# ML Hackathon – TLC Experimentation

This document uses the ML Hackathon contest problem – Predicting correct bug routing – as an example of ML experimentation in TLC. While most examples refer to the contest problem, the Sample TLC Run section is a general tutorial, applicable to other problems.

## Problem Description

The ML Hackathon’s challenge is about automatic routing of Cortana bugs to the correct area path, using bug data available at the time a user feedback is submitted – e.g., problem description, build number, binary area, etc. - and a debugging trace log collected at the time of submission (ztrace.log). A sample of the data with a list of available bug attributes is in Appendix A.

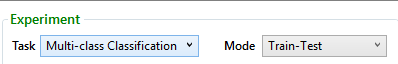
This handout describes a sample workflow from training to testing to submission of results, using TLC. TLC supports the extraction of basic features such as the transformation of categorical feature values to binary features, and extraction of term frequency weighted bag of words from text features. To engineer and extract additional features from the data provided, you are free to use any language and/or tool or your choice, such as R, Python, Perl, etc.

## Sample TLC Run – Training, Testing, Saving Predictions

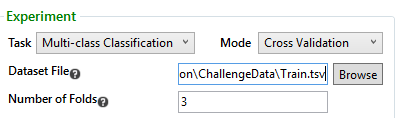
### Step 1 - Describe the Experiment Environment (Task – Input – Output)

To get started with a new experiment, the following are the common steps:

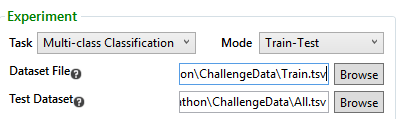
1. Select the **Task** type, e.g., **Multiclass Classification**.
2. Select the experiment mode, e.g., **Cross Validation**, or **Train-Test**.



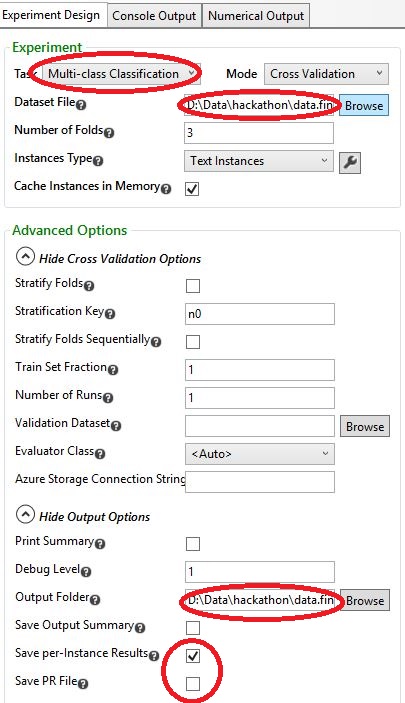
1. If you select **Cross Validation** in (b):
   * Browse to locate your training dataset.
   * Select the number of cross validation folds to run.



1. If you select **Train-Test** in (b):
   * Browse to locate your training dataset.
   * Browse to locate your testing dataset.



1. Select your output options
   * Output folder: where to save the output file(s)
   * To save the detailed predictions per instance, check **Save per-Instance Results**.
   * Do not check **Save PR file** if you’re running a task other than binary classification.



### Step 2 - Describe Input Format for TLC Parsing

To describe how TLC should parse your input datasets, select the data format type -- e.g., for a CSV or TSV file, select **Text Instances** – then click the **Settings** icon.



The following is an example of data parsing settings of the input bug data. All feature indices are zero-based. Please note that this is not a recommendation for how you should handle the challenge data; it is provided as an example only.

1. Data has a header line,



1. All feature values are defined, i.e., the data is dense,



1. Fields are tab-separated,



1. The target class (label) field is the first, i.e., index# 0,



1. The second field is an identifier which should not be considered a feature,



1. Features 1 to 4 are categorical, i.e., discrete values which should be converted to a set of binary features corresponding to each unique value appearing in the data,



1. Features 13 and 14 are text features, i.e., full text strings which should be transformed into a bag of words,



1. Ignore (do not use) features numbered 5 to 12,



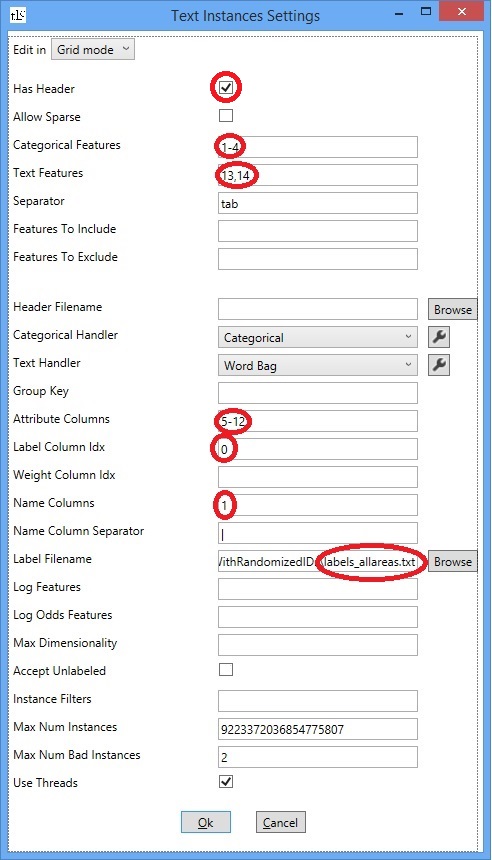
1. All other features are numeric 🡪 No settings required. This is the default type.
2. Target labels from the input datasets should be mapped to numeric values according to a mapping file.



You may optionally choose to change the default handling mode of categorical features and/or text features by clicking the **Settings** icon next to **Categorical Handler** and/or **Text Handler**.



The parsing selections above are summarized in the following **Text Instances Settings** figure.

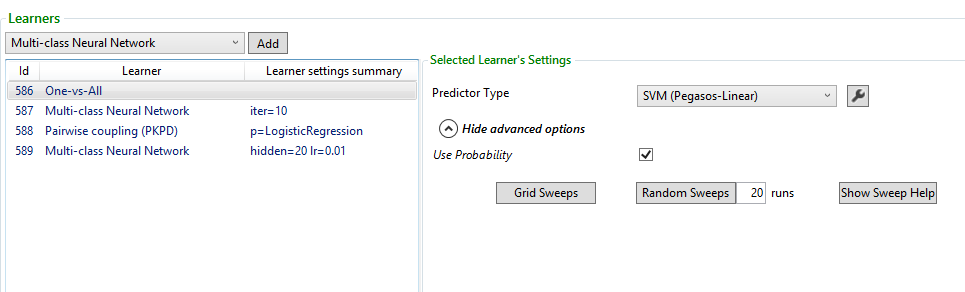


To define the same settings in the TL.exe command-line, use the following /inst parameters:

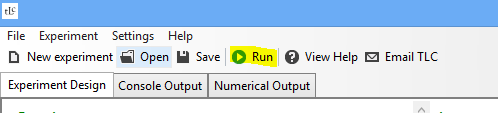
/inst=TextInstances{header=+ sparse=- cat=1-4 text=13,14 attr=5-12 label=0 name=1 lf=path\_to\_labels\_allareas.txt}

### Step 3 – Training / Parameter Sweeping

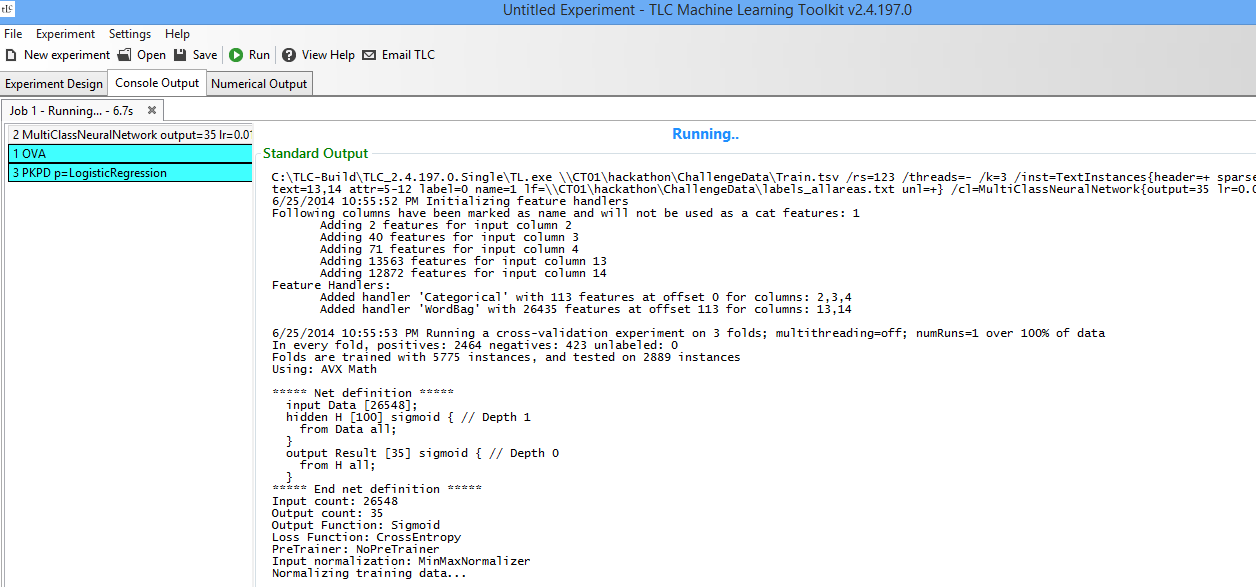
To train, tune, and select a model in TLC, select as many learners as desired in the middle panel. For each learner, you may change the learning hyper-parameters in the right panel. Multiple learners of the same or different algorithms can be added to any experiment.



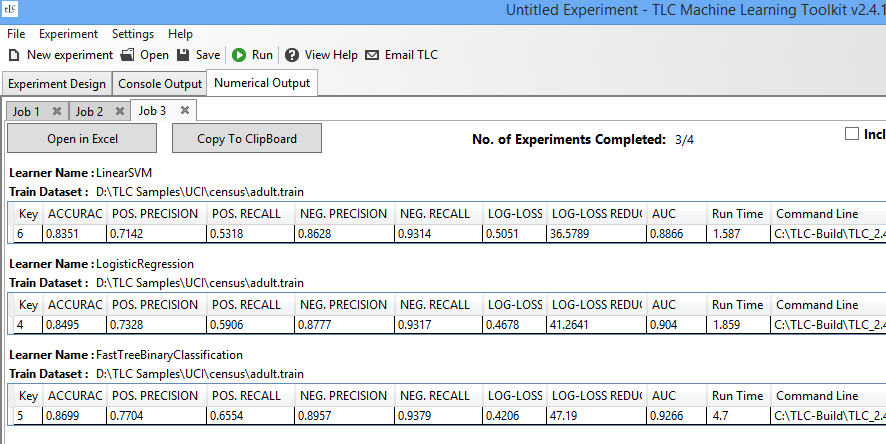
To run all selected learners with the set hyper-parameters, click **Run**.



A detailed output form each learning job will appear under the **Console Output** tab.

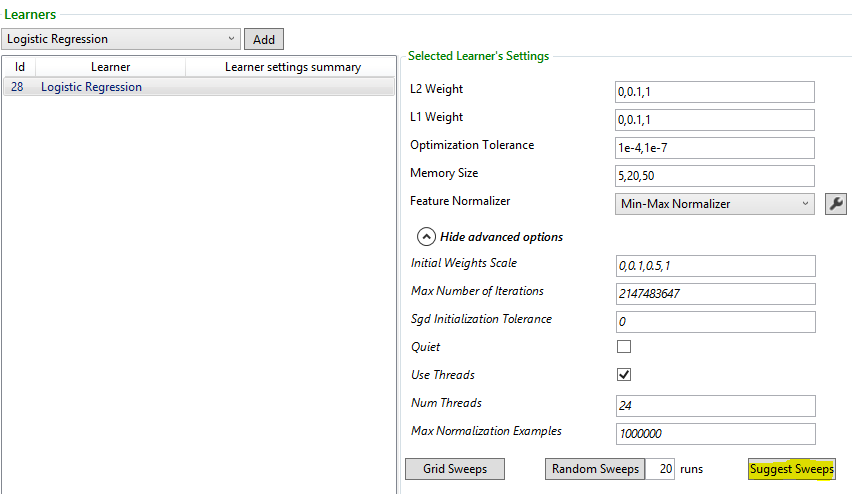


Click the **Numerical Output** tab to view/copy/save the experiment results of all learners run.

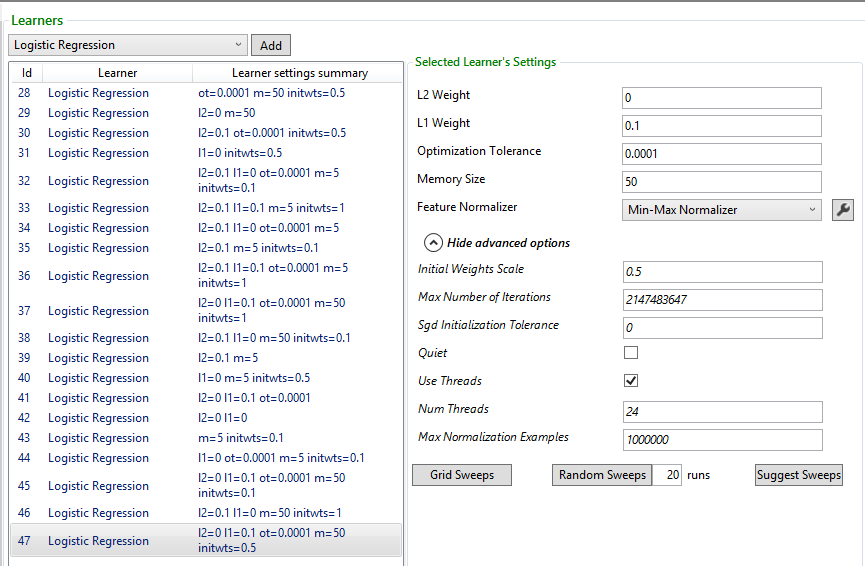


### Parameter Sweeping

Parameter sweeping of any set of hyper-parameters is supported in most tasks and learners, excluding meta-learners. To run a full grid or random sweep of hyper-parameters, add a learner to your experiment, click the experiment in the middle panel then set the parameters range you wish to sweep over. You may also use the **Suggest Sweeps** button for a suggestion of default ranges.



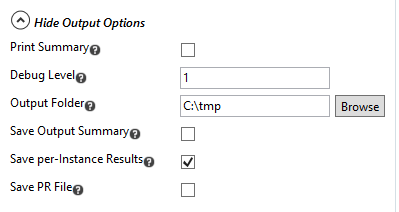
Select to run a full **Grid Sweeps** or a **Random Sweeps** with a selected number of runs. Variations of hyper-parameters within defined ranges will be added to your experiment.



### Step 4 – Generate the per-Instance Predictions

When ready to generate the final predictions, you can either:

* Run a **Train-Test** task to train the final model with the best found hyper-parameters and test it with the testing dataset in one run, or
* Run a **Train** task to train and save the final model, then run a **Test** task to generate the predictions.
* Check the output option to **Save per Instance Predictions** in the left panel -> **Output Options**.



## (Optional) Using TLC to dump feature vectors to a file

Should you wish to export the features extracted by TLC to a file to be used in other tools and/or augment the features with those generated using your own scripts or programs, you can do so using the **TL.exe** command line in a DOS command window. **CreateInstances** is the TL.exe mode which dumps the features to a file in the selected format.

The following example would save the extracted bag of words/term frequency features corresponding to field# 14 to a file. You may use any combination of /inst parameters, similar to the parsing options.

TL.exe *path\_to\_training\_data* /c CreateInstances /inst=TextInstances{header=+ sparse=- text=14 label=0 lf=*path\_to\_labels\_file*}

The feature vectors will be saved in a file named the same as the input file (*path\_to\_training\_data above*), ending with a .tlc extension.

The exported feature vector may look like the following:

0 12786 0:100 1:100 2:101 3:100 4:181 5:107 6:100 7:2 8:13 9:3 10:2 11:3 12:1 13:1 14:10 15:1 16:10 17:8 18:5 19:2 20:1 21:6 22:1 23:1 24:2 25:1 26:1 27:3 28:3 29:1 30:1 31:66 32:1 33:1 34:13 35:9 36:13 37:4 38:3 39:9 40:3 41:16 42:30 43:16 44:6 45:6 46:11 47:5 48:8 49:2 50:2 51:2 52:1 53:1 54:2 55:24 56:4 57:3 58:23 59:3 60:1

Where the first value is the label numeric value, followed by the maximum dimensionality in the case of sparse vectors, followed by a sequence of *featurenum*:*value* pairs.

To use the feature vectors in a different ML tool, reformat the instances file according to the requirements of the external tool.

## Feature Engineering and Feature Extraction

To engineer and extract additional features which are not supported in TLC, use your favorite language/packages/tools to generate the new features.

To train a model using the new features – either in isolation or combined with old features – reformat the new instances according to the TLC parsing options of your choice.

Examples of packages/tools for text features extraction:

* R: tm, textir, RTextTools
* Python: gensim

## Preparing Results for Submission

**Note: This section applies to the ML Hackathon Contest track only**.

### Step 1a – Convert TLC output to submission format

If you are using TLC for training/prediction, follow these steps to convert the TLC .inst.txt output file to the format expected by the scoring program

1. Copy tlc2codalab.zip from [\\ct01\hackathon](file:///\\ct01\hackathon) to your local directory and extract, this should give you the tlc2codalab.exe tool
2. (Optional) Copy over the .inst.txt file output from TLC to the same local directory where you have the conversion tool
3. (Optional) Copy labels\_allareas.txt from [\\ct01\hackathon\ChallengeData](file:///\\ct01\hackathon\ChallengeData) to the same local directory
4. Open a command line and navigate to the above local directory and use the following syntax

**tlc2codalab tlcinstfilename labelmappingfilename outputfilename**

tlcinstfilename, labelmappingfilename, and outputfilename may be either in the same local directory or specified by a full path.

### Step 1b – Convert output from other tools to submission format

If you are using other tools for training and prediction, prepare your results for submission in the following tab-separated format (Note: Header line below is required):

BugID Path

10304438 Windows Phone Blue\Experiences\_DevPartners\Bing\CU

### Step 2 – Prepare your results for submission

1. Copy over scoringprogram.zip from [\\ct01\hackathon](file:///\\ct01\hackathon) to local directory and extract, this would give you ScoringProgram.exe tool in the extracted directory
2. Use the following syntax from command line (after navigating to directory where scoringprogram.zip was extracted)

**scoringprogram prepare C:\hackathon\finalset.tsv**

You would be prompted to enter comments (in the command line window), after which the tool prints out

* The location of the local generated .zip file for submission
* The url location where you need to go to submit it

1. Please navigate to the above url location, sign in with your team credentials and submit the output .zip file by navigating to the local directory above (in step 2.b)

## Appendix A - Data description

1. Bug data at time of feedback submission

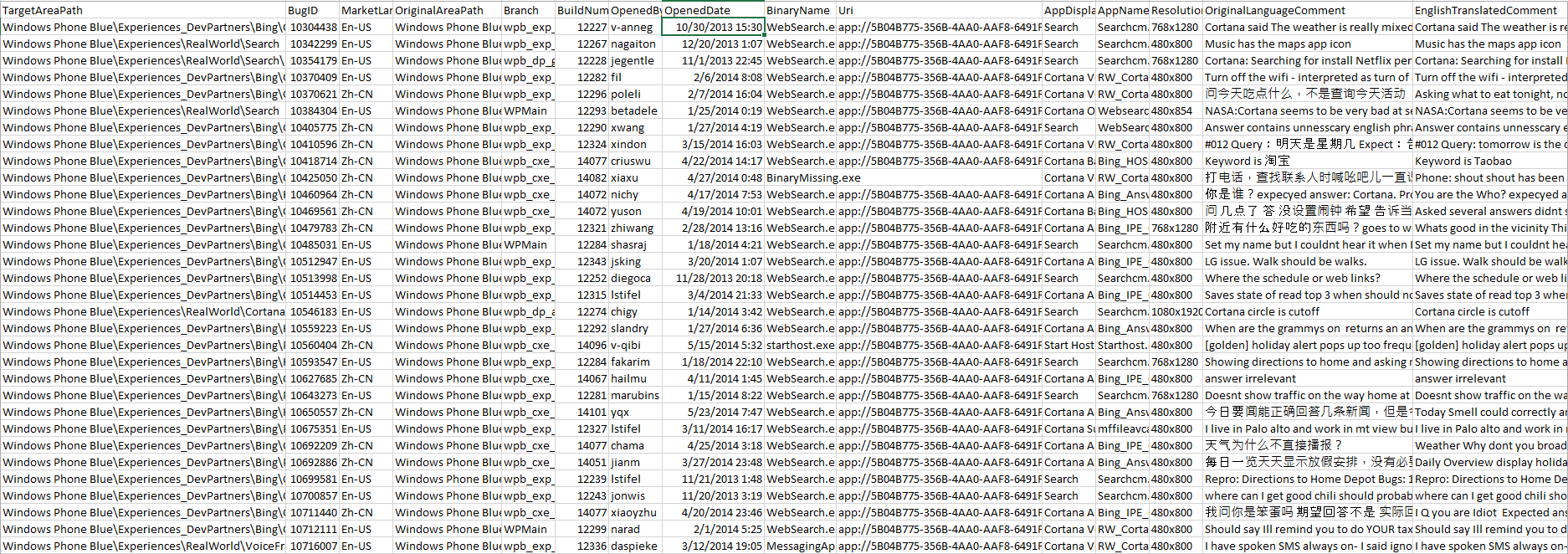
The following data attributes are available at the time of feedback submission:

**BugID, OriginalAreaPath, Branch, BuildNum, OpenedBy, OpenedDate, BinaryName, URI, AppDisplayName, Resolution, Original LanguageComment**

Additional data attribute:

**MarketLanguage**: indicates the source market/language where the feedback was submitted (En-US: United States – English, Zh-CN: China – Simplified Chinese)

**TargetAreaPath** is the class name to be predicted, which indicates the correct area path to which the bug should be routed.



1. Ztrace logs collected from Windows Phone at time of feedback submission

Debugging traces collected in a supporting text file attachment, reflecting the phone activity at (and before) the feedback submission time.