How to use Brunel's tqZ Code Documentation

The executable "**analysisMain.exe**", contains a number of methods which are used to produce results for analysis of tqZ. These methods are accessible through a number of arguments. This documentation covers the standard processes in using the executable, and its compilation. It is by no means a complete guide, but one intended provide a starting point and to give others documentation which the author desperately lacked during his student days. The author takes no responsibility for any unintended consequences stemming from use of these instructions.

If it is not covered in this document, your best bet is to run "**analysisMain.exe --help**" or "**analysisMain.exe -h**" to display a help message displaying all the argument options. If that fails, my only advice is to prepare for some diving into code to find your answer…

Contents

Compiling the program:

In order to compile the program using the provided makefile, the user needs to execute the following console command in the program's main directory:

```
>>> make
```

The makefile will need to be modified to take into account local library locations and the user will need root and config++ libraries installed for the program to compile and execute.

Configuration Files:

When running any part of the analysis process, the executable needs a configuration file to use. The example configuration file "eeeConf.txt" is found in the directory "configs/" and the various example sub configuration files are found in the sub-directories noted in the example configuration file "eeeConf.txt". The contents are:

```
datasets = "configs/datasets/eeeDataAndMC.txt";
cuts = "configs/cuts/eeeCuts.txt";
plots = "configs/plots/plotConf.cfg";
outputFolder = "plots/eee/";
outputPostfix = "eee";
channelName = "eee";
```

datasets: the location of the text file denoting each physics channel, the location of the root file list for the channel, the number of events and cross section for the channel, the data type (MC or data), and histogram name, colour, label and plot type.

cuts: the location of the text file denoting the physics cuts for tight and loose electrons and muons. This doesn't' need altering for new config files unless one has new selection cuts.
plots: the location of the text file with the instructions for plot names, axis widths, number of bins and axis labels.

outputFolder: the output directory for plots.

outputPostfix: the suffix used for output plots.

channelName: name of physics channel being analysed.

The sub-configuration files for datasets follow this general structure:

```
[tZq]
fileName = configs/datasets/fileLists/tZqFiles.txt
totalEvents = 1038665
crossSection = 0.03590
```

```
runType = mc
histoName = tZq
colour = kYellow
label = tZq
plotType = f
```

The “fileName" parameter denotes the path to a text file which lists the locations of the various input files (usually the output of the nTupiliser). Hopefully the other contents should be self-explanatory.

It is also worth noting that whilst the lepton channel is determined in the configuration file, there is an argument which overrides this:

> -k : bit mask for lepton channel selection; 1 - eee, 2 - eeμ, 4 - eμμ, 8 - μμμ, 16 – eee (inverted), 32 - eeμ (inverted), 64 - eμμ (inverted), and 128 μμμ (inverted). To run multiple channels in the same session, add the digits together, eg. 15 – all lepton channels, 240 – all inverted lepton channels, 255 – all lepton channels and inverted lepton channels.

Part 1 – Creating skims:

The first stage of producing results involves the creating of skim files. Before running this stage, the directory “skims” needs creating. To create the skims one uses the following command:

```
>>> ./tbZAnal.exe -c <user-config-file> -g -j
```

The arguments are described as follows:

> -c <user-config-file>: see above.
>
> -g: make post-lepton selection trees in the skim files.
>
> -j: make the B-Tagging efficiency trees in the skim files.
>
> -k <bit-mask>: see above (optional).

Part 2 (option 1)– Creating mvaFiles:

The second stage of producing results initially involves the creating of mva files. Before running this stage, the directories “mvaTest”, “mvaDirs/inputs” and “mvaDirs/skims” need creating. To create the mva files one uses the following command:

>>> ./tbZAnal.exe -c <user-config-file> -u -v <SYST> -z

The arguments are described as follows:

-c <user-config-file>: see above.

- u: use the post-lepton selection trees in the mva files.

- v <SYST>: do the desired systematic. This argument isn't required if one is running the program over systematic files, by their virtue of already being systematics! SYST is defined as$(2^{Number\ of\ Up\ and\ Down\ systematics} - 1)$. So for systematics for JES, JER, b-tagging, trigger, pileup and PDF, SYST = $(2^{6x2} - 1)$ = 4095.

-z, --makeMVATree: produce a tree after event selection for mva purposes.

-k <bit-mask>: see above (optional).

-t: use B-Tagging reweighting.

--jetRegion <nJets,nBjets,maxJets,maxBjets>: Sets the jet region to be looked out (optional).

--mvaDir: <directoryPath>: custom directory to output mva files to (optional).

--metCut: the cut on the MET one wishes to use during the analysis (optional).

--mtwCut: the cut on the W's transverse mass one wishes to use during the analysis (optional).

Following the creation of the mva files, the creation of the files used for the BDT tool involves the python script "scripts/makeMCAInput.py" (I am told that the script's name involves a typo). To run the script, one uses the following command:

>>> python ./scripts/makeMCAInput.py <channels> <MvaInputDir> <BdtOutputDir>

The arguments are described as follows:

<channels>: Reads in the channel one is using. A list of channels can be run over.

<MvaInputDir>: Directory where mva input files are read in from.

<BdtOutputDir>: Directory where BDT input files are outputted to.

Part 2 (option 2)– running mvaFiles to BDT python script:

The second stage of producing results involves the creating of mva files and BDT input files. Before running this stage, the directories “mvaTest”, “mvaDirs/inputs” and “mvaDirs/skims” need creating. To create the run the script to produce both the mva files and the BDT input files, one uses the following command:

```
>>> python ./scripts/runAnalToBDT.py <METcut> <mtwCut>
```

The arguments are described as follows:

<METcut>: the cut on the MET one wishes to use during the analysis.

<mtwCut>: the cut on the W’s transverse mass one wishes to use during the analysis.

Aside - Producing Plots:

An optional stage involves the creation of output plots. Before running this stage, the directory “plots/<channel-name>” (eg. <channel-name> could be “eemu”) needs creating. To create the plots one uses the following command:

```
>>> ./tbZAnal.exe -c <user-config-file> -p
```

The arguments are described as follows:

-p: makes all plots.

-k <bit-mask>: see above (optional).

Part 3 - Producing Plots:

The stage of the analysis uses a slightly altered version of jandrea’s SingleTop_tZ_Macro (https://github.com/jandrea/SingleTop_tZ_Macro), where “/TMVA/theMVAtool.C” has been altered so that our variables and input files are used. This slightly altered code can be found here https://github.com/davidcarbonis/SingleTop_tZ_Macro.

Before using this macro, the input mva files from the previous stage must be either copied to “/TMVA/inputfiles/” or the macro “/TMVA/theMVAtool.C” must be amended to reflect your chosen input directory.

To use the macro, one, in the directory "/TMVA", uses the following console commands:

```
>>> root –l theMVAtool.C+
>>> theMVAtool tmva
>>> tmva.doTraining( "<inputDir>", "<channel>", "<numberOfTrees>" )

>>> tmva.doReading( "<inputDir>", "<outputDir>", "<channel>")
```

The training produces the weight files for the BDT, and on completion loads up the TMVA GUI, which has various options to see how the training and test samples fare and the performance of the BDT.

The reading reads the TTree and calculates the BDT output for all events in the evaluation sample and produces templates (MVA distribution).

The arguments are described as follows:

<inputDir>: Directory containing the input files to be read into the TMVA's BDT algorithm for both the training and reading stages. Default is "*inputroot/met0mtw0/*".

<outputDir>: Directory containing the input files to be read into the TMVA's BDT algorithm for the reading stage. Default is "*outputroot/met0mtw0/*".

<outputDir>: Flag for which channel to be run over. Choices are "all", "eee", "eemu", "emumu", and "mumumu". Default is "all".

<numberOfTrees>: The number of trees the BDT is to be trained over. Default is "100". Note that currently the depth of the trees is hard coded, a further argument to include this functionality in a more flexible and user friendly manner is antipciated in the near future.

Part 4 - Producing input for theta:

Following running the training and reading of the BDT trees in stage 3, one can create the files necessary to run theta with. The directory "/TMVA/TemplateRootFiles" must exist. Firstly, the output files from the previous step need to be merged, through the following console commands (or for whichever channel you wish to run over):

```
>>> hadd outputroot/output_merged.root outputroot/…/output_all_*.root

>>> hadd outputroot/output_merged_eee.root outputroot/…/output_eee_*.root

>>> hadd outputroot/output_merged_eemu.root outputroot/…/output_eemu_*.root
```

```
>>> hadd outputroot/output_merged_emumu.root outputroot/…/output_emumu_*.root

>>> hadd outputroot/output_merged_mumumu.root
outputroot/…/output_mumumu_*.root
```

Following this, to use the macro which produces the input files for theta, use the following console command:

```
>>> root –l ProdTemplate.C+
```

Following this, the output (input for theta) is found in “/TMVA/TemplateRootFiles”. Note, if you are creating the output for different cuts, the ProdTemplate.C output names won’t label the cut which has been run over.

Potential Problems:

- Users are recommended not to run Crab3 setup scripts before using this problem. It has been found that this can cause compilation issues when using the makefile and also causes problems when using the python scripts as it apparently unloads NumPy.