Final Project Demonstration

Application of Machine Learning Techniques to Next Generation Sequencing Quality Control

Sam Nicholls
msn

Department of Computer Science
Aberystwyth University

May 15, 2014

Aims

Report on state of the current quality control system at Sanger

- Working with the Wellcome Trust Sanger Institute's Human Genetics Informatics Team
- `auto_qc` classifies samples as pass, fail or warn
- Current classifier consists of hard-coded simple thresholds
- `auto_qc` also requires timely human intervention

Goals

- Apply learning techniques to replicate current human rules
- Attempt to improve efficiency of current "warning" handling
- Identify new or unused parameters that improve classification

Aims

Report on state of the current quality control system at Sanger

- Working with the Wellcome Trust Sanger Institute's Human Genetics Informatics Team
- `auto_qc` classifies samples as pass, fail or warn
- Current classifier consists of hard-coded simple thresholds
- `auto_qc` also requires timely human intervention

Goals

- Apply learning techniques to replicate current human rules
- Attempt to improve efficiency of current "warning" handling
- Identify new or unused parameters that improve classification

Input Data and Format

Input: **Lanelet** QC Data

- Access to two of the largest studies at the institute
- 13,455 "lanelets"; aggregated clusters of a sample in one lane
- `auto_qc` **pass** 9,154 (68%), **fail** 1,542 (11%) **warn** 2,759 (21%)

Input Format: **"BAMcheckR'd"** Text Files

- Key-value statistical summary numbers from `samtools stats`
- `samtools stats` also generates tab-delimited dataframes measuring some metrics over cycle time
- Additional summary numbers gained by passing output of `samtools stats` through internal Sanger tool, `bamcheckr`

Handling Data

Introducing **Frontier**

- A Python package providing interfaces for reading, storage and retrieval of machine learning data sets
- Users write their own reader classes but need only provide implementations of two functions so any file can be used as input
- Presents an API for manipulation and extraction of stored data-target pairs, allowing filter by parameters or classes
- Supports 'any' machine learning problem – user merely provides simple definitions of the labels
- Returns data via the API in efficient **NumPy** containers for direct use with the **scikit-learn** framework
- Quick and easy logging of machine learning experiments

Frontier Example Usage

```
from Frontier import frontier
from Frontier.IO import DataReader, TargetReader

data_dir = "/home/sam/Projects/owl_classifier/data/"
target_path = "/home/sam/Projects/owl_classifier/targets.txt"

CLASSES = {
        "hoot": {
            "names": ["owl", "owls"],
            "code": 1,
        },
        "unhoot": {
            "names": ["cat", "dog", "pancake"],
            "code": 0,
        },
}

statplexer = frontier.Statplexer(data_dir,
                                 target_path,
                                 CLASSES,
                                 DataReader,
                                 TargetReader)
```

Introduction to the Frontier API

Feature Inspection

- **list_parameters**
  Return a sorted list of all parameters
- **find_parameters**
  Return parameters which contain any of the input strings as a substring
- **exclude_parameters**
  Return parameters which do not contain any of the input strings as a substring

Data-Target Extraction

- **get_data_by_parameters**
  Return data for all observations, but only include columns for each parameter in a given list
- **get_data_by_target**
  Return data for observations that have been classified as one of the targets specified and only return columns for the parameters in the given list

Contributions to Current QC System

- `bamcheckr`; an in-house tool written in R
- Supplements `samtools stats` key-value summary numbers which are then used by the current `auto_qc` system

What did I do?

- Patched a bug that prevented plotting of diagnostic graphs
- Authored additional routines to recover "missing" percentage and ratio based quality parameters that were typically calculated outside of bamcheckr...
- ...although **Frontier** turned out to be more efficient for this task

Testing Parameter Sets

Set	#	CV ± SD	SCV ± SD	Depth	Most Important Feature
ALL	86	90 ± 4	97 ± 1	38	T-percent-max-baseline-deviation (27%)
AQC	27	87 ± 4	95 ± 1	36	T-percent-max-baseline-deviation (31%)
AQCN	21	86 ± 4	95 ± 1	39	max-max-baseline-deviation (31%)
ERROR	1	60 ± 6	61 ± 2	53	error-rate (100%)
NO_ERROR	85	90 ± 4	97 ± 1	38	T-percent-max-above-baseline(27%)
BASELINE	34	82 ± 5	89 ± 1	46	T-percent-max-above-baseline(28%)
NOBASELINE	52	72 ± 10	91 ± 1	31	error-rate (24%)
MARP	47	87 ± 4	95 ± 1	39	T-percent-max-above-baseline (27%)
NO_MARP	39	75 ± 7	87 ± 1	38	max-max-baseline-deviation (34%)

Table : **Parameter Set Cross Validation Scores**: Results of classifying testing data into one of three classes; pass, fail or warn. Columns left to right; parameter set name, number of parameters included, average cross-validation score (max 100) ± std. deviation, average stratified cross-validation score (max 100) ± std. deviation, average depth of the generated tree and the most important parameter by Gini importance (max 100). Tree depth and parameter importance was estimated on experiments using the stratified data.

- No surprise that parameter superset **ALL** validates well
- **AQC** and **AQCN** score highly despite far smaller models
- Generally good performance, possibly reflecting simple linear nature of underlying rules or possible bias (lots of passes)

ALL vs. AQC

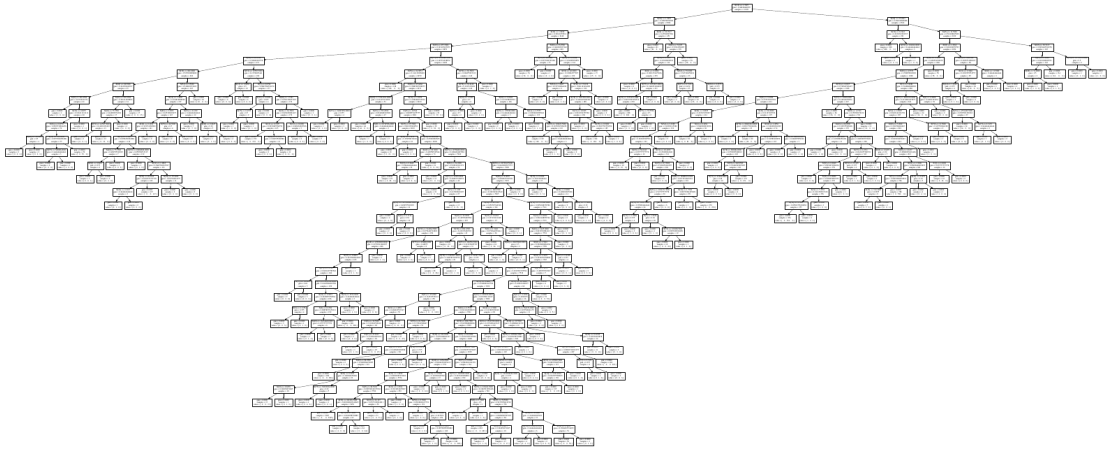


Figure : **ALL Set Decision Tree**

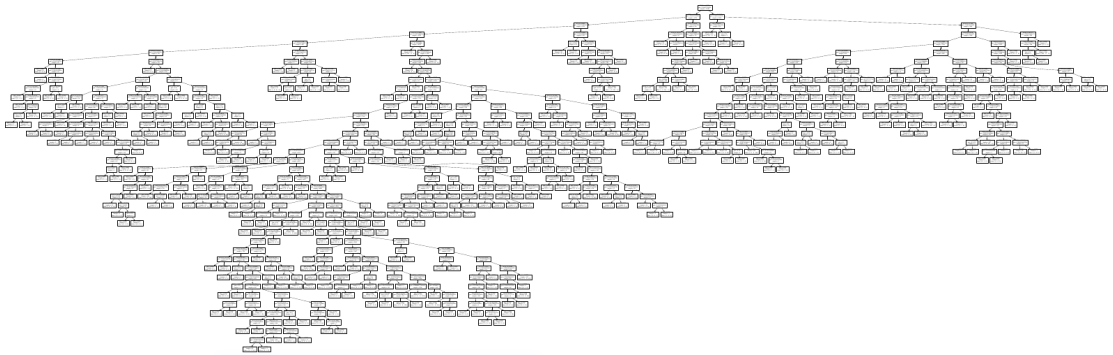


Figure : **AQC Set Decision Tree**

https://github.com/SamStudio8/frontier/tree/master/results/front/

Overfitting

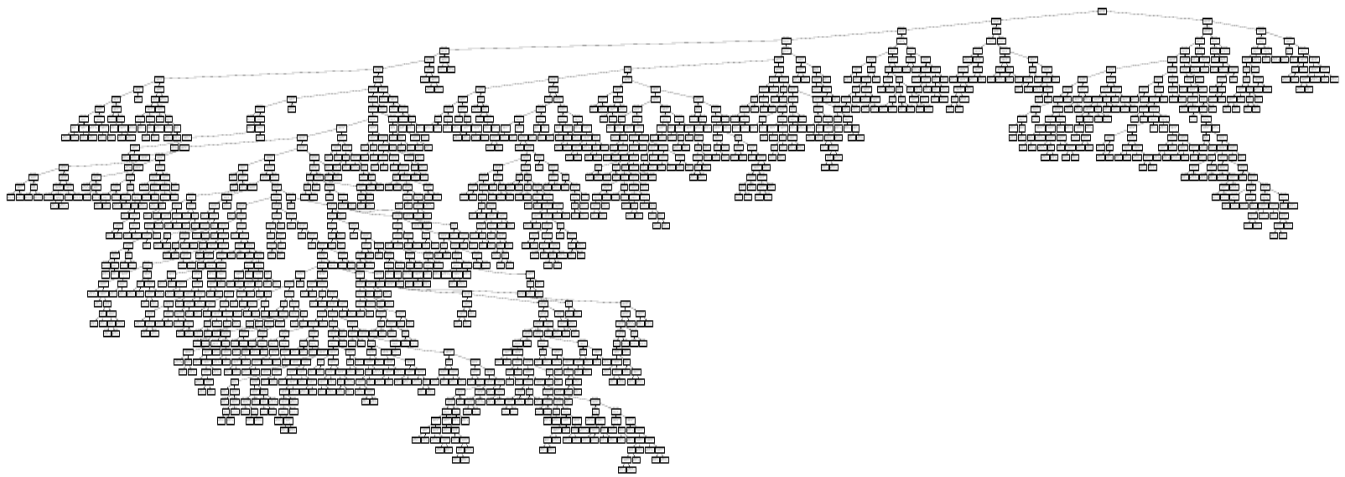


Figure : **BASELINE Set Decision Tree**

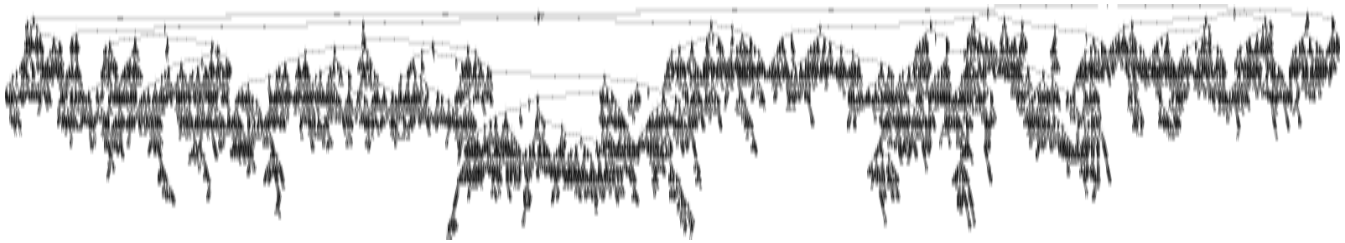


Figure : **ERROR Set Decision Tree**

https://github.com/SamStudio8/frontier/tree/master/results/front/

Augmenting Warning Handling

PSet	DSet	#	CV $\pm$ SD	SCV $\pm$ SD	Depth	Most Important Feature
ALL	IGNWARN	86	96 $\pm$ 3	99 $\pm$ 0	22	error-rate (43%)
ALL	WARNPASS	86	95 $\pm$ 3	99 $\pm$ 0	33	quality-dropoff-rev-mean-runmed -decline-low-value (32%)
AQC	IGNWARN	27	94 $\pm$ 4	98 $\pm$ 1	26	error-rate (44%)
AQC	WARNPASS	27	92 $\pm$ 4	98 $\pm$ 1	33	quality-dropoff-rev-mean-runmed -decline-low-value (33%)

Table : **Parameter Set Cross Validation Scores using Alternative Warning Handling**: Results of classifying testing data. Columns left to right; parameter set name, data set name, number of parameters included, average cross-validation score (max 100) $\pm$ std. deviation, average stratified cross-validation score (max 100) $\pm$ std. deviation, average depth of the generated tree and the most important parameter by Gini importance (max 100). Tree depth and parameter importance was estimated on experiments using the stratified data. *N.B.* **IGNWARN** and **WARNPASS** data sets perform classifications on two classes (pass and fail) rather than three.

- **IGNWARN** discards, **WARNPASS** recodes as pass
- Very high validation, appears noise has been significant reduced when compared to the previously tabulated results
- Average maximum depth reduced

Backward Elimination

A-percent-mean-below-baseline
duplicate-mapped-ratio
fwd-percent-insertions-above-baseline
insert-size-average
max-max-baseline-deviation
quality-dropoff-fwd-mean-runmed-decline-low-value
quality-dropoff-rev-mean-runmed-decline-low-value
rev-percent-insertions-above-baseline
rev-percent-insertions-below-baseline

Table : **TOP9 Parameter Set**: Features selected by a backward elimination experiment providing all observations to an iterative decision tree classifier and removing the least important feature until cross-validation fell below a threshold.

- Borrowed a method from statistical model design
- Repeatedly refitted trees after removing the least important feature until cross-validation fell below some percentage of the running average
- Would be very interesting to repeat this process for various augmentations of the warnings class handling

TOP9 Experiment Results

PSet	DSet	#	CV $\pm$ SD	SCV $\pm$ SD	Depth	Most Important Feature
RTOP9	ALL	9	87 $\pm$ 6	95 $\pm$ 1	32	max-max-baseline-deviation (32%)
RTOP9	IGNWARN	9	98 $\pm$ 1	99 $\pm$ 0	20	rev-pct-insertions-above-baseline (38%)
RTOP9	WARNPASS	9	95 $\pm$ 3	99 $\pm$ 1	24	quality-dropoff-rev-mean-runmed -decline-low-value (34%)

Table : **Backward Elimination Parameter Set Cross Validation Scores**: Results of classifying testing data. Columns left to right; parameter set name, data set name, number of parameters included, average cross-validation score (max 100) $\pm$ std. deviation, average stratified cross-validation score (max 100) $\pm$ std. deviation, average depth of the generated tree and the most important parameter by Gini importance (max 100). Tree depth and parameter importance was estimated on experiments using the stratified data. *N.B.* **IGNWARN** and **WARNPASS** data sets perform classifications on two classes (pass and fail) rather than three.

- Impressive validation results considering only 9 parameters
- Average maximum tree depth upper bound briefly overlaps the lower bound of the results found from the naive parameter sets
- Clearly we can gain accuracy on class prediction without the need for including every parameter in the model, **TOP9** is even smaller than the **AQC** set!

TOP9 Decision Tree with All Data

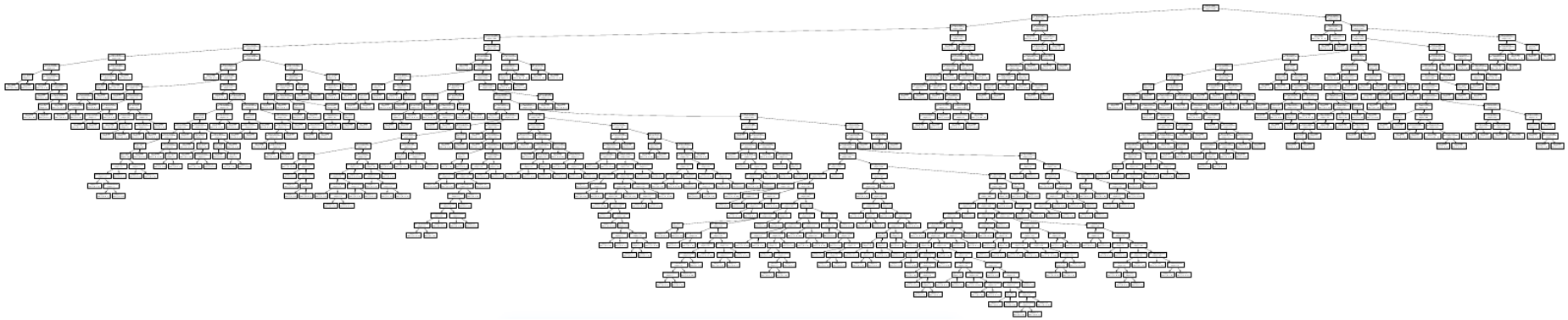


Figure : **TOP9 Set Decision Tree with ALL Data**

https://github.com/SamStudio8/frontier/tree/master/results/front/

- More complex structure and larger size than previous figures that include all parameters
- Further experimentation with the stopping criteria of the backward elimination process should be considered

TOP9 Decision Trees with Warning Augmentations

Figure : **TOP9 Set Decision Tree with IGNWARN Data**

Figure : **TOP9 Set Decision Tree with WARNPASS Data**

https://github.com/SamStudio8/frontier/tree/master/results/front/

TOP9 Example Decision Path

Figure : **TOP9 Set Decision Tree with IGNWARN Data**

- Paths still exhibit some element of arbitrariness
- Need to further investigate criteria for backward elimination
- Surprised that the set is much smaller than **AQC**
- Sensible to investigate some form of pruning to further remove smaller leaves

https://github.com/SamStudio8/frontier/tree/master/results/front/

Aims

Identify lanelet properties that affect downstream variant calling

- For better QC we need an idea of "good" and "bad"
- How does quality affect analyses performed after sequencing?

Goals

- Will leaving out a sample during variant calling affect the result?
- Select a "representative" region of the human genome for analysis
- Compare calls on whole genome samples to "SNP chips"
- Determine what is actually "good" and "bad" for QC

Aims

Identify lanelet properties that affect downstream variant calling

- For better QC we need an idea of "good" and "bad"
- How does quality affect analyses performed after sequencing?

Goals

- Will leaving out a sample during variant calling affect the result?
- Select a "representative" region of the human genome for analysis
- Compare calls on whole genome samples to "SNP chips"
- Determine what is actually "good" and "bad" for QC

Searching for Goldilocks

Introducing **Goldilocks**

- Need to locate an appropriate region for the pipeline to minimise computational time and resources
- Representative region – not too many or too few variants
- Need to handle variant data from two different types of study
- A Python module capable of parsing files containing chromosome-position pairs and conducting a variant census
- Filter and ranks censused regions of a genome based on the number of variants contained
- Presents an API to allow users to call any desired part of the module from other programs and scripts

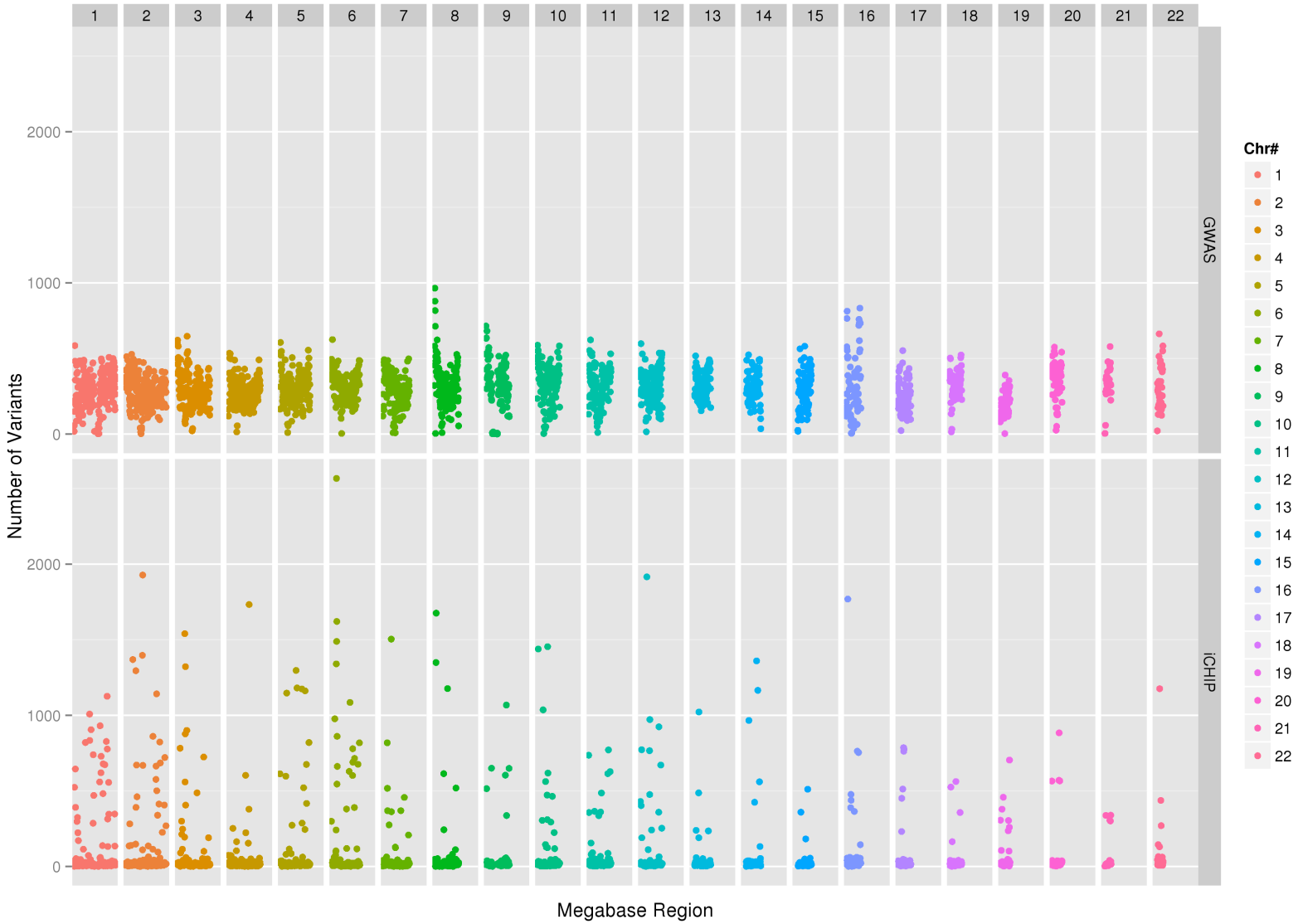
Top 25 1Mnt Goldilocks Regions

i	GWAS	iCHIP	Chr	Start	End
0234	297	470	1	117,000,001	118,000,000
1074*	294	1540	3	46,000,001	47,000,000
5222	294	336	21	16,500,001	17,500,000
3125	298	310	10	60,000,001	61,000,000
0880	293	344	2	191,500,001	192,500,000
3560	299	772	12	9,000,001	10,000,000
4407	299	512	15	78,500,001	79,500,000
1036	292	300	3	27,000,001	28,000,000
2734	300	515	9	5,000,001	6,00,0000
3426	300	486	11	76,000,001	77,000,000
0015	291	1029	1	7,500,001	8,500,000
0365	301	487	1	182,500,001	183,500,000
3415	301	419	11	70,500,001	71,500,000
1581	290	802	4	102,500,001	103,500,000
3554	290	403	12	6,000,001	7,000,000
3184	302	449	10	89,500,001	90,500,000
1580	289	603	4	102,000,001	103,000,000
1948	288	1297	5	96,000,001	97,000,000
2215	288	622	7	49,500,001	50,500,000
0414	288	346	1	207,000,001	208,000,000
2055	304	1377	5	149,500,001	150,500,000
0384	287	827	1	192,000,001	193,000,000
0959	306	406	2	231,000,001	232,000,000
4214	286	393	14	88,500,001	89,500,000
0320	307	620	1	160,000,001	161,000,000

- Filtered by median GWAS (297), ranked by maximum iCHIP
- Initially required over an hour to process the human genome, now completes the task in less than 20 seconds
- Avoided chromosome 6 due to presence of HLA system
- Suitable candidate 1074(*) located on chromosome 3

GWAS vs. iCHIP Variant Densities

GWAS vs. iCHIP Variant Densities



Pipeline Components

- **Extraction**
  Extract the Goldilocks region for all **GWAS** study lanelets
- **Indexing**
  Create indexes for the extracted Goldilocks regions
- **Merge**
  Merge the data from each extracted region into one file
- **Pileup**
  Calculate genotype likelihoods based on the reads seen across all the extracted regions
- **Call**
  Use the genotype likelihood scores to call the variants for each position of interest in each of the Goldilocks regions
- **Compare**
  For each pair of **GWAS** and **iCHIP** samples, measure the concordance of called variants

Pipeline Components

- **Extraction**
  Extract the Goldilocks region for all **GWAS** study lanelets
- **Indexing**
  Create indexes for the extracted Goldilocks regions
- **Merge**
  Merge the data from each extracted region into one file
- **Pileup**
  Calculate genotype likelihoods based on the reads seen across all the extracted regions
- **Call**
  Use the genotype likelihood scores to call the variants for each position of interest in each of the Goldilocks regions
- **Compare**
  For each pair of **GWAS** and **iCHIP** samples, measure the concordance of called variants

Pipeline Components

- **Extraction**
  Extract the Goldilocks region for all **GWAS** study lanelets
- **Indexing**
  Create indexes for the extracted Goldilocks regions
- **Merge**
  Merge the data from each extracted region into one file
- **Pileup**
  Calculate genotype likelihoods based on the reads seen across all the extracted regions
- **Call**
  Use the genotype likelihood scores to call the variants for each position of interest in each of the Goldilocks regions
- **Compare**
  For each pair of **GWAS** and **iCHIP** samples, measure the concordance of called variants

Pipeline Components

- **Extraction**
  Extract the Goldilocks region for all **GWAS** study lanelets
- **Indexing**
  Create indexes for the extracted Goldilocks regions
- **Merge**
  Merge the data from each extracted region into one file
- **Pileup**
  Calculate genotype likelihoods based on the reads seen across all the extracted regions
- **Call**
  Use the genotype likelihood scores to call the variants for each position of interest in each of the Goldilocks regions
- **Compare**
  For each pair of **GWAS** and **iCHIP** samples, measure the concordance of called variants

Pipeline Components

- **Extraction**
  Extract the Goldilocks region for all **GWAS** study lanelets
- **Indexing**
  Create indexes for the extracted Goldilocks regions
- **Merge**
  Merge the data from each extracted region into one file
- **Pileup**
  Calculate genotype likelihoods based on the reads seen across all the extracted regions
- **Call**
  Use the genotype likelihood scores to call the variants for each position of interest in each of the Goldilocks regions
- **Compare**
  For each pair of **GWAS** and **iCHIP** samples, measure the concordance of called variants

Pipeline Components

- **Extraction**
  Extract the Goldilocks region for all **GWAS** study lanelets
- **Indexing**
  Create indexes for the extracted Goldilocks regions
- **Merge**
  Merge the data from each extracted region into one file
- **Pileup**
  Calculate genotype likelihoods based on the reads seen across all the extracted regions
- **Call**
  Use the genotype likelihood scores to call the variants for each position of interest in each of the Goldilocks regions
- **Compare**
  For each pair of **GWAS** and **iCHIP** samples, measure the concordance of called variants

Pipeline Difficulties: Pileup and Calling

Scaling **samtools mpileup**

- Executing a pileup is a performance intensive task
- Initial test runs required 6.5 hours of CPU time with 1GB RAM
- Significant overhead with thousands of sample files, couldn't have performed this step multiple times without merging

Compatibility with **bcftools call**

- Produced only standard header information and no data as the pileup task had not included an appropriate reference sequence
- Using `-M` flag for *masked reference* caused software to segmentation fault instead
- Needed to rebuild all libraries due to compatibility trouble

Pipeline Difficulties: Pileup and Calling

Scaling **samtools mpileup**

- Executing a pileup is a performance intensive task
- Initial test runs required 6.5 hours of CPU time with 1GB RAM
- Significant overhead with thousands of sample files, couldn't have performed this step multiple times without merging

Compatibility with **bcftools call**

- Produced only standard header information and no data as the pileup task had not included an appropriate reference sequence
- Using `-M` flag for *masked reference* caused software to segmentation fault instead
- Needed to rebuild all libraries due to compatibility trouble

Pipeline Difficulties: Merging

Documentation for **samtools merge**

- Filed pull request to document feature allowing a file of filenames to be provided as an input instead of listing on command line

Memory Leaks in **samtools merge**

- Merge jobs repeatedly killed for excessive memory use by LSF
- Discovered several memory leaks whose severity increased proportionally to the number of input files – fixed by author
- During testing I tracked down and patched several memory leaks in both the **merge** and **split** testing harnesses using **valgrind**
- Merge jobs then repeatedly killed for exceeding time limits

Pipeline Difficulties: Merging

Documentation for **samtools merge**

- Filed pull request to document feature allowing a file of filenames to be provided as an input instead of listing on command line

Memory Leaks in **samtools merge**

- Merge jobs repeatedly killed for excessive memory use by LSF
- Discovered several memory leaks whose severity increased proportionally to the number of input files – fixed by author
- During testing I tracked down and patched several memory leaks in both the **merge** and **split** testing harnesses using **valgrind**
- Merge jobs then repeatedly killed for exceeding time limits

Pipeline Difficulties: Merging

Time Sinks in **samtools merge**

- Experimented with **callgrind** to search for heavily used functions or particularly costly calls
- Found many expensive calls to **zlib** – an open source compression library – when compressing the output
- Turned out to be proportionally insignificant, using uncompressed output still led to long execution times
- Tried **gprof** to look at actual execution time rather than CPU instruction count, found 50% of execution time was spent searching for tags in data structures
- Number of files causes non-linear increase in time, now currently believe implementation of header parsing is very inefficient and just cannot scale in current form

Conclusion

Critical Evaluation

- **Frontier** greatly assisted the analysis conducted in Part I
- Happy with choice of Python and **scikit-learn**, project benefits from mutual use of **NumPy** containers and functions
- Many interesting lines of questioning introduced in Part I results
- Demonstrated in brief that decision trees generated exhibit similar behaviour to the currently existing **auto_qc** system
- Encouraging progress in both Part I and II but ultimately cut short due to unexpected difficulties with the pipeline components

Conclusion

In summary this project...

- Introduced **Frontier**, a Python package providing users with interfaces for reading, storage and retrieval of large machine learning data sets
- Used **Frontier** and **scikit-learn** to conduct preliminary analysis as to whether behaviour of the current QC system could be recovered via machine learning
- Identified parameters (**TOP9**) that contribute to accurate classification and showed ignoring warnings reduces noise
- Created **Goldilocks** for locating an appropriate genomic region for use in Part II analysis
- Outlined a pipeline for processing thousands of samples
- Produced contributions to widely used bioinformatics tools

Conclusion

Future Plans

- Complete the assembly of the analysis pipeline (most likely requiring substantial additions to **samtools merge**)
- Use concordance results from Part II to inform new lines of inquiry where Part I left off
- Continue development of **Frontier** and submit to **PyPi**
- Explore use of other machine learning algorithms and frameworks and their application to the task of quality control classification
- Investigate post-pruning algorithms and ensemble methods as decision trees cannot promise optimality

Conclusion

Questions?