Machine Learning Engineer Nanodegree

Capstone Project

Ambuj Agrawal
April 20th, 2019

I. Definition

Project Overview

I work in Qualcomm as a Modem Engineer and I wanted to develop an AI platform which can be used to catch common UE (user equipment-more commonly known as mobile phones or cellular phones) issues. Modem is the chip which allows a mobile phone to communicate wirelessly with a base station so that a user can receive/send data or voice calls.

Right now, for catching these issues we write codes that will contain lots of if-else statements or will have lots of hard-coded values and so they are not easily scalable. We will also look at different plots and manually try to find abnormalities and then debug why such an abnormality occurred.

My goal with this project is to teach an AI agent the work that we Engineers do manually which will result in lots of time saving. I want it to learn some basic relationships between different metrics and figure out if an abnormality occurred or not and if it did then what was the root cause. This way it can learn few of the basic things that I as an engineer learned when I started working in this area. Over time, we can enhance the agent with more complicated scenarios to make it a lot more sophisticated.

Problem Statement

The AI agent needs to find abnormalities in a data set and also needs to figure out why such an abnormality occurred. We will have 4 key metrics that we will use in this project-

1) SNR (signal to noise ratio)-This basically reflects how good or bad the wireless channel being seen by the UE is.

2) CQI (Channel Quality Indicator)-UE measures the SNR that it is observing and feeds back this information to the BS (base station) in the form of CQI. Higher the SNR, higher the CQI and vice-versa.
3) MCS (Modulation and Coding Scheme)-Based on the CQI report received from the UE, the BS will choose an appropriate MCS to use while sending data to the UE. Higher the CQI, higher the MCS and vice-versa.
4) Throughput-The MCS selected by the BS will directly map to a transport block size (TBS) based on the amount of frequency resources available. In this project we will assume that frequency resources assigned to the UE are constant so TBS with respect to a particular MCS will be constant. Throughput will be either equal to TBS or 0 depending on whether UE was able to decode the data or not.

The AI agent needs to figure out if observed Tput makes sense or not. If it doesn't then it needs to flag it along with what it thinks is the root cause of the problem.

We can have 2 abnormalities-

1) Tput is 0 (say BS should have chosen MCS 25 but it ended up choosing MCS 26 and so the UE was not able to decode the data)
2) Tput is lower than expected but not 0 (say BS should have chosen MCS 25 but it ended up choosing MCS 24 and so the UE received lesser data than expected)

The root cause for abnormality can be any of the following-

1) SNR and CQI are correct but MCS chosen is wrong
2) SNR is correct but CQI chosen was wrong resulting in lower/higher MCS than expected for the corresponding SNR
3) SNR estimation itself was wrong which resulted in lower Tput

Basically, the agent will need to learn the relationship between SNR⇔CQI and CQI⇔MCS. Additionally, it will also need to monitor SNR and if it sees abrupt jumps in SNR across time then it needs to flag that as well.

Evaluation Metrics

We will calculate the $F_{0.5}$ scores for the following metrics-

1) Total number of Errors: All the errors in the entire dataset
2) CQI Errors: All the CQI errors in the entire dataset
3) MCS Errors: All the MCS errors in the entire dataset
4) SNR Errors: All the SNR errors in the entire dataset

We are choosing $F_{0.5}$ scores as we care more about precision than recall. It's fine if the model doesn't catch few errors since there will be multiple errors of the same kind in the dataset and even if one of them is caught then it will uncover an issue that needs to be fixed.

But on the other hand, if we have good samples being tagged as errors than it may lead to wastage of time and energy as it's possible that the dataset was error free but some samples were tagged as faulty and then then Engineer had to open the dataset and verify if there indeed was any issue present or not.

In this sense, the problem that we have more resembles that of spam detector, false negatives are ok, but false positives are not fine.

II. Analysis

Data Exploration

As I don't want to use proprietary data, a simple model will be used to generate the dataset. The model will be close enough to what is seen in the real world.

Following relationships will be used to generate data-

SNR	CQI	MCS	TBS
0	0	0	0
1	1	2	2000
2	2	4	4000
3	3	6	6000
4	4	8	8000
5	5	10	10000
6	6	12	12000
7	7	14	14000
8	8	16	16000
9	9	18	18000
10	10	20	20000
11	11	22	22000
12	12	24	24000
13	13	25	25000
14	14	26	26000
15	15	28	28000
16	15	28	28000
17	15	28	28000
18	15	28	28000
19	15	28	28000
20	15	28	28000

Table 1: SNR, CQI,MCS and TBS relationships

SNR will be varied from 0⇔20⇔0 at 2 units per sec for 100 seconds. Data will be generated every10ms. That will constitute one dataset. We can have as many datasets as needed based on how well the agent is learning.

In each dataset, approximately 1.5% of the samples will be corrupted: 0.5% CQI samples, 0.5% MCS samples and 0.5% SNR samples. Example of each of the three corruption scenarios is shown below-

1) CQI corruption: Say the SNR was 13, so from the table above UE should report CQI 13, but this value will be corrupted, and UE may report say CQI 10. Thus, it will be allotted MCS 20 only resulting in TBS/Throughput of 20000 only instead of 25000 which it would have got had it reported the correct CQI.
2) MCS corruption: Say the UE reports CQI 12 but BS somehow interprets it as 13 and thus allocates MCS 25 to the UE. As this MCS is higher than what UE asked for, it will result in UE not able to decode the data as SNR is not good enough to receive such a big TBS. Thus, throughput in this case will be zero.
3) SNR corruption: SNR variation should be smooth as we are varying it at a constant rate of 2 units/per sec. But there can be instances where SNR jumps happen. For ex, SNR goes like this- 9.00,9.02,9.04,5.42,9.06,9.08

The corruption will happen randomly using Gaussian noise for SNR and random integers for CQI and MCS as CQI and MCS are always integer values.

Exploratory Visualization

As can be seen from the relationship table provided in the "Datasets and Inputs" section, most of the relationships are linear. The SNR⇔CQI relationship is linear up to a point and then it flattens out as shown below.

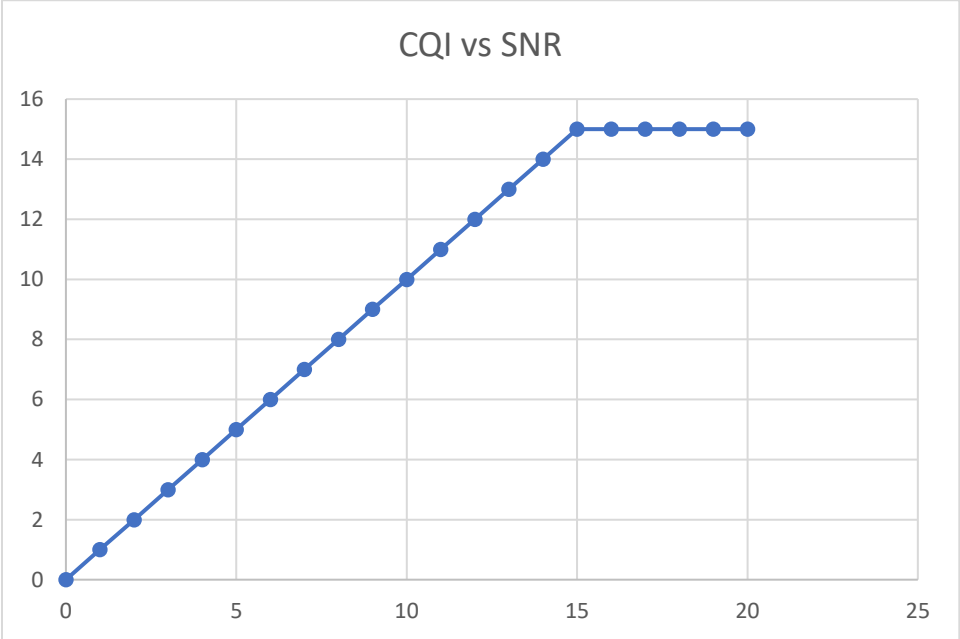

Fig.1: CQI vs SNR

But CQI⇔MCS and MCS ⇔ TBS relationships are almost linear as shown below.

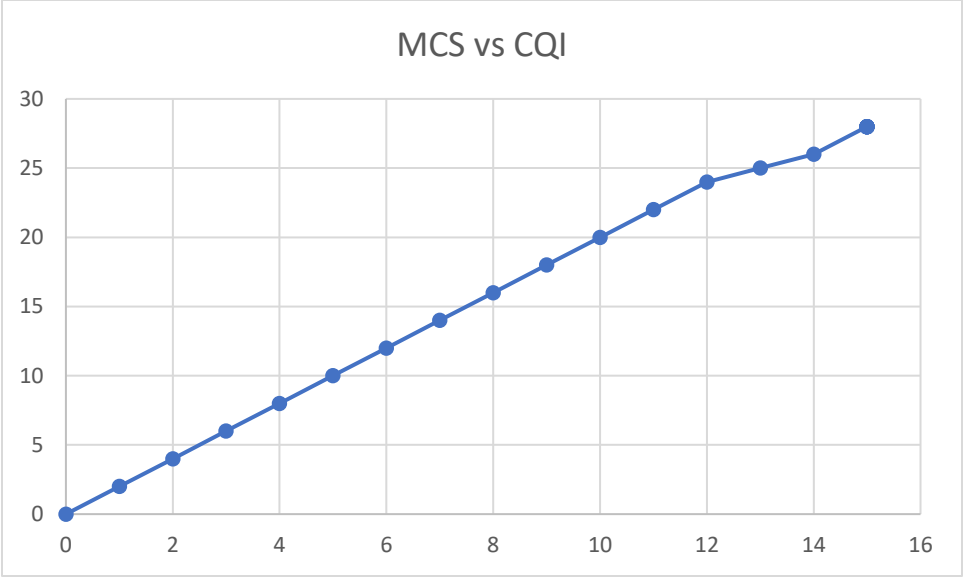

Fig.2: MCS vs CQI

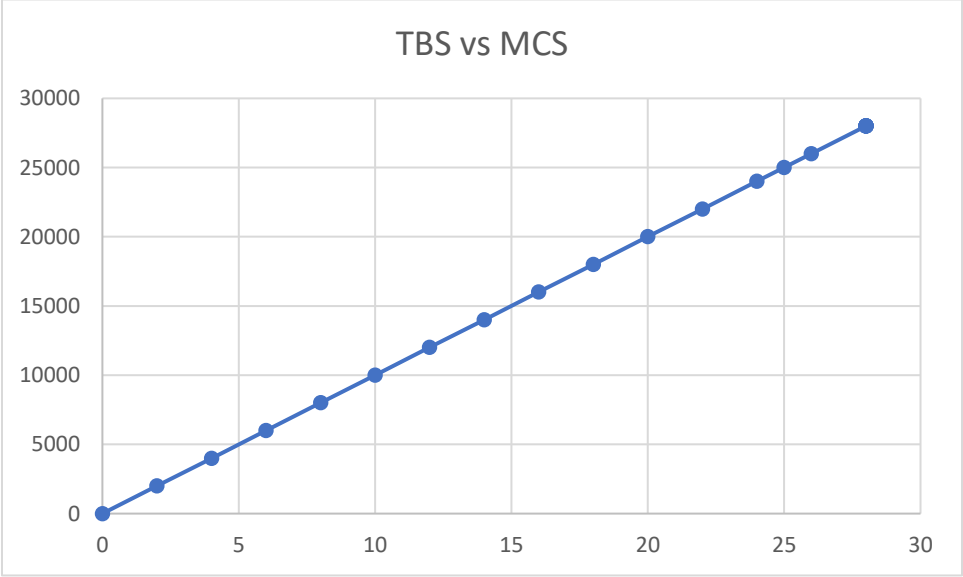

Fig.3: TBS vs MCS

Algorithms and Techniques

The following algorithms and techniques were used in solving the problem-

1) DBSCAN was used to figure out if SNR is varying smoothly or not. Any sudden jumps in SNR should be caught with the help of the clustering algorithm
2) The SNR and CQI relationship were learnt through SVC and neural network. It was seen that SVC was faster and performed better.

3) The CQI⇔MCS and MCS⇔TBS relationship were learnt through SVC because of the results obtained in step (2)
4) Once the above relationships were learnt by the AI agent, then we went about finding the errors in the following manner-
    a. Find out if a particular SNR point is part of a smooth transition or a result of a sudden jump. Flag it if it is a sudden jump.
    b. If SNR point is good then predict CQI value. If the difference b/w predicted and actual value is greater than a threshold then flag it.
    c. If CQI is also good then predict MCS value. If the difference b/w predicted and actual value is greater than a threshold then flag it.
    d. If MCS is also good then predict TBS value. If the difference b/w predicted and actual value is greater than a threshold then flag it.

Benchmark Model

As a benchmark model, we will use K-Nearest Neighbor algorithm for finding out a jump in the SNR and a linear regression model for learning and predicting other relationships- SNR⇔CQI, CQI⇔MCS and MCS⇔TBS.

It was found that KNN or any other classification model except for DBSCAN for that matter performed horribly when trying to find out jumps in SNR. KNN, Agglomerative Clustering and GMM results are provided to illustrate this.

The output from linear regression model was rounded to an integer value and clipped so that it's within the range of values taken by CQI and MCS. The $F_{0.5}$ score was very low around 0.01.

III. Methodology

Data Preprocessing

Data was generated as outlined in "Data Exploration" section. Resulting Datasets are shown in this section.

In the below plot, SNR was varied at 2 units per sec, time was varied from 0 to 100 in steps of 0.01 seconds. Then Gaussian noise with mean -10 and variance 1 was added to 0.5% samples of the clean SNR.

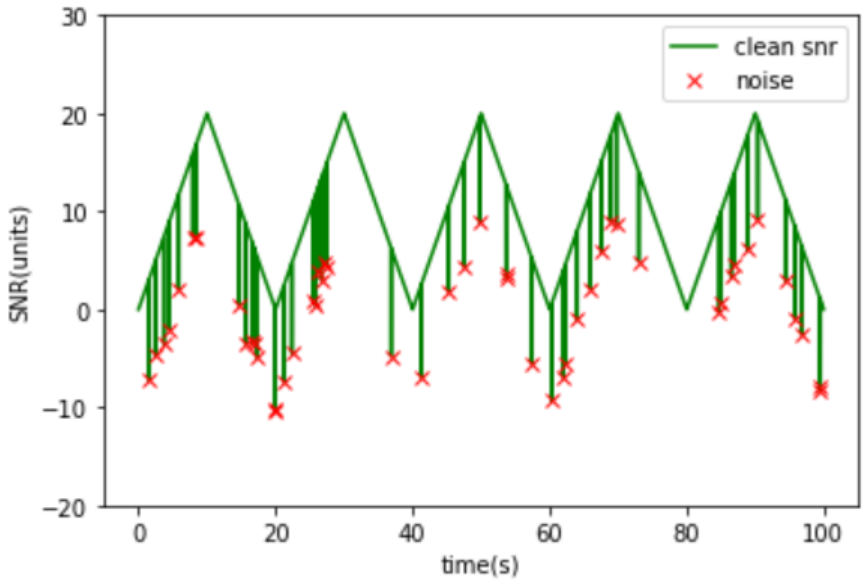

Fig.4: SNR with noise v/s time

The below plot representing CQI v/s Time was generated using the SNR⇔CQI relationship as described in "Data Exploration" section. Then 0.5% of the samples were corrupted with noise by using randint function with range [-5,5]. Please note that some samples seem noisy but are not marked as noise since they resulted from the noisy SNR samples.

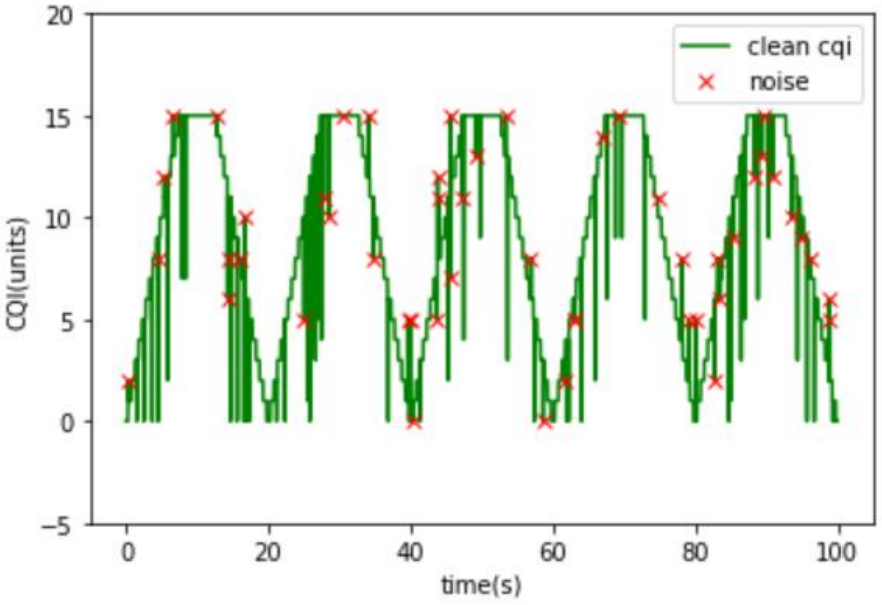

Fig.5: CQI v/s Time

Similarly, the MCS v/s Time plot is shown below.

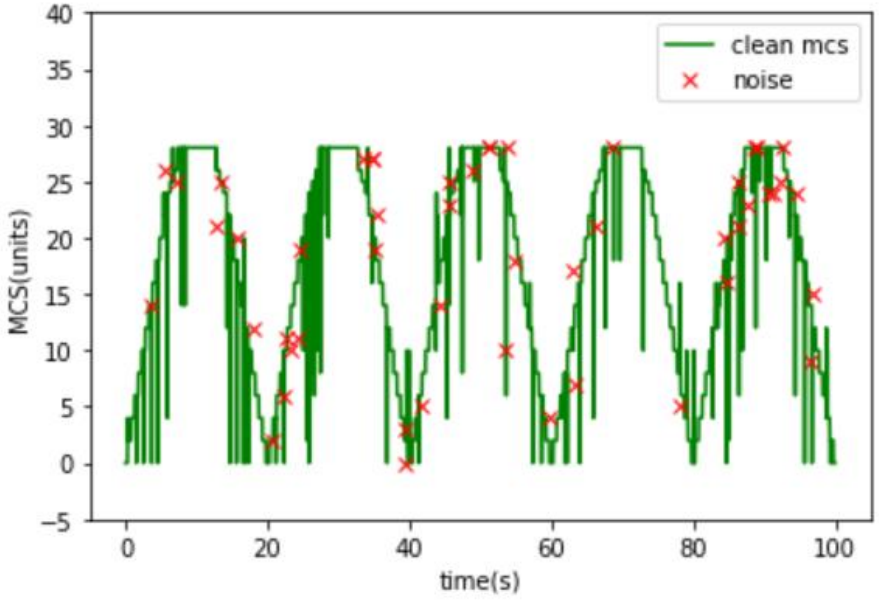

Fig.6: MCS v/s Time

In the TBS v/s Time plot shown below, the noise samples are not marked. The algorithm needs to find out the noisy samples and classify the error as SNR error, CQI error or MCS error.

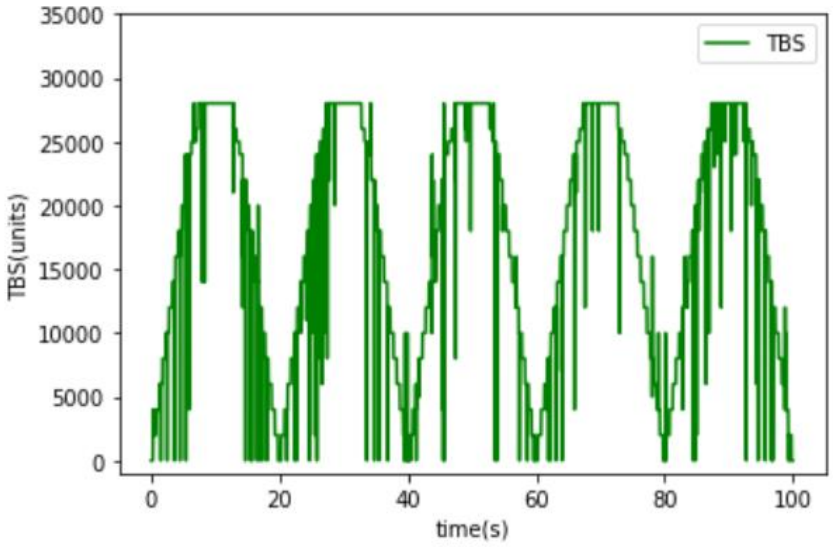

Fig.7: TBS v/s Time

Implementation

For finding out the jumps in SNR, DBSCAN algorithm was used and the default params (eps=0.5 and min_samples=5) worked very well. DBSCAN was able to filter out all the noisy samples. $F_{0.5}$ score obtained was 1!

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_38 (Dense)             (None, 32)                64
_________________________________________________________________
activation_38 (Activation)   (None, 32)                0
_________________________________________________________________
dense_39 (Dense)             (None, 32)                1056
_________________________________________________________________
activation_39 (Activation)   (None, 32)                0
_________________________________________________________________
dense_40 (Dense)             (None, 16)                528
_________________________________________________________________
activation_40 (Activation)   (None, 16)                0
=================================================================
Total params: 1,648
Trainable params: 1,648
Non-trainable params: 0
_________________________________________________________________
None
```

Fig.8: Neural network model for finding SNR⇔CQI relationship

For learning the SNR⇔CQI relationship, a neural network was used first. The activation function used was ***tanh***. Regularization and Dropout were also tried but they didn't improve the performance much so they were removed from the final model.

It was seen that performance for neural network wasn't great so SVC was tried next and it gave much better performance. Therefore, the final implementation for both SNR⇔CQI and CQI⇔MCS was done using SVC.

For calculating the $F_{0.5}$ score, the use of sklearn fbeta_score function was not possible as there were 3 different arrays that needed to be compared-

1) y_clean: array containing the clean values of a particular metric (for ex SNR)
2) y_dirty: array containing the clean and noisy samples of a particular metric
3) y_pred: array containing the predicted values of a particular metric

The confusion matrix looked like this-

actual_positives=y_clean-y_dirty

total_positives=y_pred-y_dirty

true_positives=total_positives ∩ actual_positives

false_positives=total_positives-true_positives

false_negatives=actual_positives-true_positives

		Prediction	
Actual		*Error*	*No Error*
	Error	True +ve	False -ve
	No Error	False +ve	True -ve

Table 2: Confusion Matrix for proposed model

Refinement

As was mentioned in the previous section, when trying to learn relationships between different metrics, neural network didn't perform very well. The accuracy was high (>98%) but the $F_{0.5}$ score was low (~0.26). Therefore, other solutions were tried, and SVC performed really well. Accuracy became marginally better (~99%) but $F_{0.5}$ score increased to around 0.6 from 0.26 for SNR⇔CQI. For CQI⇔MCS it was a perfect 1. So, SVC seemed like a really good solution and hence in the final implementation SVC was used instead of neural networks. Another advantage of SVC was that it was pretty fast compared to neural networks.

IV. Results

Model Evaluation and Validation

For finding out jumps in SNR, DBSAN algorithm was used as it's one algorithm that can filter out noise samples really well and thus seemed most suited to the problem at hand. The results were great as well, it was able to filter out all the noise samples and $F_{0.5}$ score obtained was 1.

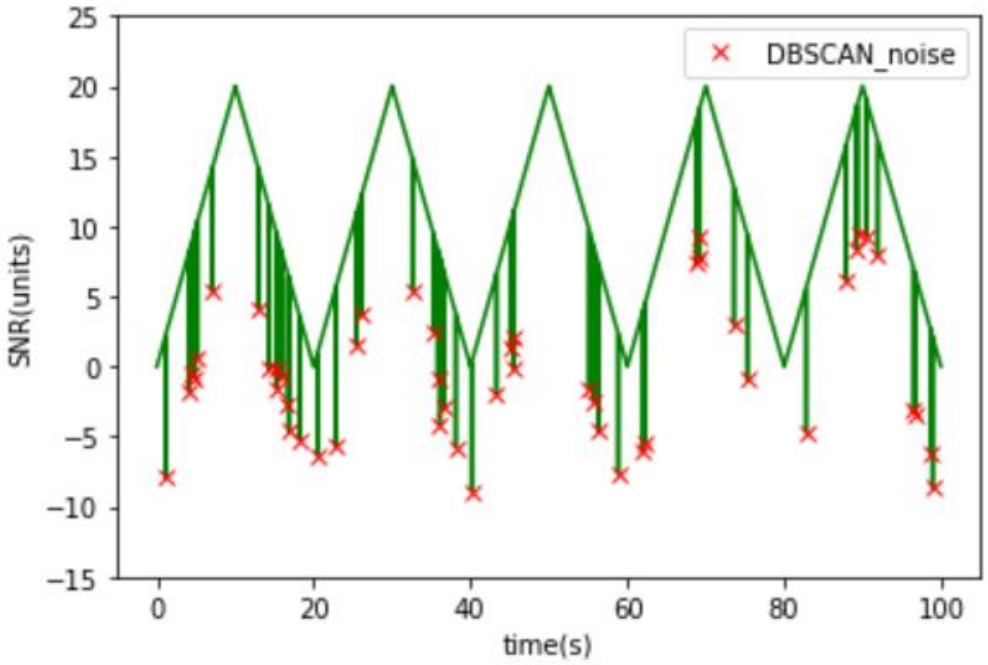

Fig.9: Performance of DBSCAN algorithm on the SNR dataset

For learning SNR⇔CQI relationship, neural network and SVC were tried. Please find a comparison of accuracy and $F_{0.5}$ score for both the algorithms below.

	Neural Network	**SVC**
Accuracy	0.98	0.99
$F_{0.5}$ Score	0.26	0.60

Table 3: Performance of SVC and Neural Network model for SNR⇔CQI

It seems like neural network did a good job in finding the CQI errors which were greater than 1 unit but it did was off by one unit on quite a few occasions.

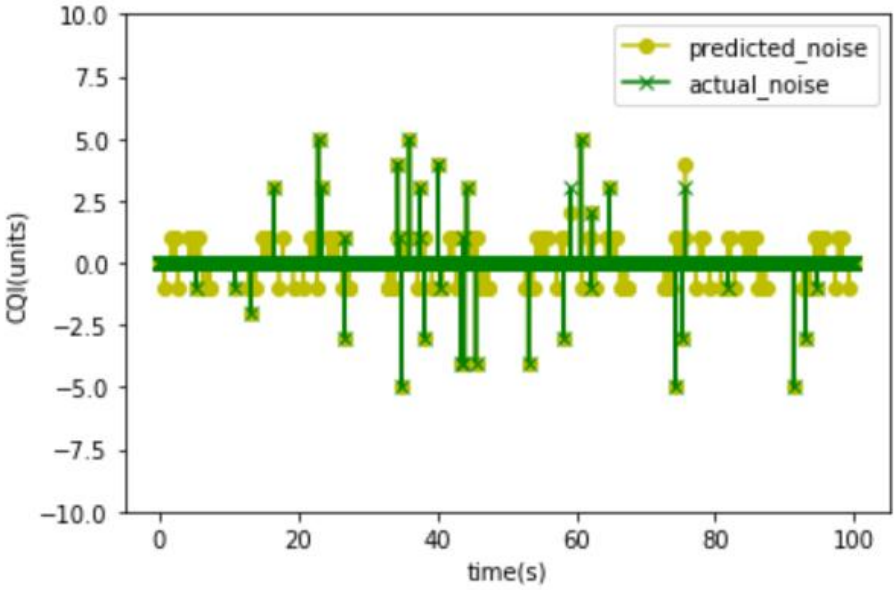

Fig. 10: Performance of Neural Network in predicting noise for CQI

When compared to neural network, SVC performs much better as shown below. There are a lot less instances where the algorithm is off by 1 unit which leads to higher $F_{0.5}$ score.

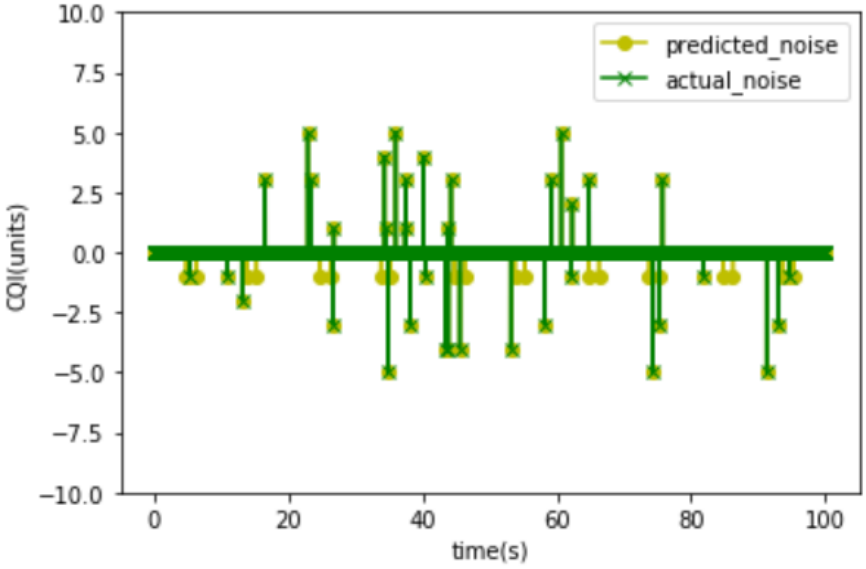

Fig. 11: Performance of SVC in predicting noise for CQI

For MCS, SVC does a real good job and is able to find all the errors without misclassifying a good sample as a noisy sample and thus $F_{0.5}$ score is a perfect 1.

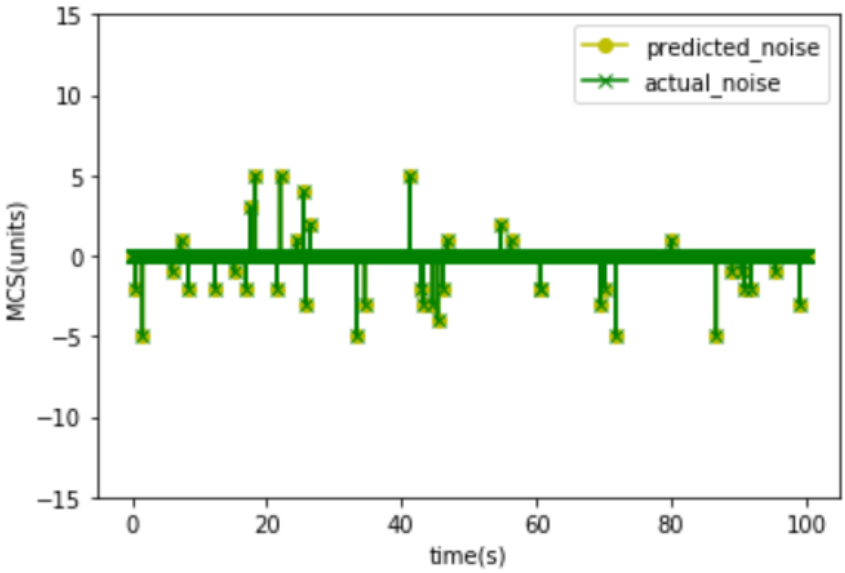

Fig.12: Performance of SVC in predicting noise for MCS

SVC is also able to predict very well as shown below, CQI values only range between 0 and 15 and MCS is predicted perfectly for all those values. The figure below is same as MCS v/s CQI plot shown in fig.2

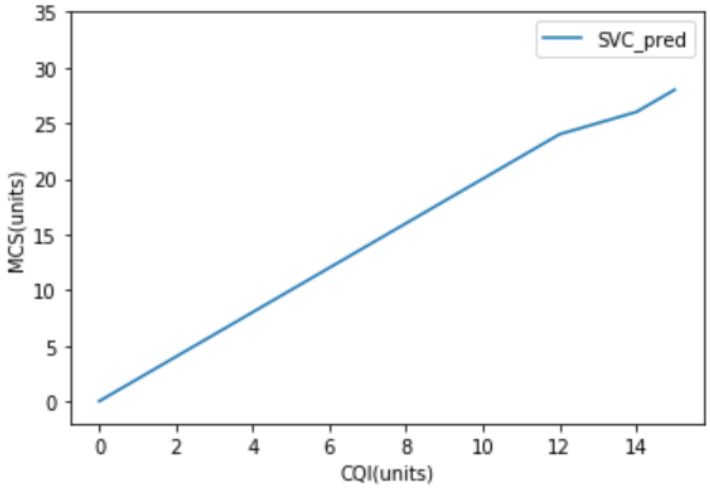

Fig.13: Predicted MCS vs CQI

But when it comes to predicting CQI, seems like SVC is overfitting. Here neural network seems to be doing a lot better. When we extend SNR on both ends (below 0 units and above 20units), SVC and neural network are equivalent when predicting CQI for SNR below 0 units but for SNR above 20 units CQI predicted by SVC goes down to 0 from 15 which is incorrect. Neural network does a real nice job of staying at 15.

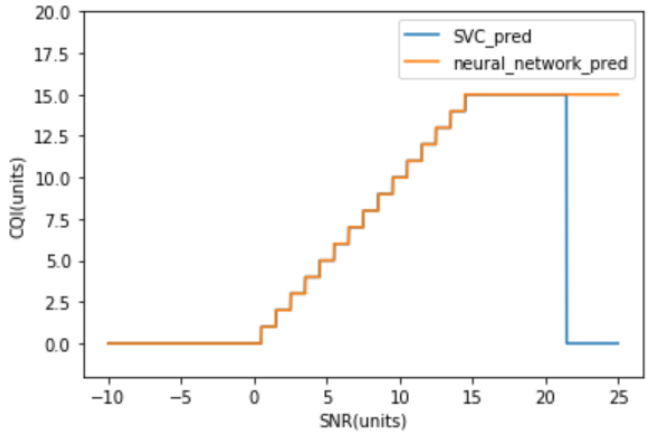

Fig.14: Predicted CQI v/s SNR

Justification

The DBSCAN model to catch SNR jumps works perfectly and when we compare it to benchmark model then we can see that none of the other clustering models come close to DBSCAN's performance. All the other models seem to be classifying half the points as part of one cluster and the other half as part of the other cluster.

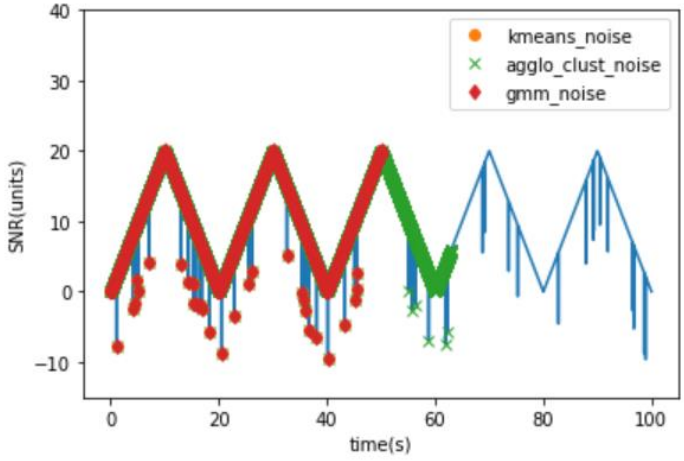

Fig.15: Performance of K-Means, Agglomerative Clustering and GMM wrt finding jumps in SNR

The benchmark models for finding SNR⇔CQI relationship and CQI⇔MCS relationship also suffer from a low F0.5 score. The accuracy also isn't very high.

	SNR⇔CQI	**CQI⇔MCS**
Accuracy	0.53	0.62
$F_{0.5}$ Score	0.01	0.01

Table 4: Performance of benchmark models for SNR⇔CQI and CQI⇔MCS

V. Conclusion

Free-Form Visualization

As DBSCAN has performed really well in filtering out the noise samples, it was again used to find out the noisy samples in the final output that is seen by the end user-TBS (which signifies the throughput received). Again, from the plot it looks like DBSCAN did a perfect job. So, the noisy samples obtained by DBSCAN were treated as ground truth noise samples and the $F_{0.5}$ score for the entire algorithm was calculated based on that. It came out to be equal to 0.71. So, performance isn't too bad, but the model does classify few good samples as noisy samples which results in F0.5 score to be not very close to 1.

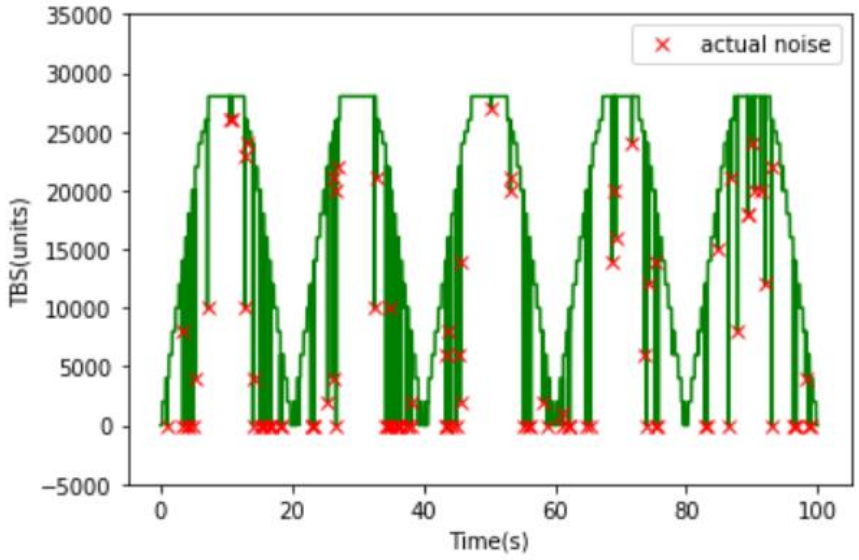

Fig.16: TBS noise samples identified by DBSCAN

As shown in the next plot, most of the noisy samples have been characterized to be due to either SNR issue, CQI issue or MCS issue as was the goal of this project. Also, most of the good samples that are getting classified as noisy samples seem to be because of imperfect noise prediction due to CQI. This is one area which can be improved further.

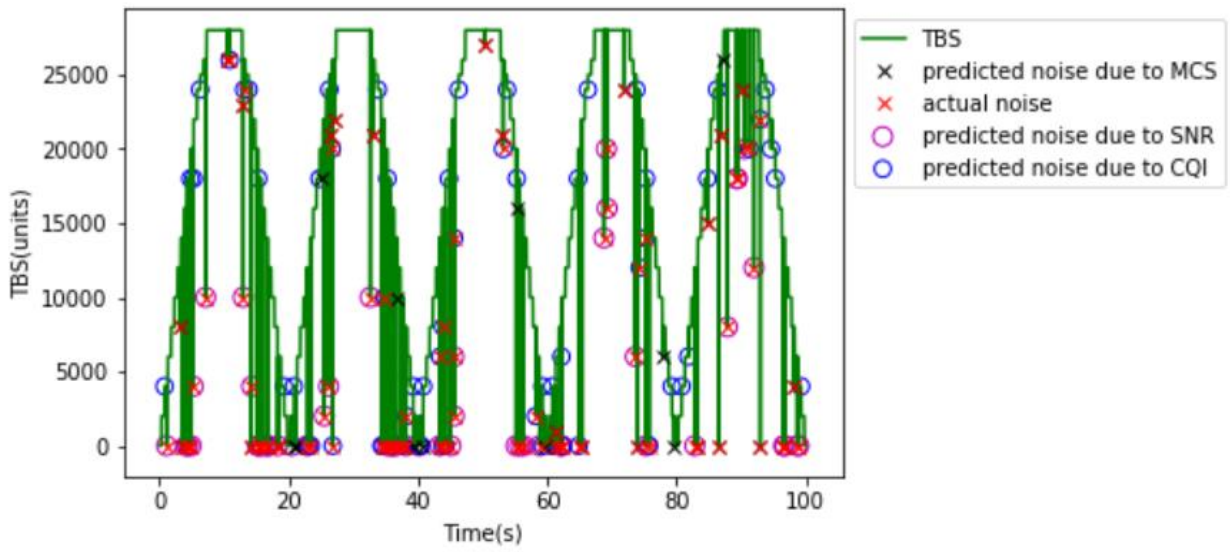

Fig.17: Characterization of noise samples of TBS as predicted by the model

Reflection

The entire end-to-end solution involved the following steps-

1) Catch SNR jumps using a clustering algorithm
2) Learn SNR⇔CQI and CQI⇔MCS relationships using supervised learning algorithms
3) Classifying the noise seen in TBS as coming from either SNR, CQI or MCS

The goal of the project was to touch upon most of the concepts that were learned during the Machine Learning Nanodegree course and also come up with a model that can actually be of help in solving a problem that is seen almost daily by me and my colleagues. I was not able to use CNN (this is the part I regret) and RNN (I had spent plenty of time on the quadcopter project so RNN is still fresh in my mind) as the project didn't really need very complicated models.

The model in the end was able to perform decently well. I realized that I can use DBSCAN to filter out noise samples in all the metrics-SNR, CQI, MCS and TBS. So, in a way there was no need to learn SNR⇔CQI and CQI⇔MCS relationships. But learning these relationships will be useful in other scenarios-for ex say CQI is totally busted and is always reported as 0 no matter what the SNR value is. In this case DBSCAN will not be of much use as all CQI samples have the same value, but if the model knows what the expected CQI is given a particular SNR (because it learned the SNR⇔CQI relationship) then it can point out such issues as well.

Improvement

There is scope of improvement particularly in the case of SNR⇔CQI relationship. One very easy modification that can be done is to classify a sample as suffering from CQI error only when difference between predicted CQI value and received CQI value is more than 1. This small modification increases the $F_{0.5}$ score for SNR⇔CQI to 0.93 from 0.60 and for the entire model to 0.87 from 0.71. The error classification plot also looks much better now as most of the incorrectly classified CQI errors are no longer there.

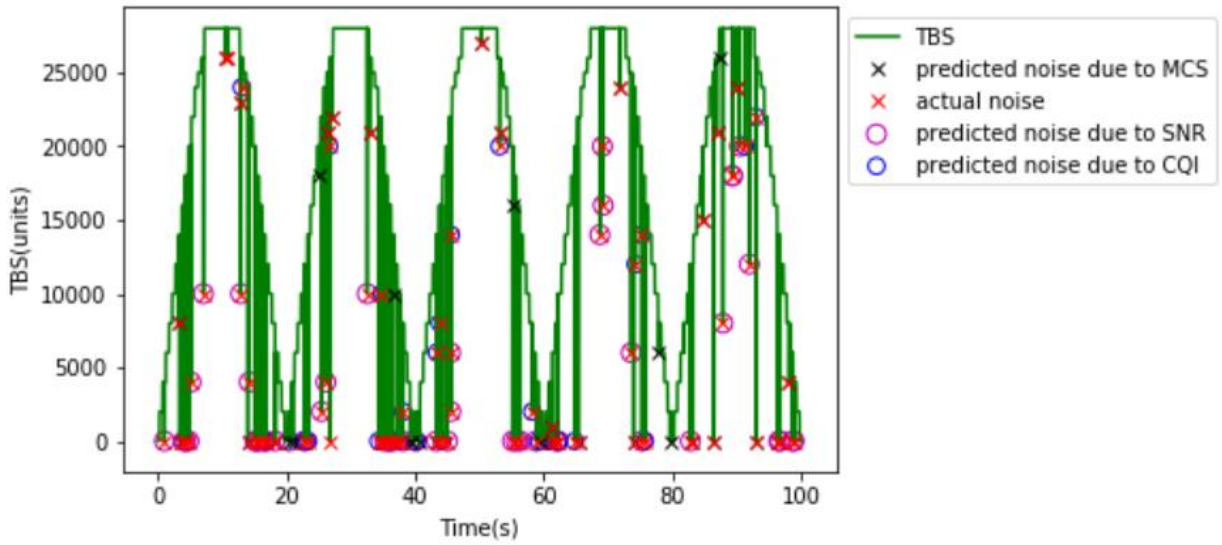

Fig.18: Error classification plot after increasing the threshold for CQI error

It was also seen that SVC was overfitting the data which can also be improved. But this is not very worrying as the SNR range is most of the times within certain limits and the model can be trained on that entire range. Chances of seeing a new SNR value are minimal. Nevertheless, we can fall back on the neural network model in case we really want to avoid overfitting. The $F_{0.5}$ score for that model also increases from 0.26 to 0.93 when we increase the threshold for CQI error classification from 0 to 1.