ESIEE PARIS

M.Sc. MOTIS

PROJECT

Telstra Network Disruptions Competition

PREPARED BY: TOMÁS QUINTEROS
PREPARED FOR: POUYA YOUSEFI

06/06/17

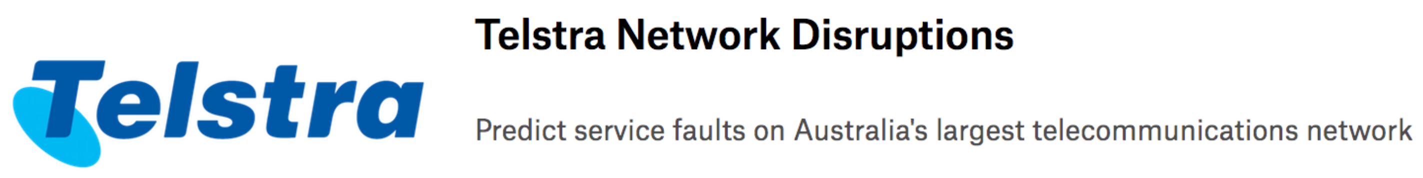
TABLE OF CONTENTS

1. INTRODUCTION

In their first recruiting competition that ended on 29th February 2016, the company Telstra, Australia's largest telecommunications network, is challenging Kagglers to predict the severity of service disruptions on their network. Telstra is on a journey to enhance the customer experience - ensuring everyone in the company is putting customers first. In terms of its expansive network, this means continuously advancing how it predicts the scope and timing of service disruptions.

This report is based on the Telstra challenge development for the BI: Data Analytics lecture of MoTIS 8 with Pouya Yousefi. It was developed on Dataiku DSS which is a collaborative data science software platform for teams of data scientists, data analysts, and engineers. Is very intuitive, easy-to-use, friendly and allows to explore, prototype, build, and deliver your own data products efficiently.

2. UNDERSTANDING THE PROBLEM



Telstra Network Disruptions

Predict service faults on Australia's largest telecommunications network

To begin with, Telstra is the largest telecommunication network in Australia providing services such as mobile phones, broadband, landline as well as digital TV. The problem is to find out whether if a disruption occurred is a momentary glitch or total interruption of connectivity. In other words, the task was to predict the severity of service disruptions on Telstra's network.

The accurate predictions of service disruptions will help Telstra serve customers better and enhance customer experience by providing better connectivity. Telstra wants to see how we would help it drive customer advocacy by developing a more advanced predictive model for service disruptions by using a dataset of features from their service logs.

3. DATA DESCRIPTIONS

The goal of the problem is to predict Telstra network's **fault severity at a time at a particular location** based on the log data available. Each row in the main dataset (train.csv, test.csv) represents a **location** and a **time point**. They are identified by the "id" column, which is the key "id" used in other data files.

Fault severity ("fault_severity" in train.csv) is a measurement of actual reported faults from users of the network, is the target variable of the problem and it has 3 categories:

0: meaning no fault,
1: meaning only a few,
2: meaning many.

This means it was a multi-classification problem (3-class).

Different types of features are extracted from log files and other sources:

event_type.csv - event type related to the main dataset,
log_feature.csv - features extracted from log files,
resource_type.csv - type of resource related to the main dataset,
severity_type.csv - severity type of a warning message coming from the log.

Note: “severity_type” is a feature extracted from the log files (in severity_type.csv). Often this is a severity type of a warning message coming from the log. "severity_type" is categorical. It does not have an ordering.

Other datasets:

train.csv - the training set for fault severity,
test.csv - the test set for fault severity,
sample_submission.csv - a sample submission file in the correct format.

4. APPROACH

4.1. Train and Test

First of all, both the train and test files were prepared by adding a column "fault_severity" into the test schema and erasing the word "location" in the location column to clean and enrich both dataset. The type of certain columns was changed e.g., ID string to integer, this required updating recipes and datasets downstream. At the same time each dataset was copied to a MySQL database connection and then stacked concatenating both prepared datasets for analysis.

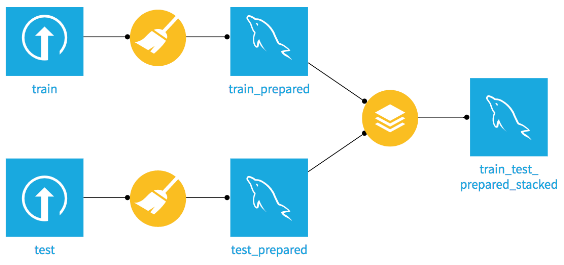
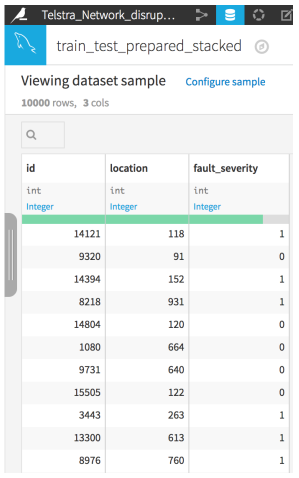


Image 1. Train and test files prepared and stacked.



train_test_prepared_stacked

Viewing dataset sample Configure sample

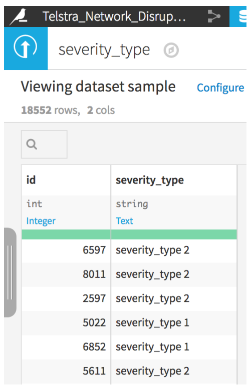
10000 rows, 3 cols

id	location	fault_severity
int Integer	int Integer	int Integer
14121	118	1
9320	91	0
14394	152	1
8218	931	1
14804	120	0
1080	664	0
9731	640	0
15505	122	0
3443	263	1
13300	613	1
8976	760	1

Image 2. Train and test data.

4.2. Severity_type

Afterwards, the severity_type dataset was analyzed. It contains the same number of rows as train and test files combined, and for each id there is one value for severity_type. So, first the preparation was done (same procedure as with train and test datasets) and then the merge with train and test stacked. A left join on ID, keeping all rows of the left dataset (id and severity_type) and adding information from the right dataset (location and fault_severity).

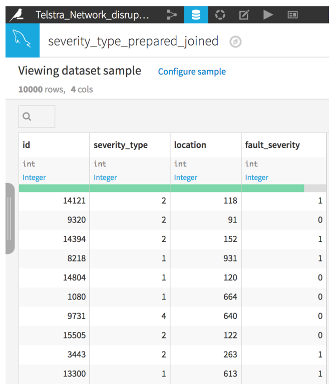


severity_type

Viewing dataset sample Configure

18552 rows, 2 cols

id	severity_type
int Integer	string Text
6597	severity_type 2
8011	severity_type 2
2597	severity_type 2
5022	severity_type 1
6852	severity_type 1
5611	severity_type 2



severity_type_prepared_joined

Viewing dataset sample Configure sample

10000 rows, 4 cols

id	severity_type	location	fault_severity
int Integer	int Integer	int Integer	int Integer
14121	2	118	1
9320	2	91	0
14394	2	152	1
8218	1	931	1
14804	1	120	0
1080	1	664	0
9731	4	640	0
15505	2	122	0
3443	2	263	1
13300	1	613	1

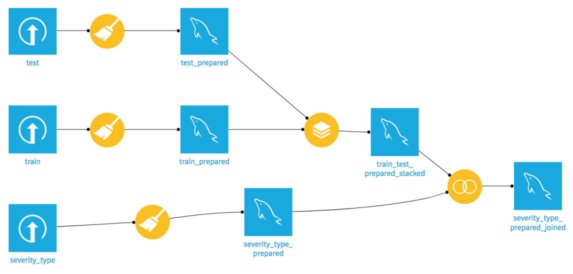


Image 3. Severity_type data.

Image 4. Severity_type_joined data.

Image 5. Train, test and severity_type joined.

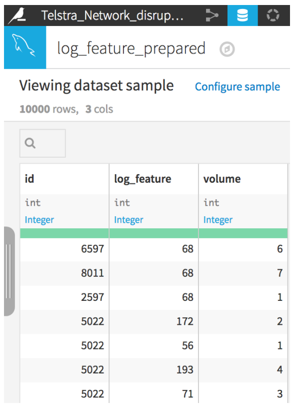
4.3. Log_feature

The next step was to analyze and prepare log_feature. This dataset includes for each log_feature a volume column which specifies a quantity associated with each variable. A particular log can contain many different types of log features. And each feature might occur multiple times in the log. For example: ID=3952, log_feature='feature 82', volume='4'.

The interpretation was: For the location & date associated with id 3952, feature 82 appeared in the log 4 times.

The next SQL recipe was created for count and concatenate log_features and volumes of each ID:

```
SELECT
    `id` AS `id`,
    COUNT(`log_feature`) AS `log_feature_count`,
    GROUP_CONCAT(log_feature,' ') AS `log_features`,
    COUNT(`volume`) AS `volume_count`,
    GROUP_CONCAT(volume,' ') AS `volumes`
 FROM `TELSTRA_NETWORK_DISRUPTIONS_1_log_feature_prepared`
  GROUP BY `id`
```

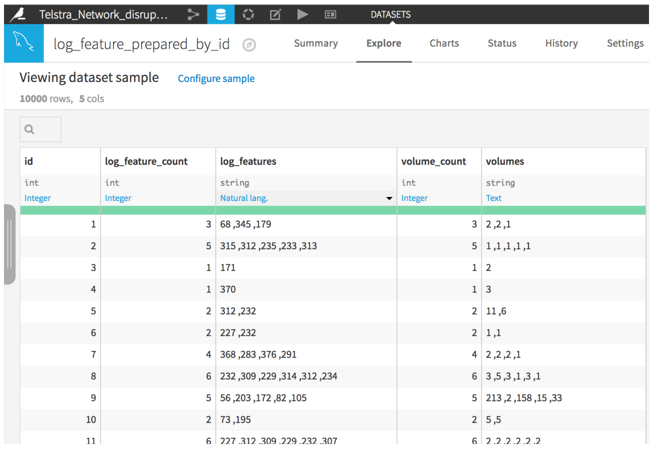


Viewing dataset sample Configure sample

10000 rows, 3 cols

id	log_feature	volume
int	int	int
Integer	Integer	Integer
6597	68	6
8011	68	7
2597	68	1
5022	172	2
5022	56	1
5022	193	4
5022	71	3

Image 6. Log_feature data



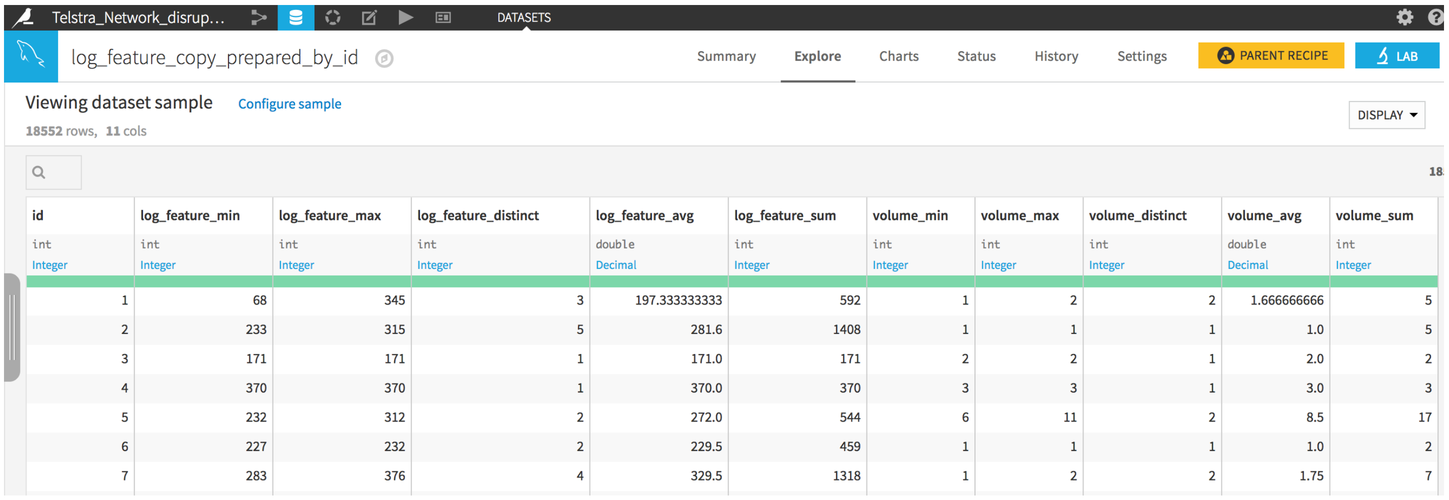
Viewing dataset sample Configure sample

10000 rows, 5 cols

id	log_feature_count	log_features	volume_count	volumes
int	int	string	int	string
Integer	Integer	Natural lang.	Integer	Text
1	3	68 ,345 ,179	3	2 ,2 ,1
2	5	315 ,312 ,235 ,233 ,313	5	1 ,1 ,1 ,1 ,1
3	1	171	1	2
4	1	370	1	3
5	2	312 ,232	2	11 ,6
6	2	227 ,232	2	1 ,1
7	4	368 ,283 ,376 ,291	4	2 ,2 ,2 ,1
8	6	232 ,309 ,229 ,314 ,312 ,234	6	3 ,5 ,3 ,1 ,3 ,1
9	5	56 ,203 ,172 ,82 ,105	5	213 ,2 ,158 ,15 ,33
10	2	73 ,195	2	5 ,5
11	6	227 ,312 ,309 ,229 ,232 ,307	6	2 ,2 ,2 ,2 ,2 ,2

Image 7. Log_feature data grouped by id

In order not to mixed the analysis, the log_feature dataset was duplicated and prepared it by setting the log_feature and volume fields from string to integer to compute various aggregates of feature and volume logarithm by id (distinct, min, max, avg, sum). Count was not included due to its already in the first log_feature dataset.



Viewing dataset sample Configure sample

18552 rows, 11 cols

id	log_feature_min	log_feature_max	log_feature_distinct	log_feature_avg	log_feature_sum	volume_min	volume_max	volume_distinct	volume_avg	volume_sum
int	int	int	int	double	int	int	int	int	double	int
Integer	Integer	Integer	Integer	Decimal	Integer	Integer	Integer	Integer	Decimal	Integer
1	68	345	3	197.333333333	592	1	2	2	1.666666666	5
2	233	315	5	281.6	1408	1	1	1	1.0	5
3	171	171	1	171.0	171	2	2	1	2.0	2
4	370	370	1	370.0	370	3	3	1	3.0	3
5	232	312	2	272.0	544	6	11	2	8.5	17
6	227	232	2	229.5	459	1	1	1	1.0	2
7	283	376	4	329.5	1318	1	2	2	1.75	7

Image 8. Log_feature_1 data grouped by id.

4.4. Resource_type and event_type

Like log_feature, this datasets contain more than one value (resource_type and event_type) for each ID. So the same procedure will be applied for both. First the classic preparation and connection to the database, then an SQL recipe to count and concatenate the values for each ID and finally the dataset duplication for changing the type, group and compute new values by id.

From severity_type_prepared_joined (joined with train and test) a left join with all the new datasets and the flow looks like this:

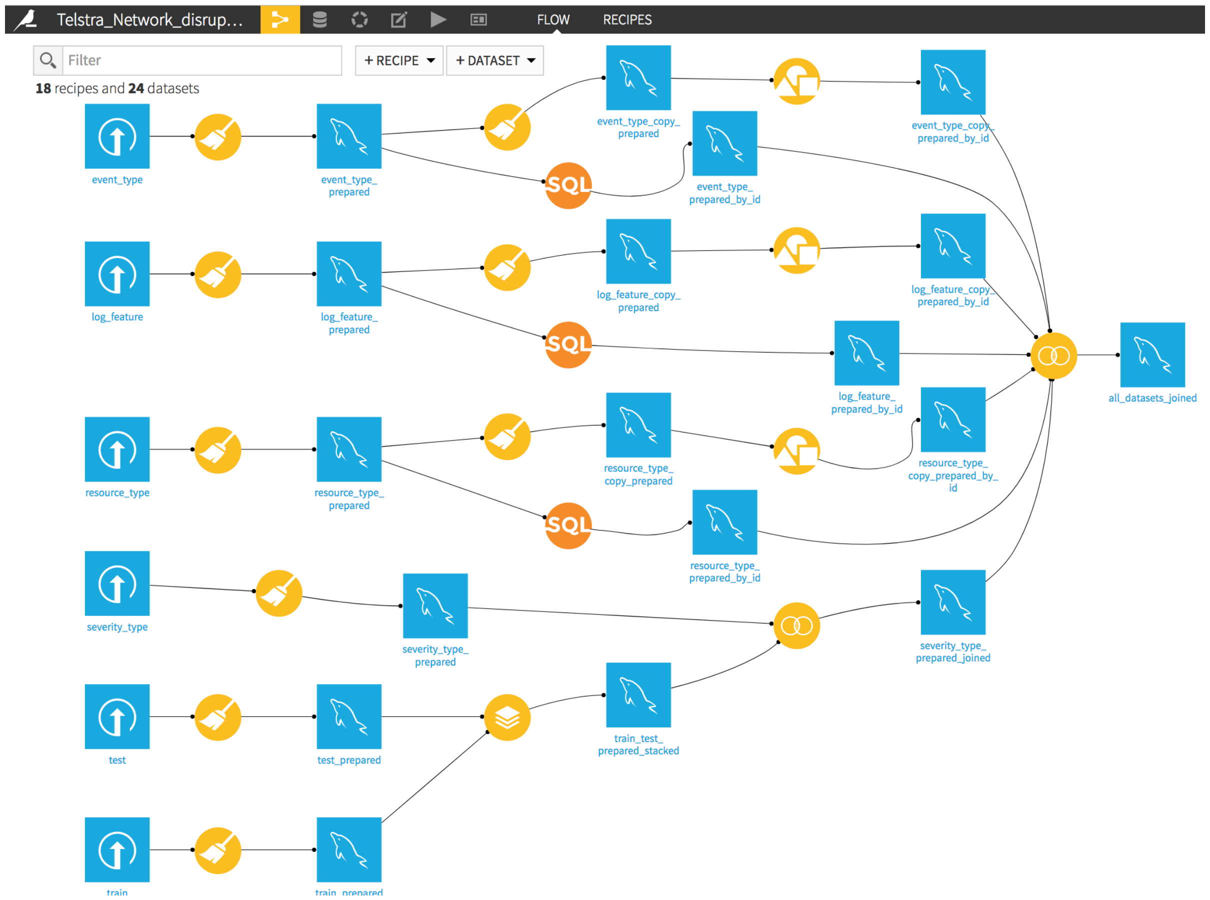
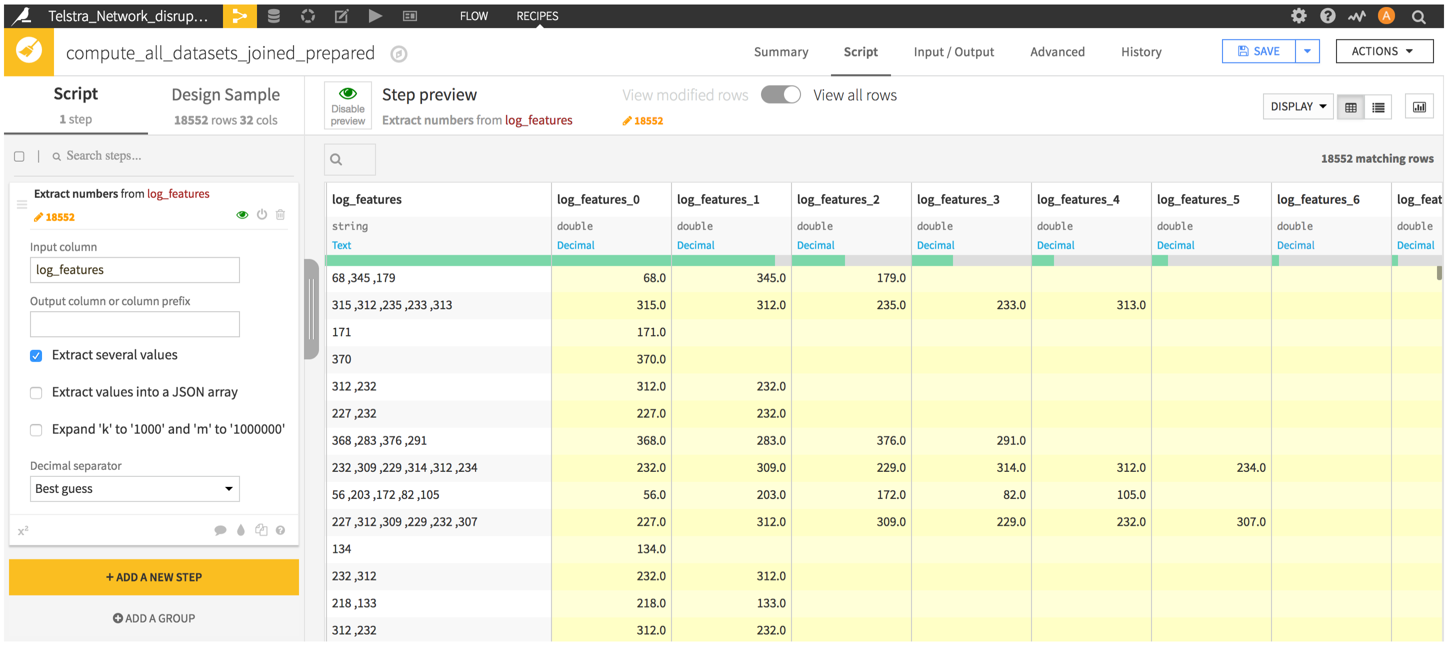

Telstra_Network_disrup...
FLOW
RECIPES
Filter
+ RECIPE
+ DATASET
18 recipes and 24 datasets
event_type
event_type_prepared
event_type_copy_prepared
event_type_prepared_by_id
event_type_copy_prepared_by_id
log_feature
log_feature_prepared
log_feature_copy_prepared
log_feature_prepared_by_id
log_feature_copy_prepared_by_id
resource_type
resource_type_prepared
resource_type_copy_prepared
resource_type_prepared_by_id
resource_type_copy_prepared_by_id
severity_type
severity_type_prepared
severity_type_prepared_joined
all_datasets_joined
test
test_prepared
train
train_prepared
train_test_prepared_stacked
SQL

Image 9. All_datasets_joined.

At this point we are able to split the joined dataset into train and test according to the fault_severity column, and then prepare the predictive model to obtain the test dataset scored. However, we can still improve the dataset with a couple of more features.

5. IMPROVEMENTS AND PREDICTIVE MODELS

In the first place we will use the string concatenated columns of log_feature, resource_type and event_type created: "log_features", "volumes", "resource_types" and "event_types". For each column we will use extract numbers processor that finds and extracts numerical values from a text column. In this way we are adding the exact value of an event, resource, volume or log feature into a column that can represent for example a date/time or a category. After this the string columns makes no sense so they can be erased.



Extract numbers from log_features
18552

Input column
log_features

Output column or column prefix

[✓] Extract several values

[ ] Extract values into a JSON array

[ ] Expand 'k' to '1000' and 'm' to '1000000'

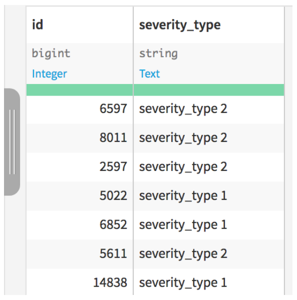
Decimal separator
Best guess

Extract numbers from log_features 18552

log_features	log_features_0	log_features_1	log_features_2	log_features_3	log_features_4	log_features_5	log_features_6	log_feat
string	double	double	double	double	double	double	double	double
Text	Decimal	Decimal	Decimal	Decimal	Decimal	Decimal	Decimal	Decimal
68 ,345 ,179	68.0	345.0	179.0					
315 ,312 ,235 ,233 ,313	315.0	312.0	235.0	233.0	313.0			
171	171.0							
370	370.0							
312 ,232	312.0	232.0						
227 ,232	227.0	232.0						
368 ,283 ,376 ,291	368.0	283.0	376.0	291.0				
232 ,309 ,229 ,314 ,312 ,234	232.0	309.0	229.0	314.0	312.0	234.0		
56 ,203 ,172 ,82 ,105	56.0	203.0	172.0	82.0	105.0			
227 ,312 ,309 ,229 ,232 ,307	227.0	312.0	309.0	229.0	232.0	307.0		
134	134.0							
232 ,312	232.0	312.0						
218 ,133	218.0	133.0						
312 ,232	312.0	232.0						

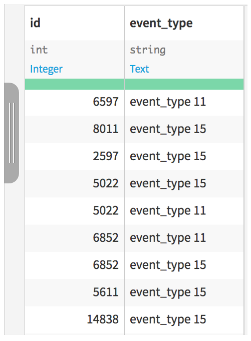
Image 10. Extracted numbers

Another nice feature is the location sort. In the first part of the analysis we can realize that each of the four datasets come in the same order of ID.



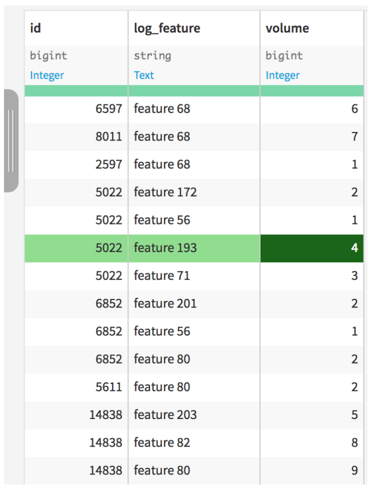
id	severity_type
bigint Integer	string Text
6597	severity_type 2
8011	severity_type 2
2597	severity_type 2
5022	severity_type 1
6852	severity_type 1
5611	severity_type 2
14838	severity_type 1

Image 11. Severity_type



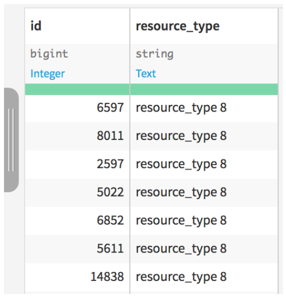
id	event_type
int Integer	string Text
6597	event_type 11
8011	event_type 15
2597	event_type 15
5022	event_type 15
5022	event_type 11
6852	event_type 11
6852	event_type 15
5611	event_type 15
14838	event_type 15

Image 12. Event_type



id	log_feature	volume
bigint Integer	string Text	bigint Integer
6597	feature 68	6
8011	feature 68	7
2597	feature 68	1
5022	feature 172	2
5022	feature 56	1
5022	feature 193	4
5022	feature 71	3
6852	feature 201	2
6852	feature 56	1
6852	feature 80	2
5611	feature 80	2
14838	feature 203	5
14838	feature 82	8
14838	feature 80	9

Image 12. Log_feature



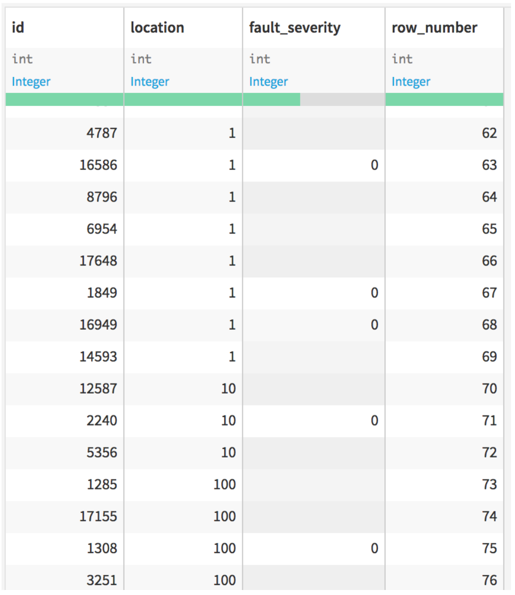
id	resource_type
bigint Integer	string Text
6597	resource_type 8
8011	resource_type 8
2597	resource_type 8
5022	resource_type 8
6852	resource_type 8
5611	resource_type 8
14838	resource_type 8

Image 13. Resource_type

Once you merge one of this files with the train and test files stacked, it turns out that the records are sorted by location, but not location as a number, but location as an ASCII string, this means 1000 comes before 999 and the order is like 1, 10, 100…
So, for keeping this particular order of IDs in one of the ordered datasets, a row_number was created in severity_type_prepared, with the next SQL recipe:

```
SELECT    *,
          @curRank := @curRank + 1 AS `row_number`
FROM      `TELSTRA_NETWORK_DISRUPTIONS_1_severity_type_prepared`, (SELECT @curRank := 0) r
```

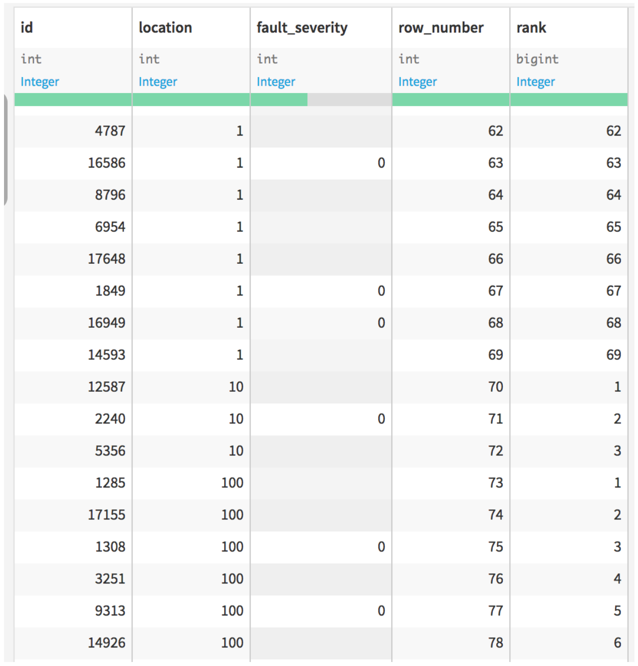
And then joined to train_test_prepared_stacked:



id	location	fault_severity	row_number
int Integer	int Integer	int Integer	int Integer
4787	1		62
16586	1	0	63
8796	1		64
6954	1		65
17648	1		66
1849	1	0	67
16949	1	0	68
14593	1		69
12587	10		70
2240	10	0	71
5356	10		72
1285	100		73
17155	100		74
1308	100	0	75
3251	100		76

Image 14. Prepared, stacked, ordered, joined

Since its order by location, it could be also order by time (maybe represented by the ID). In this case, a rank order was added to each location individually using the Window Visual Recipe of Dataiku. First, partitioning is made on location and order by row_number to maintain the original order of IDs.



id	location	fault_severity	row_number	rank
int	int	int	int	bigint
Integer	Integer	Integer	Integer	Integer
4787	1		62	62
16586	1	0	63	63
8796	1		64	64
6954	1		65	65
17648	1		66	66
1849	1	0	67	67
16949	1	0	68	68
14593	1		69	69
12587	10		70	1
2240	10	0	71	2
5356	10		72	3
1285	100		73	1
17155	100		74	2
1308	100	0	75	3
3251	100		76	4
9313	100	0	77	5
14926	100		78	6

Image 15. Prepared, stacked, ordered, ranked by location

Prepared and joined to the rest of the datasets, keeping the location order, the flow looks like this:

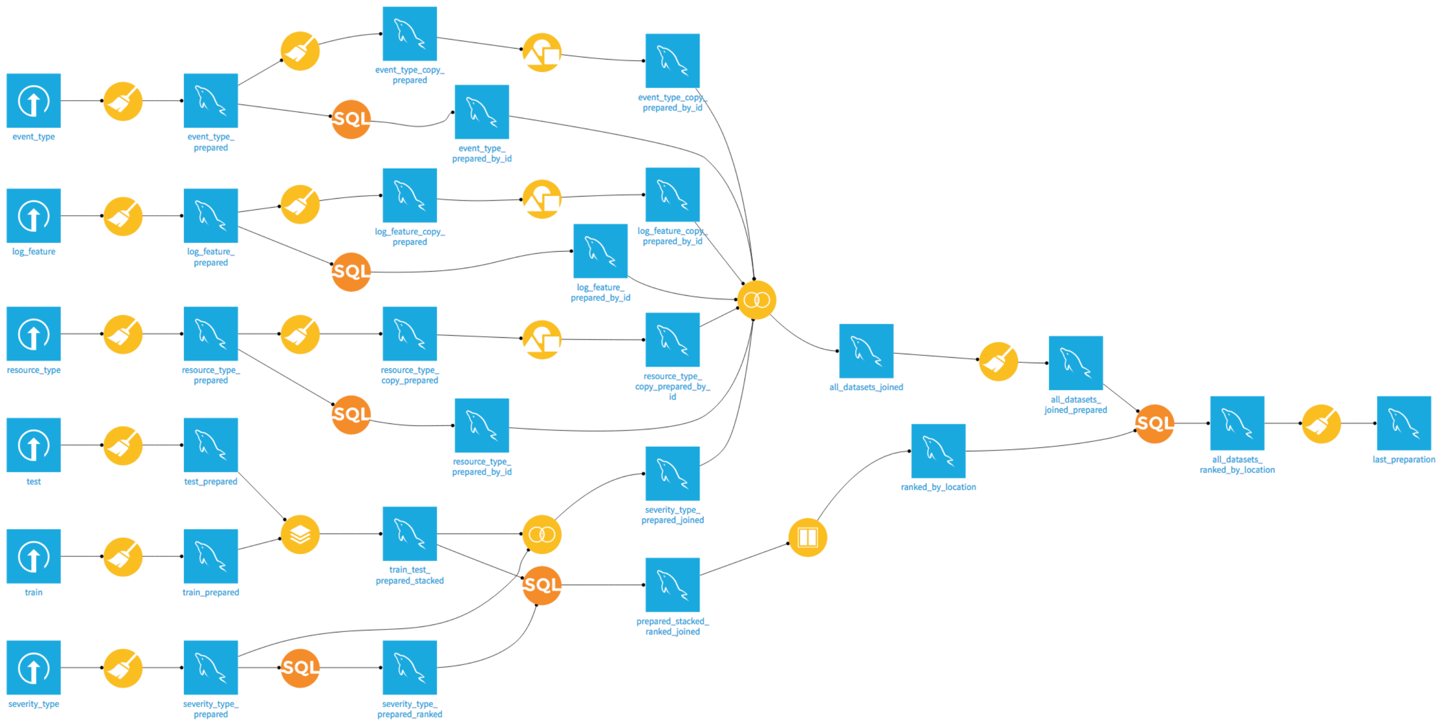
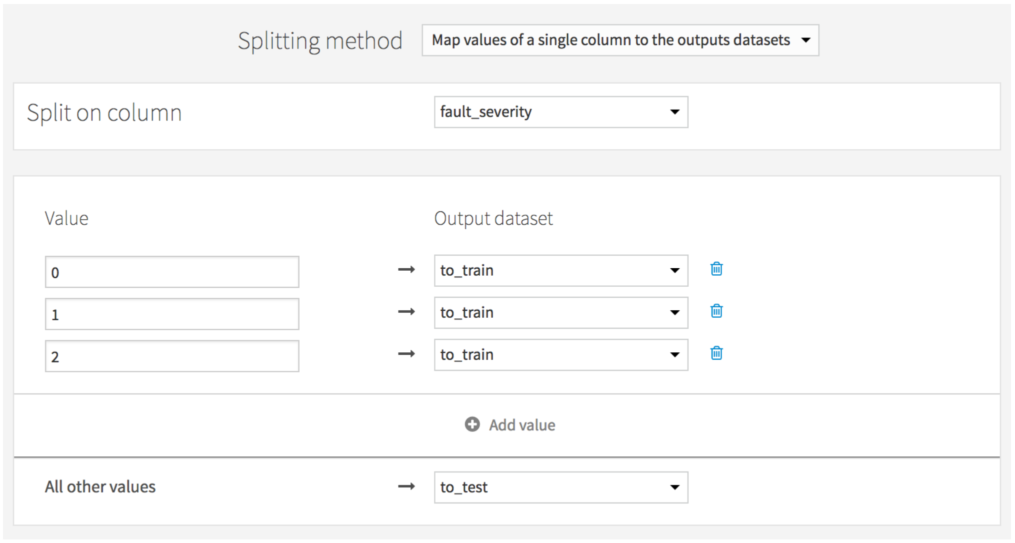

event_type
event_type_prepared
event_type_copy_prepared
event_type_copy_prepared_by_id
event_type_prepared_by_id
log_feature
log_feature_prepared
log_feature_copy_prepared
log_feature_copy_prepared_by_id
log_feature_prepared_by_id
resource_type
resource_type_prepared
resource_type_copy_prepared
resource_type_copy_prepared_by_id
resource_type_prepared_by_id
test
test_prepared
train
train_prepared
train_test_prepared_stacked
severity_type_prepared_joined
prepared_stacked_ranked_joined
severity_type
severity_type_prepared
severity_type_prepared_ranked
all_datasets_joined
all_datasets_joined_prepared
ranked_by_location
all_datasets_ranked_by_location
last_preparation
SQL

Image 16. Last preparation

5.1. Predictive Models

Now we are ready to split the data and train the predictive model. First, a split was made in the last prepared dataset with the next settings:



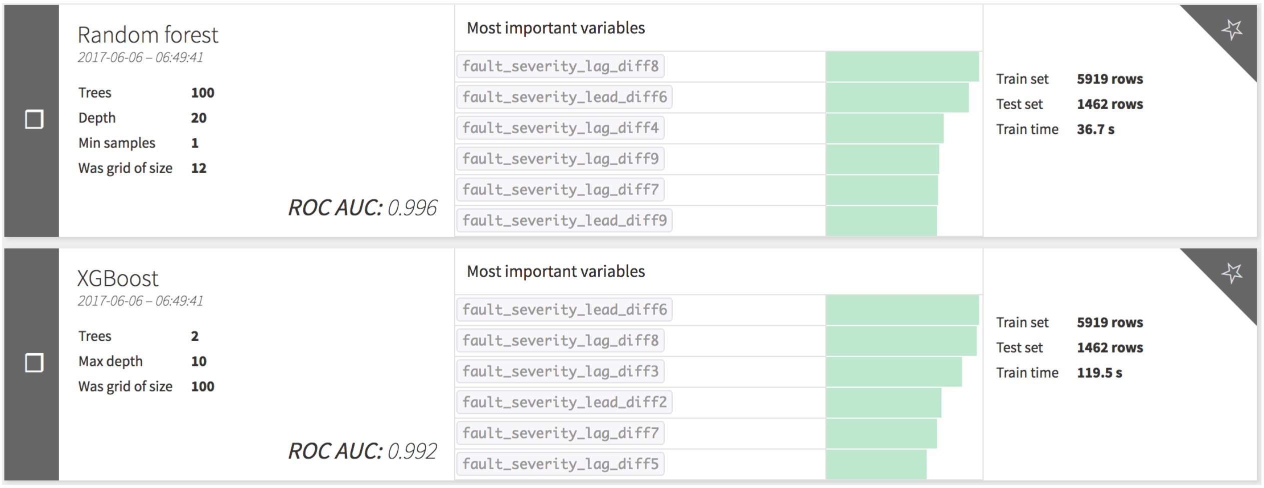
Splitting method: Map values of a single column to the outputs datasets

Split on column: fault_severity

Value	Output dataset
0	to_train
1	to_train
2	to_train
All other values	to_test

Image 17. Split settings

Next step is to create a prediction model in "to_train dataset" with the fault_severity variable to predict. A **Performance** model was chosen to train a highly cross-validated model to attempt to obtain the best possible predictions.



Random forest
2017-06-06 – 06:49:41

Trees	100
Depth	20
Min samples	1
Was grid of size	12

ROC AUC: 0.996

Most important variables

fault_severity_lag_diff8
fault_severity_lead_diff6
fault_severity_lag_diff4
fault_severity_lag_diff9
fault_severity_lag_diff7
fault_severity_lead_diff9

Train set	5919 rows
Test set	1462 rows
Train time	36.7 s

XGBoost
2017-06-06 – 06:49:41

Trees	2
Max depth	10
Was grid of size	100

ROC AUC: 0.992

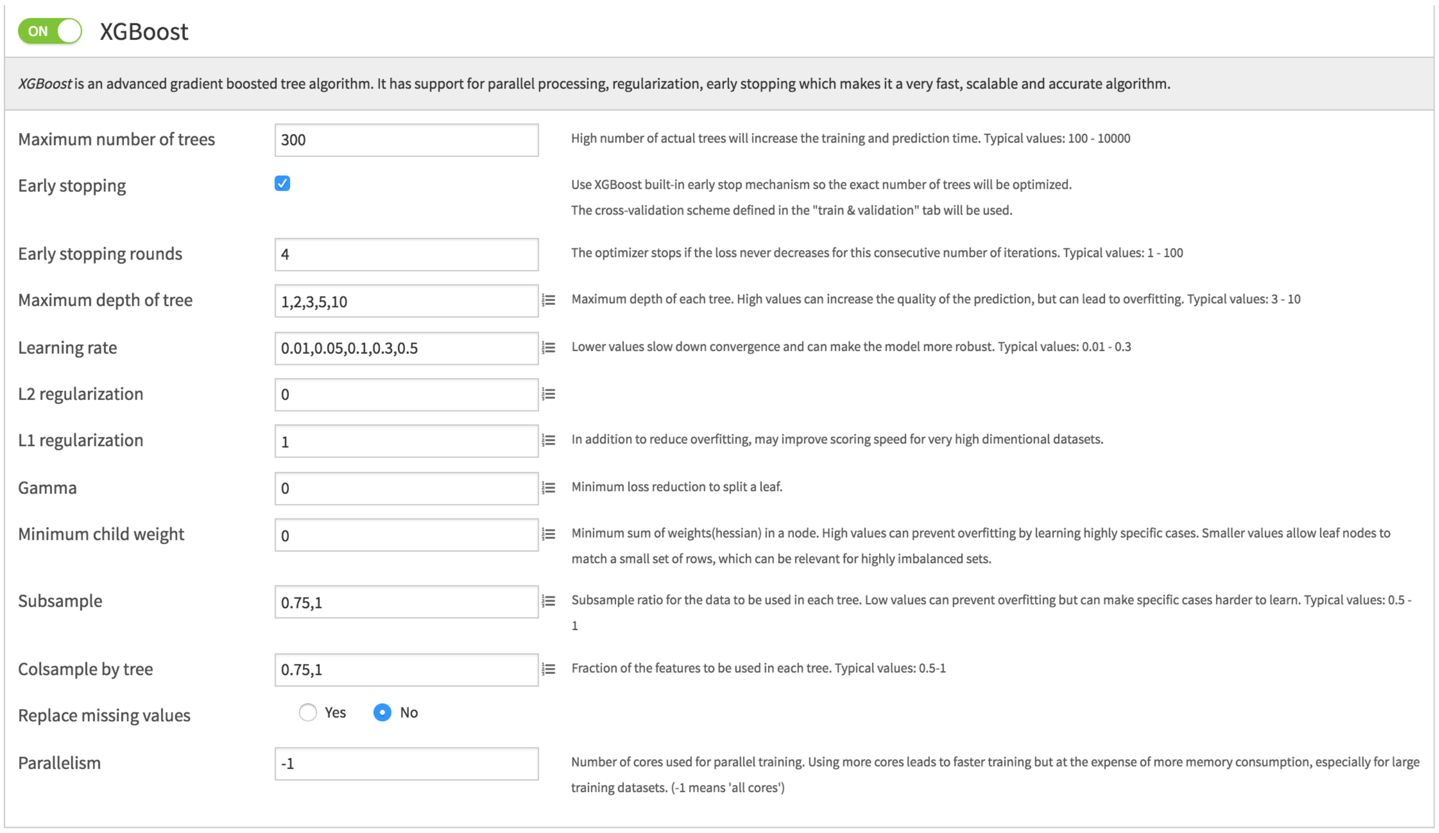
Most important variables

fault_severity_lead_diff6
fault_severity_lag_diff8
fault_severity_lag_diff3
fault_severity_lead_diff2
fault_severity_lag_diff7
fault_severity_lag_diff5

Train set	5919 rows
Test set	1462 rows
Train time	119.5 s

Image 18. Performance models

The XGBoost is an advanced gradient boosted tree algorithm. It has support for parallel processing, regularization, early stopping which makes it a very fast, scalable and accurate algorithm. The settings where used by default:


ON XGBoost

XGBoost is an advanced gradient boosted tree algorithm. It has support for parallel processing, regularization, early stopping which makes it a very fast, scalable and accurate algorithm.

Maximum number of trees	300	High number of actual trees will increase the training and prediction time. Typical values: 100 - 10000
Early stopping	✓	Use XGBoost built-in early stop mechanism so the exact number of trees will be optimized. The cross-validation scheme defined in the "train & validation" tab will be used.
Early stopping rounds	4	The optimizer stops if the loss never decreases for this consecutive number of iterations. Typical values: 1 - 100
Maximum depth of tree	1,2,3,5,10	Maximum depth of each tree. High values can increase the quality of the prediction, but can lead to overfitting. Typical values: 3 - 10
Learning rate	0.01,0.05,0.1,0.3,0.5	Lower values slow down convergence and can make the model more robust. Typical values: 0.01 - 0.3
L2 regularization	0	
L1 regularization	1	In addition to reduce overfitting, may improve scoring speed for very high dimentional datasets.
Gamma	0	Minimum loss reduction to split a leaf.
Minimum child weight	0	Minimum sum of weights(hessian) in a node. High values can prevent overfitting by learning highly specific cases. Smaller values allow leaf nodes to match a small set of rows, which can be relevant for highly imbalanced sets.
Subsample	0.75,1	Subsample ratio for the data to be used in each tree. Low values can prevent overfitting but can make specific cases harder to learn. Typical values: 0.5 - 1
Colsample by tree	0.75,1	Fraction of the features to be used in each tree. Typical values: 0.5-1
Replace missing values	Yes [✓ No]	
Parallelism	-1	Number of cores used for parallel training. Using more cores leads to faster training but at the expense of more memory consumption, especially for large training datasets. (-1 means 'all cores')

Image 19. XGBoost

The other model was the Random Forest for decision tree classification and is a simple algorithm which builds a decision tree. Each node of the decision tree includes a condition on one of the input features. A Random Forest classifier is made of many decision trees. When predicting a new record, it is predicted by each tree, and each tree "votes" for the final answer of the forest.

The forest chooses the class having the most votes. When "growing" (ie, training) the forest:

for each tree, a random sample of the training set is used;

for each decision point in the tree, a random subset of the input features is considered.

Random Forests generally provide good results, at the expense of "explainability" of the model.

Once the models are trained, the next step is to apply them on the test dataset to predict the fault severities. Then, the result is prepared one last time before the submission in the website of the competition. The final preparation consist on leaving just the ID and the predict_0, predict_1 and predict_2 columns, then export the .csv file and upload it to the platform to obtain both private and public score.

This is how the flow looks at the end and with some other predictive model trials on the same datasets:

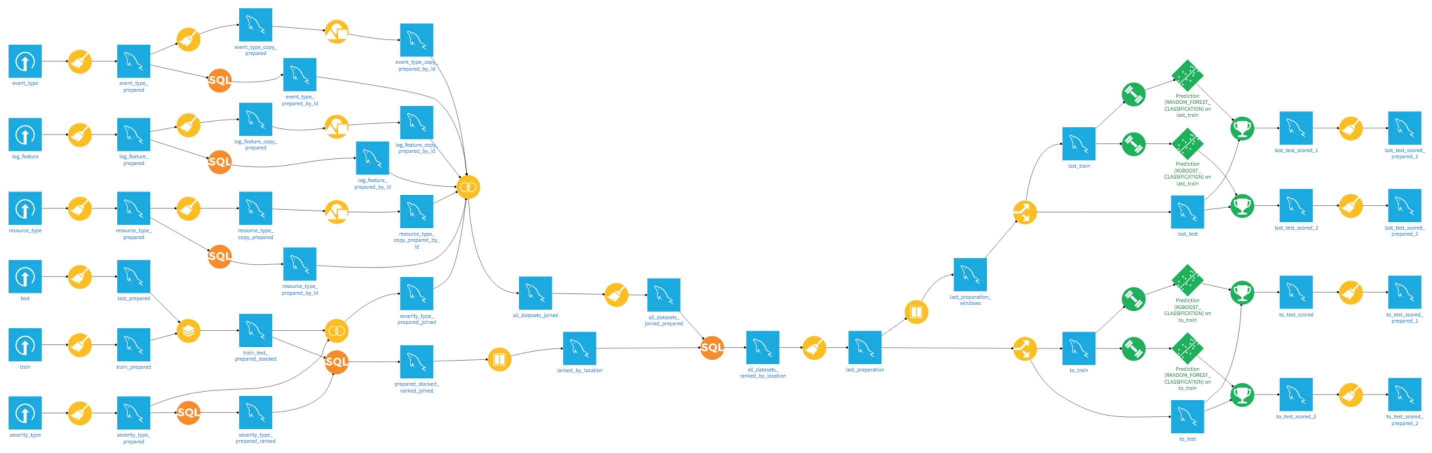
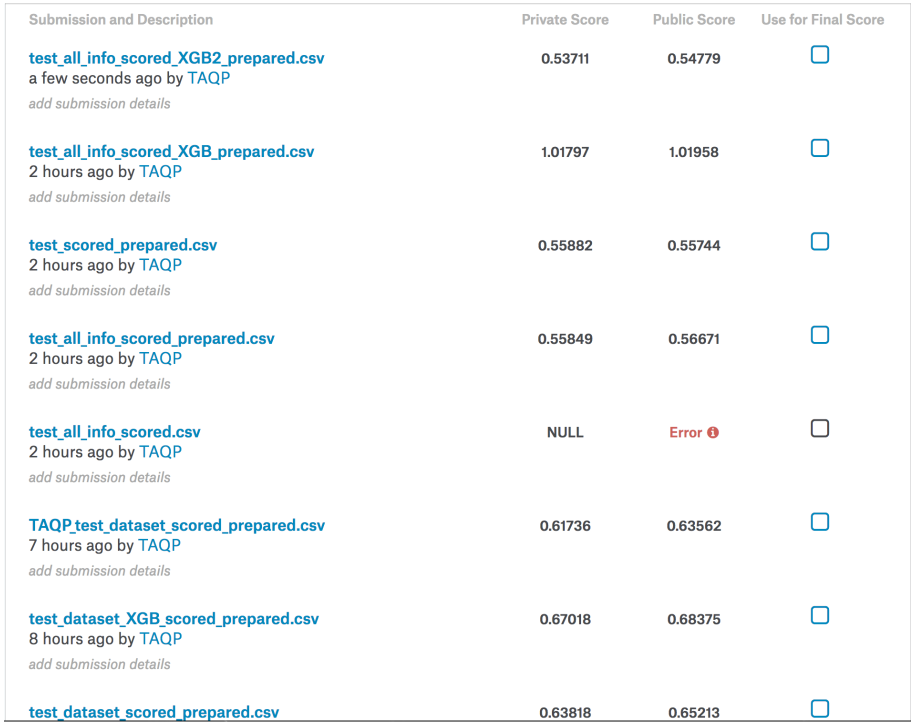

event_type
event_type_prepared
event_type_copy_prepared
event_type_prepared_by_id
event_type_copy_prepared_by_id
log_feature
log_feature_prepared
log_feature_copy_prepared
log_feature_prepared_by_id
log_feature_copy_prepared_by_id
resource_type
resource_type_prepared
resource_type_copy_prepared
resource_type_prepared_by_id
resource_type_copy_prepared_by_id
test
test_prepared
train
train_prepared
train_test_prepared_stacked
severity_type
severity_type_prepared
severity_type_prepared_ranked
severity_type_prepared_joined
prepared_stacked_ranked_joined
SQL
all_datasets_joined
all_datasets_joined_prepared
ranked_by_location
all_datasets_ranked_by_location
last_preparation
last_preparation_windows
last_train
last_test
to_train
to_test
Prediction (RANDOM_FOREST_CLASSIFICATION) on last_train
Prediction (XGBOOST_CLASSIFICATION) on last_train
Prediction (XGBOOST_CLASSIFICATION) on to_train
Prediction (RANDOM_FOREST_CLASSIFICATION) on to_train
last_test_scored_1
last_test_scored_prepared_1
last_test_scored_2
last_test_scored_prepared_2
to_test_scored
to_test_scored_prepared_1
to_test_scored_2
to_test_scored_prepared_2

Image 20. Final flow

6. RESULTS AND CONCLUSION

At first, the results of the submissions were above the 500th positions in the leaderboard. There were just a few features and some basic preparation in the datasets. However, being curious with the DataScienceStudio and in general with what Data Science is about, there was a considerable improvement in the model and features besides a very exiting and fun learning process.



Submission and Description	Private Score	Public Score	Use for Final Score
test_all_info_scored_XGB2_prepared.csv a few seconds ago by TAQP *add submission details*	0.53711	0.54779	☐
test_all_info_scored_XGB_prepared.csv 2 hours ago by TAQP *add submission details*	1.01797	1.01958	☐
test_scored_prepared.csv 2 hours ago by TAQP *add submission details*	0.55882	0.55744	☐
test_all_info_scored_prepared.csv 2 hours ago by TAQP *add submission details*	0.55849	0.56671	☐
test_all_info_scored.csv 2 hours ago by TAQP *add submission details*	NULL	Error	☐
TAQP_test_dataset_scored_prepared.csv 7 hours ago by TAQP *add submission details*	0.61736	0.63562	☐
test_dataset_XGB_scored_prepared.csv 8 hours ago by TAQP *add submission details*	0.67018	0.68375	☐
test_dataset_scored_prepared.csv	0.63818	0.65213	☐

Image 21. Competition submissions

The datasets for this competition seemed really simple at first but turned out to have some hidden depths and tricky but significant steps. It was more about feature engineering to get a good result, rather than tuning a powerful predictive model.

If there is one take away from this exercise, is that you have to be creative and the most important thing to do is to first capture all the possible information presented in the data. Data Science is about working out how to solve not so straight forward problems, with exploration, feature engineering and finally Predictive Modeling using advanced machine learning algorithms.

Personally, the fact that competitors who stand out were considered for data science roles in Telstra's Big Data team is really amazing and means that Data Scientists are being highly requested by the industry.

The final and best result on the submissions was a private score of ~0.52 (position 414). However, there are more features in development to keep improving the entire model.