Problem definition

What is the problem you're trying to solve, why is it important to solve it?

My problem is about prediction HTTP-Response size based on HTTP-Request. I want to design a prediction system based on HTTP-Request headers fields. I will explain my problem with an example, first a brief explanation about HTTP-Request.

What's in an HTTP-Request?

Whenever your web browser fetches a file (a page, a picture, etc) from a web server, it does so using *HTTP* - that's "Hypertext Transfer Protocol". HTTP is a request/response protocol, which means your computer sends a request for some file (e.g. "Get me the file 'home.html'"), and the web server sends back a response ("Here's the file", followed by the file itself).

The HTTP verbs comprise a major portion of our "uniform interface" constraint and provide us the action counterpart to the noun-based resource. The primary or most-commonly-used HTTP verbs (or methods, as they are properly called) are POST, GET, PUT, PATCH, and DELETE. These correspond to create, read, update, and delete (or CRUD) operations, respectively. There are a number of other verbs, too, but are utilized less frequently. Of those less-frequent methods, OPTIONS and HEAD are used more often than others. (More information about HTTP methods)

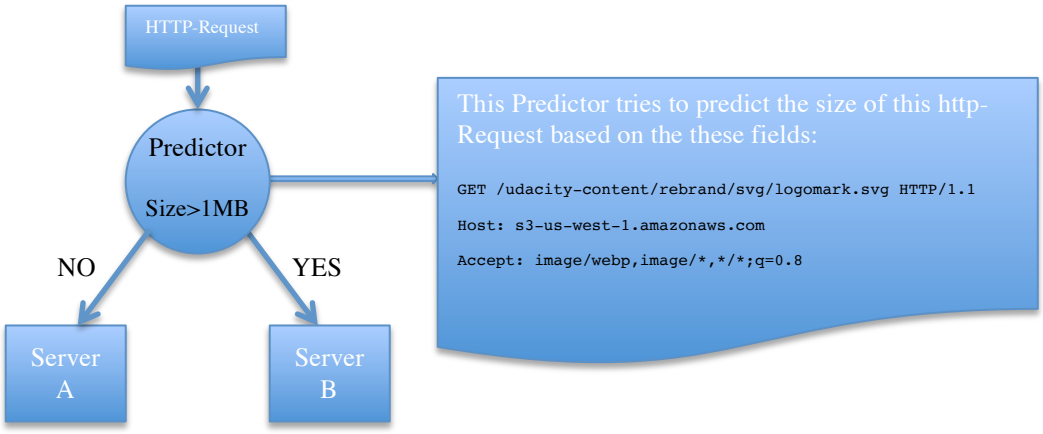
In this report/classifier we only process Get requests.
Example:

- It is an HTTP-Request for Udacity logo file:

```
GET /udacity-content/rebrand/svg/logomark.svg HTTP/1.1
Host: s3-us-west-1.amazonaws.com
Connection: keep-alive
Cache-Control: max-age=0
Accept: image/webp,image/*,*/*;q=0.8
If-None-Match: "5160692df65598202eec0f02e3f76e1d"
If-Modified-Since: Fri, 05 Feb 2016 19:36:30 GMT
User-Agent:Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_3) AppleWebKit/537.36 (KHTML, like
Gecko)
Referer: https://www.udacity.com/me
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8,fa;q=0.6,es;q=0.4 2.
```

- When you send this HTTP-Request to the Udacity-Server, Udacity-Server starts to process your request and as a result it returns a proper response; this response can be a file or any type of objects. If you look at the HTTP-Request, you can figure out this HTTP-Request is asking for logomark.svg file.

- Now for simplicity Imagine I have 2 serves. When I receive an HTTP-Request (in server side) I want to split my HTTP-Requests between these two servers based on the HTTP-Response size (at this example response is a file logomark.svg). This figure illustrates the problem.



- In Predictor we have to make an important decision (predict), at this level we have to predict how big is the HTTP-Response size ($_{\text{the response is a file}}$) for this HTTP-Request. If the response size is less than 1MB send the HTTP-Request to Server A, if it is more than 1MB send it to Server B to get processed.
- Most probably you want to ask why prediction? When you send an HTTP-Request you don't mention about response size, it means there is no field in HTTP-Request header to mention about the response size, and when there is no information or rules to calculate the response size, I have to predict it.
- When we realize how big is the response size? After we process the HTTP-Request in the server.

Hopefully you have a good understanding of the problem and you know why I want to predict the response size in the Predictor. Because I want to split my HTTP-Requests to two types, HTTP-Request with response size less than 1MB and HTTP-Request with response size more than 1MB.

Now in the real problem I have more than two classes (1MB and more than 1MB). I will explain how many class do I need, future in this report.

What metrics I want to use to evaluate the Predictor?

I decided to solve this problem by designing a multi-class classifier. One of the best measurements for classification is F1, which is a combination of Precision and Recall. In this classifier I have 10 classes (class 0 to 9). I want to calculate Precision, Recall and F1 for each of my classes separately.

Recall shows us what percentage of HTTP-Requests which we predicted class-0 for actually belong to class-0 and what percentage of them belong to other classes and we incorrectly predicted class-0. For example, it says 95% of HTTP-Requests that we predicted class-0 for actually belong to class-0 but 5% of them actually belong to other classes (1 to 9).

Precision is a little different. Precision tells us what percentage of HTTP-Requests that **belong to class-0** incorrectly predicted class 1 to 9 and what percentage of HTTP-Requests correctly predicted class-0. For example 90% of HTTP-Requests with actual label 0 predicted class-0, but 10% of HTTP-Requests that are actually class-0 incorrectly predicted other classes.

True Positive: Percentage of HTTP-Requests that actually belong to class-0 and we predicted class-0.
False Positive: Percentage of HTTP-Requests that actually belong to other classes but we predicted class-0.
False Negative: Percentage of HTTP-Requests that actually belong to class-0 but we predicted other classes.

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive + False Negative}} \qquad \text{Precision} = \frac{\text{True Positive}}{\text{True Positive+ False Positive}}$$

F1 = ((Precision* Recall) / (Precision+ Recall)) * 2

How does it affect people? How does it affect you?

One of the biggest challenges for smart load balancing in Content Delivery Network (CDN) system is about response size. The goal of a CDN is to serve content to end-users with high availability and high performance. CDNs serve a large fraction of the Internet content today, including web objects (text, graphics and scripts), Downloadable objects (media files, software, documents), applications (e-commerce, portals), live streaming

media, n-demand streaming media, and social networks. (For more information about load balancing and CDN check Wikipedia)

All of the content on the CDN are static, so there is no heavy process (CPU intensive) in a CDN system.

Now let's make our example close to the real problem. We had two servers (A and B); each server can process a file with response size less than 1MB in 1 second and HTTP-Request with response size more than 1MB in 5 seconds. It means one server can process 10 HTTP-Requests with the response size less than 1 MB in 10 seconds but in the same time (2 seconds) they only can process 2 HTTP-Requests with the response size more than 1 MB.

Now if I receive a list of HTTP-Requests, we can split them between two servers and process all HTTP-Requests with small response faster than HTTP-Request with large response size, Exactly like Express line in Grocery Store. You are serving all HTTP-Request with small response size together and we don't let any HTTP-Request with large response size come to this server, because it takes more process-time and makes or process speed slow.

Analyze The Problem

What is the size of your dataset?

The size of my Data-Set is so big; I don't have any estimation about the size of this Data-Set because this it coming from a CDN logs that serves about 8% of the Internet traffic. This Data-Set is private and I cannot publish it.

I used hash function to hash important part of my Data-Set in order to use in this project. I added some of the features from my Original Data-set at the end of this paragraph. I chose 500,000 unique HTTP-Request and HTTP-Response from this Data-Set for the report and this project.

Time Stamp, IP Address, Customer ID, Accept-Encoding, URL, Host ID, Server name, http request line, client TCP info, Time of HTTP-Request in Sec, Software version, Server port number, Authenticated username….

How many features are present?

Based on my research we need all of the features/metric that are fully or partially related to HTTP-Response size, such as file extensions. Why the file extensions?

For example most of JavaScript (*.js) file are less than 20 KB or most of images (*.jpg) are bigger than 1 MB and less than 5 MB. "`Accept`" is another feature/metric which is highly related to response size, this features is telling us what kind of content HTTP-Request is looking for, this feature along with extension can bring a lot of information to the mode.

Another features/metric that is kind of related to HTTP-Response size is "Agent" or HTTP-Request sender, this feature tells us where this request is coming from, most of request from mobile devises have smaller size than desktop devises. Combination of these features/metrics plus some technique can tell us about response size.

If we add the HOST name to these sets (Extension, Agent and ACCEPT) of features we can say we have almost enough information to predict how big is a HTTP-Response, for example most of HTTP-Requests with HOST[www.udacity.com], Extension[jpg] and ACCEPT[Image] are more than 750 kilobytes and less than 2 MB.

As I mentioned before all I have as a Data-Set is a list off HTTP-Request and HTTP-Response header fields, to check list of HTTP headers files please check this link.
https://en.wikipedia.org/wiki/List_of_HTTP_header_fields.

You have to consider most of these HTTP header fields are optional and only few of them always available on each HTTP-Request, and second my Data-Set does not contain all off the HTTP headers because we did not log all of them (saving more space, because it was not important for company at that time).

I am using only four of them. This table shows my sample Data-Set. This Data-Set contains some features of Original Data-Set.

Features				Target
HOST	File Extension	Accept	Is-Mobile	Response Size
www.udacity.com	jpg	image/webp,image/*,*/*;q=0.8	0	19
www.netflix.com	rt	movie/webp	1	23

My original Data-Set (before data cleaning) is a list of HTTP-Request and HTTP-Response headers, but the logging system in order to use less space to save these HTT-Request and Response, did not save every parts of HTTP-Request and HTTP-Response header, therefor in my Original Data-Set I don't have access to all headers information.

Do I need to clean or convert my data when I want to predict new (Unseen) HTTP-Request in real Environment (Production Server)? No because at that time, I will get a copy of original HTTP-Request which contains every parts of a real HTTP-Request.

I choose these four features because they are always available in my Data-Set and they are not optional (it means I always have these features and none of them miss).

Second, based on my calculation with "information gain" and "entropy" formula, these four features are bringing the most information about the target into the model.

For other features "Information gain" and "entropy" aren't zero, but they are making the model bigger (size) and slower to make a prediction, but just adding some information plus a lot of noise, and they are not always available in the HTTP-Request.

Which features seem most promising?

Host: Extracted from the URL (**www.udacity.com**/Machine_learning/images/demo.jpg

File-Extension: Extracted from the URL (www.udacity.com/Machine_learning/images/demo.**jpg**)

Accept: Extracted from the http-Request header (`Accept: text/html`)

Is-Mobile: I extracted this feature and made a lot of changes on the USER-AGENT field in the http-request header to make this feature as a Boolean value, this feature only says whether this request is coming from a mobile devices (i.e. smartphone, tablets and etc.) or not.

Are there any categorical variables that may need to be converted?

All of my features are categorical, but there is no end for these categories. They are like words in a text. I tried to use text-processing (NLP) techniques to categorize them into a certain number of categories but I could not , because there is a big difference between NLP and this problem.

For more information:
In most of NLP tools such as, Dependency Grammar, Semantic Role Labeling (SRL), Information Extraction (IE) or Name Entity Recognition (NRE), they don't work with words directly, most of them use POS tagger to label words and then process those words based on their Part of Speech Tag.

In this system there is now way to combine or categorize some Hosts, File-Extension and Accept to one or many meaning full category.

That's why I cannot work with some techniques like dummy variable.

Which ones would you expect to perform well?

Based on the Information Gain and entropy for all my four features. Extension and Host perform well.

How easy is it to convert the available data into a suitable form?

I spent most of my time to crate my data set and doing some data cleaning. I spent a lot of time to remove or replace null value with mean or the biggest category. Unfortunately my Data-Set is not so clean and categorize, that's why I spent a lot of time on data cleaning.

My original Data-Set (before data cleaning) is a list of HTTP-Request and HTTP-Response headers, but the logging method/system in order to use less space, did not save every parts of HTTP-Request and HTTP-Response headers and they removed some part of this data.

Do I need to clean or convert my data when I want to predict new (Unseen) HTTP-Request in real Environment (Production Server)? No because at that time, I will get a copy of original HTTP-Request which contains every parts of a real HTTP-Request.

Example:

Earlier in (*Which features seem most promising? part*) I mentioned that I made a lot of changes to extract Is-Mobile. The original data for device was like

"Mozilla/5.0 (FVHG OS 7_1_2 like 6600) Epson.Printer/537.51.2 (KHTML, like Gecko) Version/7.0 /11D257 /9537.53"

"Mozilla/5.0 (Windows NT 6.3; WOW64; Trident/7.0; MDDCJS; rv:11.0) like Gecko"

"Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/45.0.2454.101 Safari/537.36"

"Mozilla/5.0 (iPad;CPU OS 7_1_2 like Mac OS X) AppleWebKit/537.51.2 (KHTML, like Gecko) Version/7.0 Mobile/11D257 Safari/9537.53"

I extract this information from User-Agent part that explain about HTTP-Request sender like browser and Operation system. I used regular Expression and a dictionary of "mobile operation systems" and "mobile device name" to extract this feature.

Another example is about response size from HTTP-Response headers. This value is integer and indicates the Size of HTTP-Response that is my Label for this calcification system, but in some cases there wasn't any real

value so I had to fix them or remove them as a outlier, also some of them were really big, which does not make any sense and some of them where negative that is totally wrong because HTTP-Response size can not be negative. I removed some of them as outlier and replaced some of the with median value.
I categorized my Data-Set based on the file extension, Host-Name and Accept type then I used Median of HTTP-Response size for each category to replace with missed data.

Describe a solution.

How accurate do your results need to be? How quickly does your algorithm need to produce an answer?

Ideally accuracy must be 100%, what does it mean 100% accuracy? It means we classify all Http-Request with response size less than 1MB as A and all Http-Request with response size more than 1MB as B.

Based on my pre-test on the data set we have to reach more than 85% accuracy for this system, other wise we cannot see a tangible improvement or real impact on our performance.

My result needs to be at least 85% accrue.

The speed is one of my biggest concerns on this system, because this application (Predictor) should be capable to predict more than 1,000,000 requests per second, and also my second problem and concern is about online learning, this application has to be online and learn from upcoming requests. It has to be able to keep the model up-to-date during the time.

Why we have to keep the model up-to-date during the time? Because some Http-Request response size (file size) are variable and they change during the time, it means if for the first time a Http-Request response size be less than 1MB there is no guaranty that second time should be less than 1MB, because file owner can change his file, and changing the file, change its size.

Why I think this problem is online? Because after we process an Http-Request in server A or B, in the way back in response header, we can read response size and we can check our prediction with actual value. For example if I predict it as A but when I process it I realize it is B, then I can update my model to not make same mistake again, next time predictor will predict same Http-Request as a B Request.

When you can see the actual value for your prediction, it calls online learning, because you able to compare your prediction with actual value and improve your system based on the feedback.

I will talk more about my implementation and solution under *what is my solution and which Machine-learning Algorithm I want to use? Section*.

What are the labels?

In our simple example we only had two classes (A and B) but in the real problem we have more than two classes. In this system I don't know how big is the biggest file but I know the smallest one can be 1 byte.

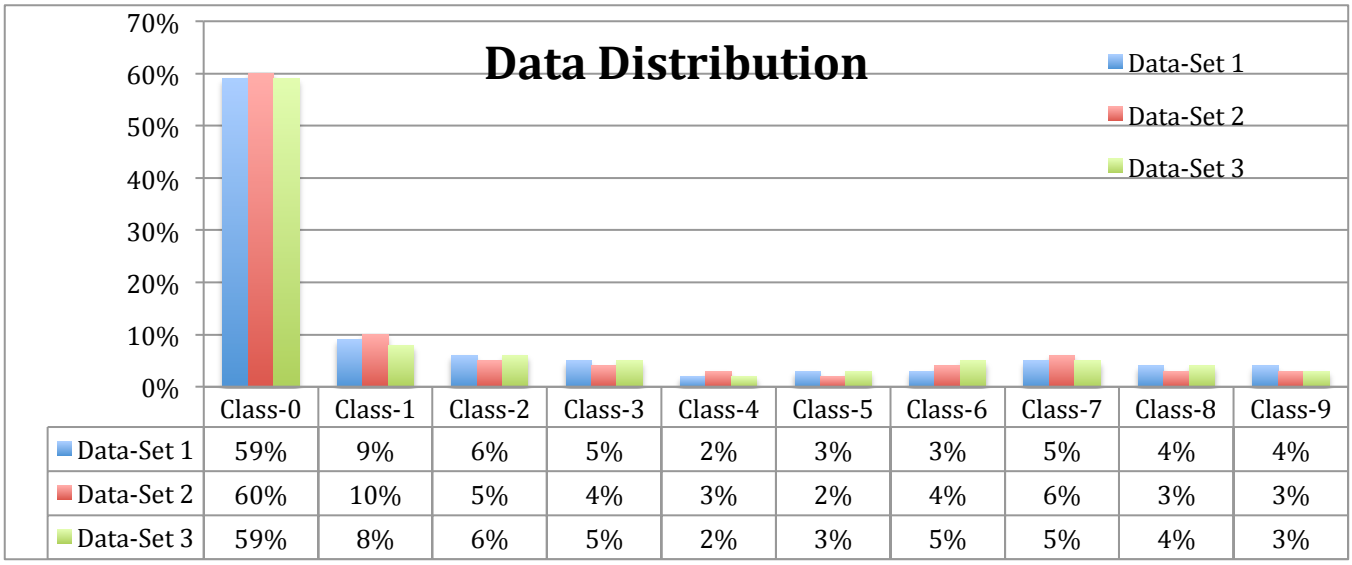
We can use regression for this problem because response size has continues value and there is no end, but I want to change my problem to a classification problem. My solution to change this problem from continues value to categorical value is applying ($Log_2^{(Size)}$) on the response size. So my target label become like this.

#	Response Size **From** (Byte)	Response Size **To** (Byte)	Label
1	0	524288	19
2	524289	1048576	20
3	1048577	2097152	21
4	2097153	4194304	22
5	4194305	8388608	23

#	Response Size **From** (Byte)	Response Size **To** (Byte)	Label
6	8388609	16777216	24
7	16777217	33554432	25
8	33554431	67108864	26
9	134217729	134217728	27
10	268435457	268435456 and bigger	28

I merged label 0 to 19 together and also all labels above 28.This table says if http request response size is (equal or grater than 0) and (equal or less than 524288), the best label for this Http-Request is 19. Why 19? Because $Log_2^{(524288)} = 19$.

Now we know this problem is a classifier, with 10 target labels (19 to 28) and each one represents a range of Http-Request response size.



	Class-0	Class-1	Class-2	Class-3	Class-4	Class-5	Class-6	Class-7	Class-8	Class-9
Data-Set 1	59%	9%	6%	5%	2%	3%	3%	5%	4%	4%
Data-Set 2	60%	10%	5%	4%	3%	2%	4%	6%	3%	3%
Data-Set 3	59%	8%	6%	5%	2%	3%	5%	5%	4%	3%

What pre-processing operations do you have to carry out on the features? (e.g. scaling, normalization, selection, transformation)

As I mentioned earlier I extracted all of my features from a bag of uncategorized dataset. For example there wasn't any **Is-mobile** feature and I had to process "Agent" property in the Http-Request fields to realize where this request is coming from (i.e. OS, Browser, OS version and etc.)

Extracting the "Extension" is a difficult task, because most of the URLs don't contain extension, I had to use their "content-type" in response or ACCEPT in the HTTP-Request to realize what is the extension. Some of the URLs are using query string to mention about which file they are looking for, I had to process all query string to realize that, sometimes a query string is encrypted and there is no clear way to extract information.

Another thing I encountered was Encoding.

One another thing that I have done was changing my Target-label from a continues value to a categorical value, because I could not solve this problem with regression algorithm, so I changed my solution from regression to classifier, but the next question was how should be the categories and how big each category should be. Eventually I clustered my target data with "Log_2".

What is my solution and which Machine-learning Algorithm I want to use?

Based on what I have explained until now, my algorithm has to be so **fast**, it has to be able to **up-to-date its self** with new data and simultaneously **forget its past information** and using small amount of **memory**.

I started to work with Perceptron model to take full advantage of its online capability, and simultaneously I was thinking about other Neural Network, Deep learning and reinforcement learning techniques, but after spending a lot of time to implement those Ideas, I realized Neural Network is really slow for my system despite getting really good result. Perceptron guarantees to converge, if data is linearly separable. But my data is not linearly separable.

I continued my research with Decision Tree (ID3). It was really fast (which was my most important criteria) and it would have let me update the Model (Tree Structure) whiteout creating a new model from scratch and it was compatible with text data and I would not have to change text data to numerical equivalent.

My model has to be able to **up-to-date its self** with new data point (unseen data point). This metric is highly depending on how I implement my Decision Tree (ID3).

I am keeping my Tree up-to-date by adding new data point (unseen data point) to the Model (TREE). If I get a data point which is not exists in the tree, I will predict it with the biggest class (or default class which is 19). After processing the same request we have access to the real class (actual value) and we can compare the prediction with actual value. If prediction was correct we don't need to do any thing even if data point is new. If prediction was Incorrect it means the model is not working well and we need add this data point to the model (Only when data point is new)

Similar Idea to keep our model Up-to-date with changes and new data point is to wait to see one request N times and after N times we will decide what should be the label for this type of new data-points and then we will add it to the tree.

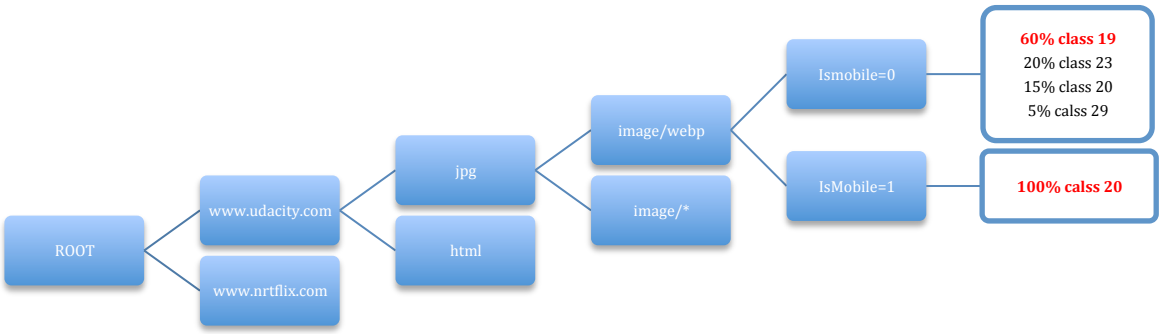
I decided to use Decision Tree (ID3) because it supports most of my criteria.

I am using Decision Tree but I am aware of its weakness, for example over-fitting is one of them, but despite this problem, it is still the best algorithm for Reponses size prediction.

This is my core implementation for creating and updating tree. I am using nested list(dictionary) and recursive technique.

```
def UpdateTree(self,node,features):
      if features != None:
         if features[0] in node:
             if len(features)>1:
                self.UpdateTree(node[features[0]],features[1:])
             else:
                   node[features[0]]+=1
         else:
            if len(features)==1:
                if features[0] in node:
                    node[features[0]]+=1
                else:
                    node[features[0]]=1
             else:
                 node[features[0]] = defaultdict(dict)
                self.UpdateTree(node[features[0]],features[1:])
```

- This chart is showing our final model. The final node is showing the percentage of each class at that final node. Some final nodes don't have uncertainty and we only have one label like (**100% class 20**) some have uncertainty and we chose the highest probability or the biggest class (at this example **60% Class 19**)



You will see some output like this (below image) if you run my code (FS.py). I calculated F1 separately for each class. And overall Precision was around 89% percent. But if I run this application with huge amount of data I will get 93% F1 Score. The only reason for 4% increase is over-fitting problem because the Model is not general enough to cover all test data points ant it is suffering from a lack of data.

I can keep this score during running this application, because I check my prediction with actual values after prediction process. (Updating the Model)

```
Train Size:  160000
Test Size :  40000
Start Building Tree at:  2016-02-25 01:56:31.959844
Number of Fist level Nodes:  2184
Tree Size 196888
Start Prediction Tree at:  2016-02-25 01:56:40.750718
------------------------------Report--------------------------------
--------------------------------------------------------------------


H           0|     1|     2|     3|     4|     5|     6|     7|     8|     9|     Recall
0       33843     56    109     13      0      3      0      1      0      0      - 99.47
1         596     93    274     18      1      1      0      0      0      3      - 9.43
2         256     52   3072     79      4      6      2      0      0      1      - 88.48
3          91     27    283    203      8      7     10      2      0      1      - 32.12
4          44      1    120     53     33     20     18      9      0      7      - 10.82
5          13      0      4      3     20     38     42      5      6      3      - 28.36
6          12      0      3      1      9     11     68      7     31      4      - 46.58
7           7      1      0      2      2      8     45      7     40      3      - 6.09
8           9      0      0      1      1      2     18      6     61      5      - 59.22
9           7      0      0      0      2      3      8      4     27     31      - 37.8
Precision: 97.03|  40.43|  79.48|  54.42|  41.25|  38.38|  32.23|  17.07|  36.97|  53.45|


--------------------------------------------------------------------
```

This image shows, precision and Recall for each label (19-28)

Conclusion:

At this report I talked about the problem and my best solution, but there was some other idea that I like to talk about.

I applied a range of Decision-Tree algorithm such as C4.5, C5 and CART on my Data-Set and in some cases I got a better result than my final result, but in general I had more accurate and faster prediction with ID3. Also ID3 was so easy to update (ADD/Remove).

One interesting part and most probably difficult part of this project was about converting the labels from continuous value to fixed number groups. There were three reasonable ideas around this problem. Firs was about creating some groups with equal size, for example each group 5 MB, Second solution was using LOG2(Response Size) which I am using this solution and third solution was about creating groups based on

the best result, it means we move or change our portions(Group) size until we get the best result. Each solution has some advantages and some disadvantages.

First: Creating group with fixed size gives us consistency and we know each group has a fixed size but the problem is some groups are null (they are not include any data points) and most of data was only in one or two groups, this solution couldn't split our data-points equally between groups.

The second solution doesn't have the unbalanced distribution problem. I have so many small numbers and just few large numbers (Response Size), Log2 helps us to split our data points equally between 10 groups (partitions), first groups have smaller size and last groups have bigger size and this trick can split out data-points equally between groups. The problem with this solution is consistency, we don't have consistency any more an each group had different size.

Third solution was about creating group based on the best result. This solution suffers from over-fitting, because we created our groups based on the training data and we cannot generalize it for testing data.

One more topic that I like to talk about is F1 score and how much we can improve this score later. One of the most important problems that I think have bad impact on my Model is suffering from a lack of features. Technically I cannot separate my data points better (purer) even if I use more complex mode. So increasing the F1 score is highly related to finding and adding more new features.