model (by finding the best values for hyper-parameters).

2.1. Language

All manuscripts are in English. The required graphs and pictures are used when found needed. We have tried to give the formulas and definitions for every-concept used (when found necessary).

3. Types of Kernels

There are various kinds of kernels. In this section, we will be briefly discussing about a few of them..[2] Please check figure 1 for few of the commonly known kernels. The most commonly used kernel is RBF which assumes that the model is infinitely derivable. Its a very smooth function.The strong smoothness property, however, is often unrealistic for representing physical behavior. Due to this unrealistic behaviour of RBF, Matern family kernels are preferred. Note that there are many functions in matern family like matern12, matern32 ,matern52 etc, however matern32 is most widely used. Matern kernels is a generalization of RBF. It has an additional parameter v which controls the smoothing of the resulting function.[3] The rational quadratic kernel function can be thought of as an infinite sum of SE functions. Linear kernels make the model grow linearly. Periodic kernels are used when there is some kind of periodicity(like temporal or spatial) in data. White noise kernels are used to add noise to model. Its mostly used as a sum function. We even have a universal kernel called SE-ARD kernel which is a product of infinite kernels. They are capable of learning any continues function provided enough data is given and some other conditions are met. They are in fact default kernels for many models.Apart from these we have many other kernels like constant kernel, neural network kernels etc. Each of these kernels corresponds to various types of assumption we make about the function that we wish to model. Kernels can be broadly classified into two categories, stationary kernels(those whose value depends only on difference between two points) and non-stationary kernels(which do not depend on or only on the difference between the two data points).

covariance function	expression	S	ND
constant	σ_0^2	√	
linear	$\sum_{d=1}^{D} \sigma_d^2 x_d x'_d$		
polynomial	$(\mathbf{x} \cdot \mathbf{x}' + \sigma_0^2)^p$		
squared exponential	$\exp(-\frac{r^2}{2\ell^2})$	√	√
Matérn	$\frac{1}{2^{\nu-1}\Gamma(\nu)} \left(\frac{\sqrt{2\nu}}{\ell} r\right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}}{\ell} r\right)$	√	√
exponential	$\exp(-\frac{r}{\ell})$	√	√
γ-exponential	$\exp\left(-\left(\frac{r}{\ell}\right)^\gamma\right)$	√	√
rational quadratic	$\left(1 + \frac{r^2}{2\alpha\ell^2}\right)^{-\alpha}$	√	√
neural network	$\sin^{-1}\left(\frac{2\tilde{\mathbf{x}}^\top \Sigma \tilde{\mathbf{x}}'}{\sqrt{(1+2\tilde{\mathbf{x}}^\top \Sigma \tilde{\mathbf{x}})(1+2\tilde{\mathbf{x}}'^\top \Sigma \tilde{\mathbf{x}}')}}\right)$		√

Figure 1. various types of kernels

[2] https://github.com/ubdsgroup/mladvanced-notebooks/blob/master/CovarianceFunctions.ipynb

[3] https://www.cs.toronto.edu/ duvenaud/cookbook/

4. Combining Kernels

Combination of kernels can be performed in two ways, via addition or via multiplication. As stated above, universal kernel is an example of combination via multiplication. Combination via multiplication is specially useful when we want to catch even small interactions between various sub-kernels. In fact multiplication allows properties of one feature to interfere or affect with that of others. However this combination takes long time to optimize and many times can't make predictions far from training data. Thus its possible to say that this type of connectivity suffer from curse of dimensionality. On the other hand, combination via addition makes a more generalized model. Addition in fact allows us to make strong assumptions about features that make up the total sum. Addition doesn't allow properties of one feature to affect or interact with that of others and in contrast to multiplication, helps us to make predictions far from training set. In fact, if we have two functions on two different features such that fa belongs to GP(0,K1) and fb belongs to GP(0,K2), then fa+fb belongs to GP(0+0,K1+K2).

5. Data-Set

The data-set we use here is Concrete Compressive Strength Data Set. It got 7 quantitative components ie Cement(kg in a m3 mixture), Blast Furnace Slag (kg in a m3 mixture),Fly Ash (kg in a m3 mixture) ,Water (litres in a m3 mixture),Superplasticizer (kg in a m3 mixture),Coarse Aggregate (kg in a m3 mixture),Fine Aggregate (kg in a m3 mixture) and one qualitative component ie age (1-365 days).The value which needs to be predicted is Concrete compressive strength which describes what is the strength of our Concrete. As the title of project suggests, its a regression problem. The data-set has 1030 instances. Please note that until not stated, we, by default have taken 900 instances for training and remaining for testing.The names of each of these features are self explanatory and we have tried to give as much information as possible. However, any further information,if needed can be found at Kaggle(from where we downloaded this data).[4]

6. Creation of Model

Now we are aware of the data-set, the different kinds of basic kernels and ways by which we can combine kernels, in this section ,we will build a personalized kernel for the given data-set. Please note that we have two options, ie combination via addition or via multiplication.. We will develop kernels via both models, however, only the kernel developed via addition will be used in future. We will compare their performances and prove kernel obtained via addition is far better than that obtained from multiplication (for our data-set).

In order to develop the kernels. we started taking every feature independently and thus created 8 new data sets, in with we had the independent variable as feature and the depend variable as the output(ie Concrete compressive strength).The performance measuring criteria used here is RMSE. We were tempted to use time too, however, time didn't play a big role when working on a single feature. In fact in most of the cases, time taken to optimize the models consisting of only single feature was almost same.

Please check figure(2) to figure(9) which tells what all kernels or combination of kernels were used and how much was the RMSE value.

KERNEL APPLIED	RMSE VALUE ACHIEVED
GPy.kern.PeriodicExponential(1)	13.461925302664945
GPy.kern.PeriodicExponential(1)+GPy.kern.White(1)	13.856605518005395
GPy.kern.RBF(1)	13.983043515849 44
GPy.kern.Matern32(1)	13.858215020457132
GPy.kern.Exponential(1)	13.823317378247484

Figure 2. RMSE obtained when various kernels were applied on Age

Kernels applied	RMSE
GPy.kern.RBF(1)	9.000628238999132
GPy.kern.Matern32(1) + GPy.kern.White(1)	8.72459536734425
GPy.kern.Matern52(1)	9.024058917828649
GPy.kern.RatQuad(1)	9.01502892930406
GPy.kern.Exponential(1)	8.83373221866587
GPy.kern.Linear(1)	10.53589695798355
GPy.kern.Matern32(1)+GPy.kern.RBF(1)	9.546986614096664

Figure 3. RMSE obtained when various kernels were applied on Cement

Kernel applied	RMSE
GPy.kern.PeriodicExponential(1)	36.784137629859013
GPy.kern.RBF(1)+ GPy.kern.PeriodicExponential(1)+ GPy.kern.Matern32(1)	9.000628238999132
GPy.kern.RBF(1)	13.684127854374712
GPy.kern.Matern32(1)	13.659202096266661
GPy.kern.Matern32(1)+GPy.kern.RBF(1)	11.761169078526917
GPy.kern.Exponential(1)	12.626720208364523

Figure 4. RMSE obtained when various kernels were applied on Blast-Furnace

Based on RMSE values obtained, the kernel or combination of kernels which gave least RMSE value (in other words, which best fitted the feature) where taken and added. This in-turn created a customized kernel for the complete data-set. Please check figure(10) and figure(11) which show

[4] https://www.kaggle.com/elikplim/concrete-compressive-strength-data-set

Kernel applied	RMSE
GPY.KERN.LINEAR(1)	27.6073104570133
GPY.KERN.MATERN32(1)	12.50485847671929
GPY.KERN.RBF(1)	12.50239603736967
GPY.KERN.RBF(1) + GPY.KERN.MATERN32(1) + GPY.KERN.WHITE(1)	12.502295762999804
GPY.KERN.RBF(1)+GPY.KERN.MATERN32(1)	12.103275893457544
GPY.KERN.PERIODICEXPONENTIAL(1)	34.13661093973437

Figure 5. RMSE obtained when various kernels were applied on Fly-Ash

Kernel Applied	RMSE
GPy.kern.Linear(1)	13.66653106940722
GPy.kern.PeriodicExponential(1)	34.990339942355014
GPy.kern.Matern32(1)	12.43364848017863
GPy.kern.RBF(1)	13.034199832254432
GPy.kern.RBF(1)+ GPy.kern.Matern32(1)	12.054193100102555
GPy.kern.Exponential(1)	11.878978008767849
GPy.kern.Exponential(1)+ GPy.kern.RBF(1) + GPy.kern.Matern32(1)	12.095257133960029

Figure 6. RMSE obtained when various kernels were applied on Water

Kernel applied	RMSE
GPY.KERN.LINEAR(1)	17.732165235838085
GPY.KERN.PERIODICEXPONENTIAL(1)	33.53884281358673
GPY.KERN.RBF(1)	15.540531377681656
GPY.KERN.EXPONENTIAL(1)	15.75890518155692
GPY.KERN.MATERN32(1)	14.664666739501342
GPY.KERN.RBF(1)+GPY.KERN.MATERN32(1)	15.201651641286457
GPY.KERN.RATQUAD(1)*GPY.KERN.RBF(1)	15.20151181959136
GPY.KERN.RATQUAD(1)+ GPY.KERN.RBF(1)+ GPY.KERN.MATERN32(1)	15.201669551205047
GPY.KERN.RATQUAD(1)	15.323532

Figure 7. RMSE obtained when various kernels were applied on Superplasticizer

Kernel applied	RMSE
GPy.kern.Linear(1)	13.09417249253765
GPy.kern.Linear(1)+ GPy.kern.Linear(1)+ GPy.kern.Matern32(1)	13.103546078645838
GPy.kern.Matern32(1)	13.739969270595836
GPy.kern.RBF(1)	14.471621363933507
GPy.kern.Linear(1)	12.880922103574205

Figure 8. RMSE obtained when various kernels were applied on Coarse Aggregate

Kernel applied	RMSE
GPY.KERN.LINEAR(1)	13.09417249253765
GPY.KERN.RBF(1)	14.039653112895623
GPY.KERN.MATERN32(1)	13.526485866648104
GPY.KERN.LINEAR(1)+GPY.KERN.MATERN32(1)	13.029881952346738
GPY.KERN.LINEAR(1)+GPY.KERN.MATERN32(1)+GPY.KERN.RBF(1)	12.770320323843084
GPY.KERN.RATQUAD(1)	12.572698910453155
GPY.KERN.RATQUAD(1)+GPY.KERN.MATERN32(1)+GPY.KERN.RBF(1)	12.452278548296935
GPY.KERN.RATQUAD(1)+GPY.KERN.MATERN32(1)	12.45225085344628

Figure 9. RMSE obtained when various kernels were applied on Fine Aggregate

all individual kernels and their optimized hyper-parameters (for the whole data-set)

Individual Feature	Kernel
Cement	Matern32
Age	Periodic
Blast furnace Slag	Matern32 +RBF+Exponential
Fly Ash	Matern32 +RBF
Water	Exponential
Superplasticizer	RBF+Matern32+Exponential
Coarse Aggregate	Linear
Fine Aggregate	Rational_Quadratic + Matern32

Figure 10. Member kernels of final kernel

```
#Cement
K1=GPy.kern.Matern32(1,ARD=True,variance=4169.465933438534,lengthscale=1549.9147563919944,active_dims=[0])

#Blast_Furance
k1=GPy.kern.PeriodicExponential(1,variance=9.361692650954631e-
07,lengthscale=147.2139294986865,period=19.101741117437783,active_dims=[1])
k2=GPy.kern.RBF(1,ARD=True,variance=189.65509369615947,lengthscale=214.5073292829563,active_dims=[1])
k3=GPy.kern.Matern32(1,ARD=True,variance=112.94406169745655,lengthscale=1.3663048254769852e-95,active_dims=[1])
K2=k1+k2+k3


#Fly_ash
k4=GPy.kern.RBF(1,ARD=True,variance=1221.6749702362074,lengthscale=1132.6018572227545,active_dims=[2])
k5=GPy.kern.Matern32(1,ARD=True,variance=11.712243768585925,lengthscale=0.058175069229113856,active_dims=[2])

K3=k4+k5

#water
K4=GPy.kern.Exponential(1,ARD=True,variance=1445.9271756267851,lengthscale=50.55227037981031,active_dims=[3])


#Superplasticizer
k6=GPy.kern.RBF(1,ARD=True,variance=1304.0943832273217,lengthscale=50.9404018497552,active_dims=[4])
k7=GPy.kern.Matern32(1,ARD=True,variance=1164.1031437031368,lengthscale=327.4938419600094,active_dims=[4])
k8=GPy.kern.Exponential(1,ARD=True,variance=49.39428716580325,lengthscale=1.2396576667342685e-11,active_dims=[4])
K5=k6+k7+k8



# coarse aggregate
K6=GPy.kern.Linear(1,ARD=True,variances=0.001338907851601497,active_dims=[5])


# fine aggregrate
k9=GPy.kern.RatQuad(1,ARD=True,variance=1185.4575468073592,lengthscale=0.08254820314509405,power=3.9762096123034e-
55,active_dims=6)
k10=GPy.kern.Matern32(1,ARD=True,variance=101.32714527577565,lengthscale=1.759107268443224e-27,active_dims=[6])

K7=k9+k10

#Age
K8=GPy.kern.PeriodicExponential(1,variance=197.98490097302098,lengthscale=0.3662920954077468,period=9.733693649716308,act
ive_dims=[7])
```

Figure 11. The Final Kernel obtained for given data-set

7. Performance Comparison

After generating the customized kernel, we applied various other kernels on the data-set. Finally, we choose 4 different kernels based on the time taken to optimize and the RMSE obtained. In fact, we even used ridge regression to compare the performance of parametric model with non-parametric models. Please check Figure (12) which shows the performance evaluation of all these models.

Kernel used	RMSE observed	Time taken to optimize our model
RBF	7. 766599129924104	30.23 seconds
Matern32	7. 301972258490346	44.42 seconds
Additional Model(custom)	5.477313270855048	95.97 seconds
Multiplied model(custom)	198.09449602868352	119.10 seconds
Ridge Regression	7.77920530176781	1.009 seconds

Figure 12. Performance Evaluation of all models used with 900 training points

We can see that the model based on our kernel gives the best performance in terms of RMSE. However Ridge Regression is the fastest model. But if we ignore parametric models,both RBF and Matern32 are really fast compared to the customized kernel. Kernel obtained by multiplication really ill-performed both in terms of time and RMSE. So which kernel is best? Well, the answer to this question depends on the requirement of user. In case if he wants the best RMSE value and is ready to wait few seconds longer, customized kernel should be his preferred choice. However, if he don't want to wait, but can compromise on RMSE by some few values, then he can opt for RBF or Matern32 kernel.

Now, we applied another performance evaluation. In this case, we reduced the number of training data from 900 to 700 and again compared the performance of top 3 non-parametric models ie the ones obtained using customized kernel, the RBF and the Matern32.

Kernel used	RMSE observed	Time taken to optimize our model
RBF	13.769431089002744	11.23 seconds
Matern32	12.151202404303396	15.42 seconds
Additional Model(custom)	6.994940190967806	65.97 seconds

Figure 13. Performance Evaluation with 700 training points used

Figure(13) actually shows the real performance difference between the models. When we reduced the training data size, though the time taken to train model reduced for all three models,however, the RMSE for matern32(with ARD=True) and RBF(with ARD=True) increased. In fact, it became twice as it was initially, whereas for the customized kernel, it increased only by 2(approximately). Thus, we feel that the additional property of combining kernels is allowing our model to make predictions for values which might be far from training data points. This figure shows that our model is able to perform well even after getting trained on relatively small data-set .

8. Initialization of Hyper-Parameters

The training phase or a parametric model is not very different from that of a non-parametric model. In parametric models, we aim to find the right values of weights, which will in turn minimize the cost-function, whereas when it comes to non-parametric models, the focus is on hyper-parameters of kernel. Here,we aim to find the values of kernel's hyper-parameters which will best fit for the given data. In fact, both parametric and non parametric models use gradient descent. The issues faced while performing gradient descent is well known. The path to optimal minima is not easy. We face many local minimas.

There are fair chances that hyper-parameters, while trying to reach the local minima may fell into local minimas and stop any further convergence, feeling, they have reached the optimal minima. However, these values are not optimal and thus hinders the model from giving their best performance. Thus it becomes utterly important to try to initialize the initial values of these hyper-parameters from a good position. We agree that its very tough to get the right value for initialization. However, here, we compared two models whose initial initialization were very different. In case one, we randomly choose values for every hyper-parameters whereas in case two, we used those values which we got when we were preparing the model, ie taking every single feature and applying various kernels on it to check which one fits it best.While optimizing those models, we got values of their hyper-parameters which were giving optimal performance. We choose those values here by considering them as a better initial values for hyper-parameters.

The results were quite different. Though the time taken by both resultant models to optimize was almost same, ie 65.5 seconds, the RMSE values were way apart. RMSE for case one model was 14.43719158325 whereas for case two model, it was 6.99494019. Here, 700 training data points where used to train the model.As seen in figure (11), we have many member kernels in our customized model, so its not possible to show all their initial and final values.Thus, we are showing values of some of them. Please check figure(14), figure (15) and figure(16) for performance evaluation of these two models.Please note that figure(14) shows case one , the left side of which shows values of hyper-parameters before optimization and the right side, after optimization. Similarly, figure(15) shows both scenarios of case two.

Figure(16)shows the performance comparison of both models.

Thus the above observations show how initialization of hyper-parameters can change the performance. Thus, we would recommend to try different ways and values for initialization of these hyper-parameters.

9. Interpretability of models

This section will focus on interpretability of models. interpretability is a very important part of model development. Its the way,by which the model developer can explain the results to user. Interpretability is the concept of explaining the reason for outcome. As it implies, its all about explaining how the model is predicting a particular value or classifying the input to a particular class.It describes what is the impor-

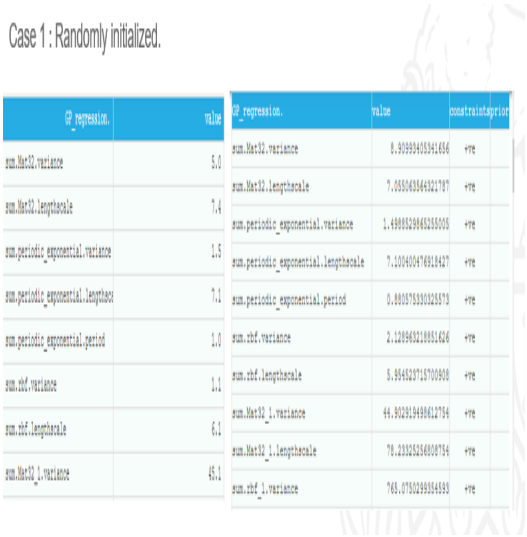

Figure 14. Random-initialization before and after optimization

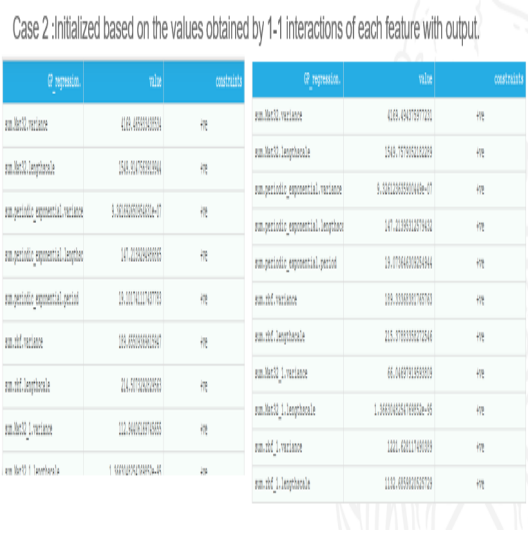

Figure 15. Initialized based on the values obtained by 1-1 interactions of each feature with output.

tance of every feature, which feature,among the set of features are more involved (either directly or inversely) when generating the result. The project was developed in GPy and lucky for us, there is a functionality called ARD(Automatic Relevance Determination) which depicts the importance of each feature. Its the interpretability which provides transparency to the model. It makes understanding and hence manipulation of model easy. This concept is ,missing in deep learning. This in turn forces us to look a deep learning model as a black box,because we don't know what is happening inside the model. How the features are getting selected,which feature is given more importance and at the end, if someone asks why a particular value was predicted or the outcome was a particular class, we don't have any

Model	RMSE
Randomly initialized model	14.43719158325
Model obtained by 1-1 interactions of each feature with output	6.9949401990967806

Figure 16. Initialization comparison

answer. ARDs in Gpy,however has a catch. They can be applied only to stationary kernel based models. Thus ,it can not be used on kernels like linear Kernel, which is a non stationary kernel. This is a bad news for our kernel. As its a combination of both stationary and non-stationary kernels, we cant apply ARD on it. However, in order to understand ARDs , we applied them on two stationary kernels, ie RBF and Matern32. In fact, we applied it on two different sized data-sets. First case has 700 data-points and case two has 500. Please check figure(17) and (18) for performance evaluation.

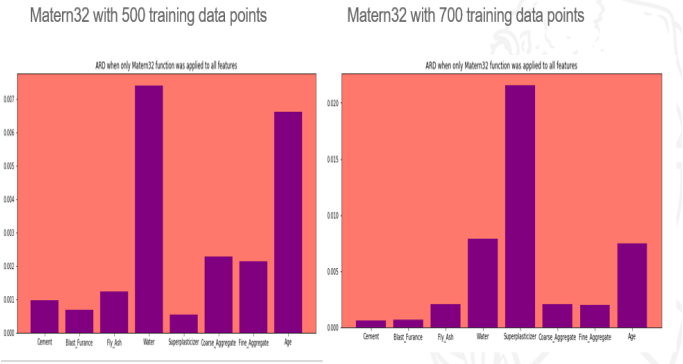

Figure 17. ARD evaluation on Matern32 with 500 and 700 training data points

Please note the right side of both figure (17) and figure(18) show ARD evaluation of model trained on 500 training data points whereas the other side shows ARD evaluation of model trained on 700 training data points. We find difference in ARD evaluation as training size changes. By the term evaluation here, we mean the importance given by ARD to various features. This gives a hint about the type of data. We feel that there is not only one feature which is dominating the results and in fact, composition of data-set is not uniform. Few data-points might have constant or null value for a feature whereas others may have varying values. It may be the case the in our 500 selected

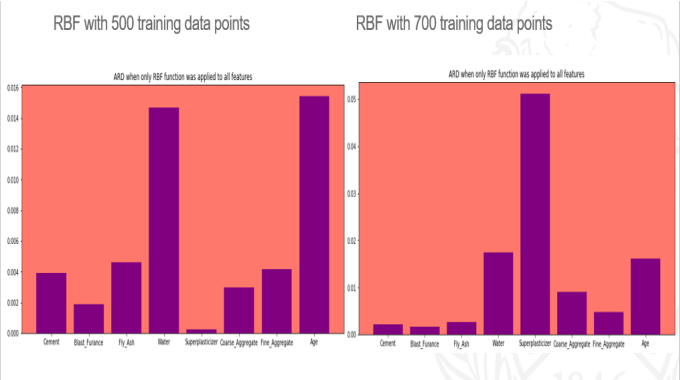

Figure 18. ARD evaluation on RBF with 500 and 700 training data points

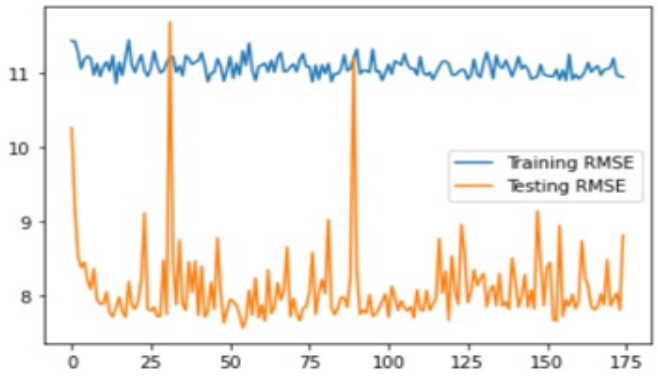

Figure 19. Deep Learning vs proposed model

data-points,a majority of points might have that feature's value either constant or zero. Thus model was not giving it much importance, whereas when the same model encountered more training data,it found varying values for that feature and found its value is quite important for predicting the outcome. Here, we were talking about super-plasticizer feature.

Thus we can assume that ARD not only explains the model, but also gives hint about the distribution of data.

10. Deep Learning vs Parametric models

We tried to compare the performance of a simple deep-learning model to that of our proposed model. This data-set is in fact an ideal case to prove that deep-learning can't be used to solve all the problems. The fundamental concept of deep learning is to gather huge amount of training data, create a deep model and apply regularization to generalize it well. These, in turn helps the model to catch hidden interactions between various features of data-set and finally, based on those interactions, predict the outcome. However, here, the data size is very small and even a simple deep-learning model won't perform well. In fact, to prove this point, we compared its performance with our model. Please check figure(19) for performance comparison of a deep learning model with that of our model.

Figure (19) shows the test vs training accuracy of a fully connected 3 hidden layer deep model. Note that even though its a very simple model, its performance is fluctuating. If we closely observe, the RMSE value is far greater than that obtained by our model. In fact, for some epoch size, it goes above 12. Thus it can be concluded that deep learning models may not be the best choice in these cases . Deep learning models can be rather used in situations where the training data size is really large (and thus non parametric models due to nearly cubic time complexity will take very long time to optimize.)

11. Applying Sparsity to reduce optimization time

Though Parametric models give really good accuracies, the time taken by them to train a model is a big reason why they are not so prominently used. In fact, the time complexity of these models is nearly

$$0(N^3)$$

(even after applying various techniques), where N represents the training data size. We can only imagine the time taken to train a model which has like 10k training points. This is simply not feasible.

In this section, we will use the concept of sparse input to solve this problem. Using sparsity, we will create fewer training points ie "induced inputs" which will try to capture characteristics of original training data. This will thus, reduce the training size to train the model and greatly reduce the required time. Initially we created 20 new data points and the results where encouraging. In fact the time taken to generate these new training points and train the model was about 40 seconds and RMSE achieved was 8.4925271519683365. Please note that here, initially 700 training points where taken. Then we started to increase the number of induced inputs by factors of 5. We started on 20 and went till 50 and figure(20) and figure (21) display the results obtained.

Thus it can be observed that the time taken to create new data points and optimize the model is not in any order. Whereas RMSE seems to be going down as the number of created data points increase.

After seeing this, a question rises, if usage of sparsity is giving such good results in comparatively less time, then why not always use it ? Well there are few issues with this technique too. First, it all depends on the induced set to capture properties of given training data-set. In case if it fails to do so, then, the whole model fails. Further, like ARD, it can not be applied to all kernels. Also the time taken to create induced data points and optimize the model

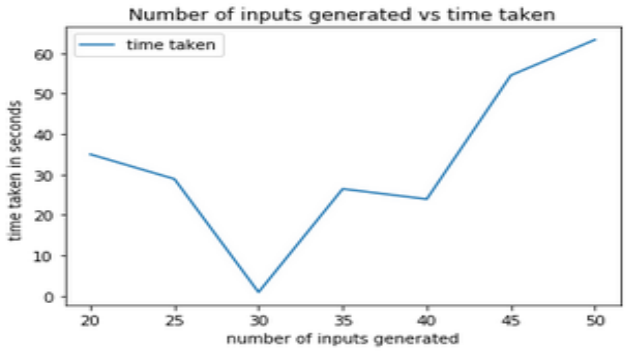

Figure 20. time taken when using sparse data points

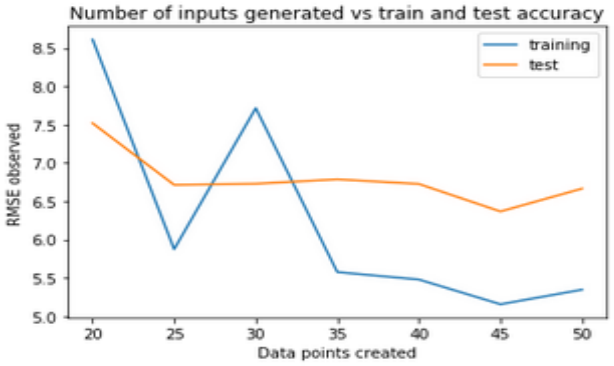

Figure 21. RMSE obtained when using sparse data points

seems to be random. Sometimes, it takes long time to do so, whereas, other times, it gets converged really quickly. This can be seen in figure (20) where , in general, the time taken should increase as the number of "induced inputs" increase (here there are two tasks. First, time taken to create these induced points, which should increases when number of induced points increase and second, the time taken to optimize the model which should decrease greatly compared to when we use whole training data.). We are in fact doing more research on why this is happening and which properties of training data does these induced inputs are trying to learn.

12. Conclusion

This project taught us many things like what is kernel, how to combine kernels, how to create personalized kernels for any real world data-set, how to check performance of models created by these kernels, usage of ARD in expressing the interpretability of any model, the importance of proper initialization of hyper-parameters, usage of sparsity in reducing the time required to optimize the model etc. This project can be useful to solve many real world problems as it explains how to create kernels, customized for a given data-set. This project is specially useful for those cases in which, we don't have huge training data-set. This was verified by comparing performance of a simple deep learning model with the developed model.However, there are few issues, like making a customized kernel is tough. We need lot of experience and understanding to create these kinds of kernels.Also though sparsity may have brought down the time taken to optimize the model, we saw it has its own issues like inconsistency. Hence we can say that non-parametric machine learning is a very powerful tool and can solve many real life problems.However in order to use them to their full potential, we need to have thorough understanding of kernels and few other basic concepts.

13. Acknowledgement

We would like to end this report by thanking our mentor Dr Varun Chandola for encouraging us to do this project and giving his valuable feedbacks. It wouldn't have been possible without his support,guidance and encouragement. Also we are thankful to Dr David Kristjanson Duvenaud for this amazing thesis titled "Automatic Model Construction with Gaussian Processes" which is a treasure to read. We got many of our doubts cleared by reading his thesis.

14. References

All the references are mentioned at the footnotes.Also names of the people who directly or indirectly helped us in this project are mentioned in the Acknowledgement section.