P89 RNA-seq CAR transcript analysis

James Eddy

March 30, 2016

Summary

I compared **coverage profiles** and **estimated abundances** for *CAR* and related transcripts using three sets of samples:

- **P89 bulk:** sorted CD8+ T-cells, sequenced as bulk populations (expected to express *CAR*)
- **P89 single-cell:** CD8+ T-cells, captured using **C1** instrument and sequenced invididually (expected to express *CAR*)
- **P85 single-cell:** MAIT-cells, captured using **C1** and sequenced individually (not expected to express *CAR*)

Based on read coverage across the *CAR* transcript, the construct appears to be (i) expressed in bulk samples from P89; (ii) not expressed in single-cell samples from P85; (iii) expressed in a small fraction of single-cell samples from P89.

Pipeline overview

The *CAR* detection pipeline utilizes two tools: `Salmon` and `RapMap`. Both use the concept of "lightweight" / "quasi-" / "pseudo" alignment to rapidly map RNA-seq reads to the transcriptome. `Salmon` is primarily geared towards transcript *quantification*, using a probabilistic algorithm to estimate abundance (i.e., counts or TPM) for each reference transcript. `RapMap` provides a stand-alone version of the quasi-mapper used under the hood by `Salmon`. The output of `RapMap` is a SAM-like record of aligned reads, which enables further inspection of transcript coverage. These tools were chosen because they're extremely fast — ~2-5 minutes to map/quantify an RNA-seq library with ~5-10 million reads — allowing for rapid prototyping and testing.

For each library, reads are mapped to a modified human reference transcriptome, including all annotated transcripts from the hg38 gene model GTF **plus** the *CAR* transcript sequence.

Data used for analysis

Data presented below was generated and/or compiled from several sources:

- **`sample_metrics_data`:** sample annotation (e.g., donor ID, timepoint, etc.) as well sequencing & alignment metrics from RNA-seq processing

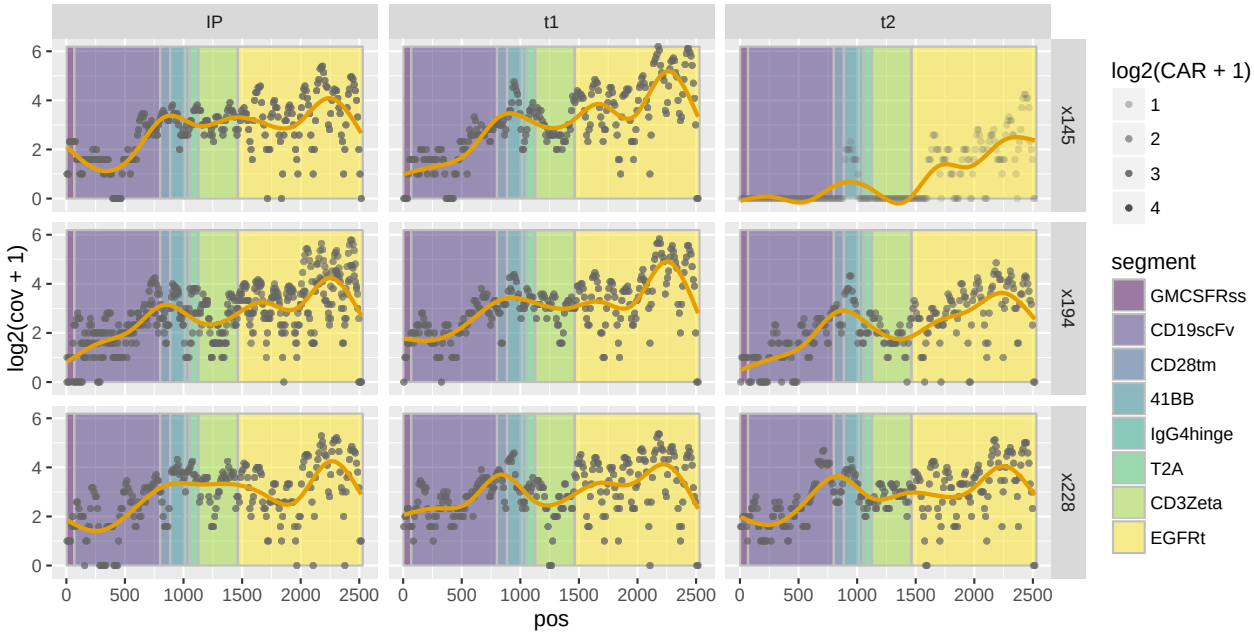
- **`sample_rapmap_data`:** read coverage measured across the length of the *CAR* transcript, based on mapping with the `RapMap` tool
- **`sample_salmon_data`:** abundance estimates (e.g., TPM, count) produced by the `Salmon` tool for *CAR* and several relevant transcripts when mapping to a modified human reference transcriptome (hg38)
- **`salmon_imgt_data`:** predicted/identified TCR junction sequences and alleles in single-cell libraries, as produced by assembly with `Trinity` followed by matching with `IMGT High V-QUEST`
- **`gff_file`:** custom-built GTF file describing where individual segments are located along the length of the *CAR* transcript

Results

Visualizing *CAR* coverage & abundance

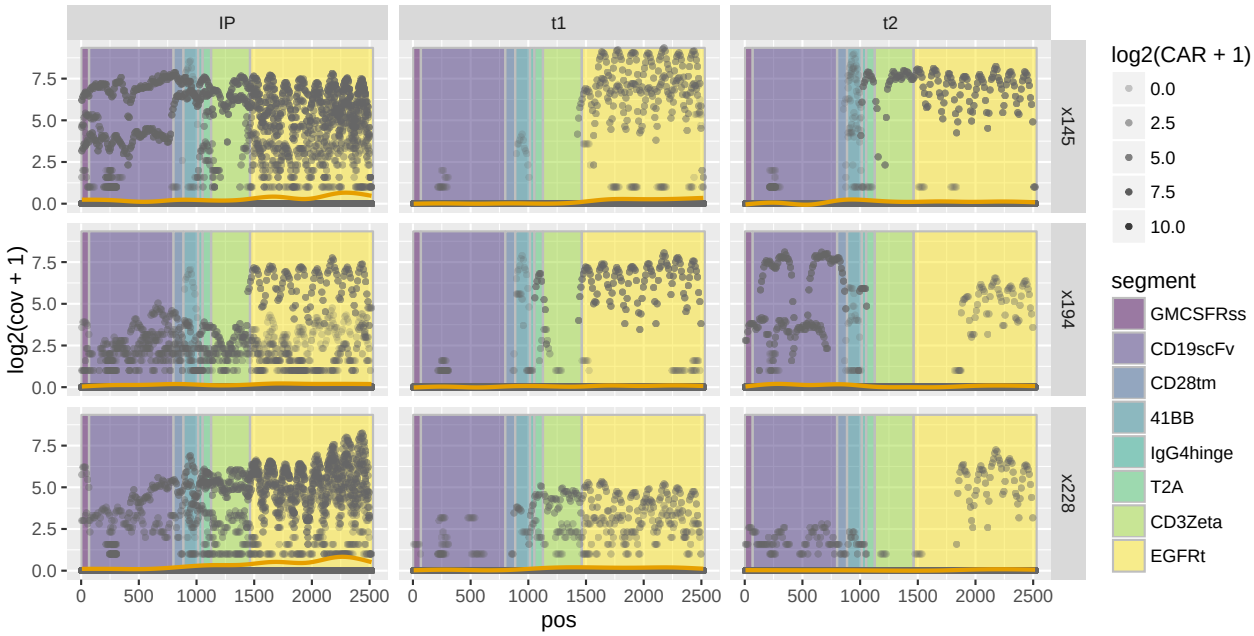
The plots below show read coverage from `RapMap` mapping across the length of the *CAR* sequence. Segments in the transcript, corresponding to the gene parts used to build the construct, are depicted by colored boxes in each plot. Transparency (i.e., alpha) is scaled based on the estimated abundance of the *CAR* transcript (TPM) as measured by `Salmon`.

Bulk libraries from project P89 While expressed at what would be considered low levels, the *CAR* transcript appears to be present in all donors at all timepoints. Note that for all but the **x194~IP** timepoint (2 replicates), each plot shows data from a single library.



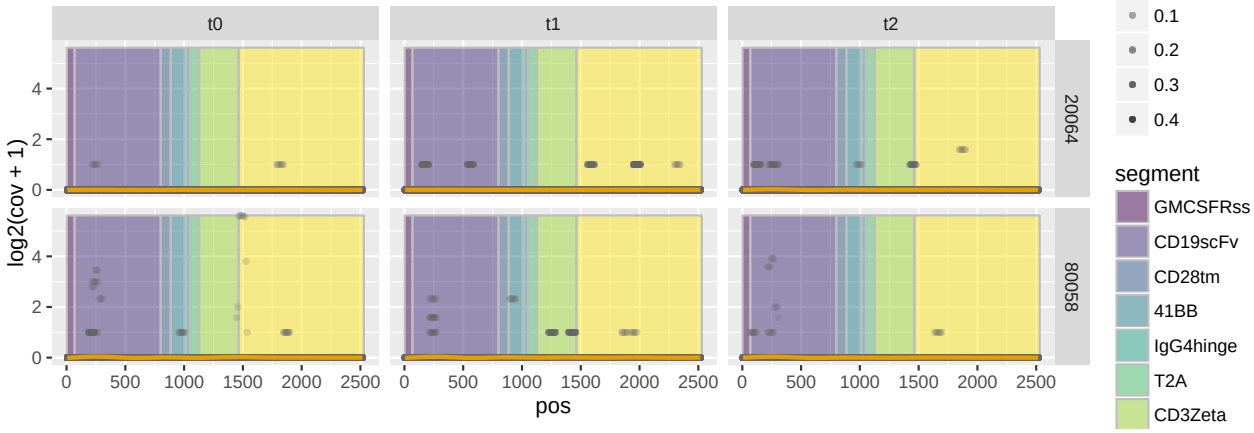
Single-cell libraries from project P89 In contrast, some single-cell libraries clearly show signs of *CAR* expression, but most do not. Each plot in this case shows data from all cells from a given donor/timepoint combination. The orange fit line depicts the average across all cells. The table below shows the breakdown of libraries (cells) per group.

donor_id	timepoint	num_libs
x145	IP	79
x145	t1	56
x145	t2	60
x194	IP	71
x194	t1	74
x194	t2	57
x228	IP	58
x228	t1	35
x228	t2	69



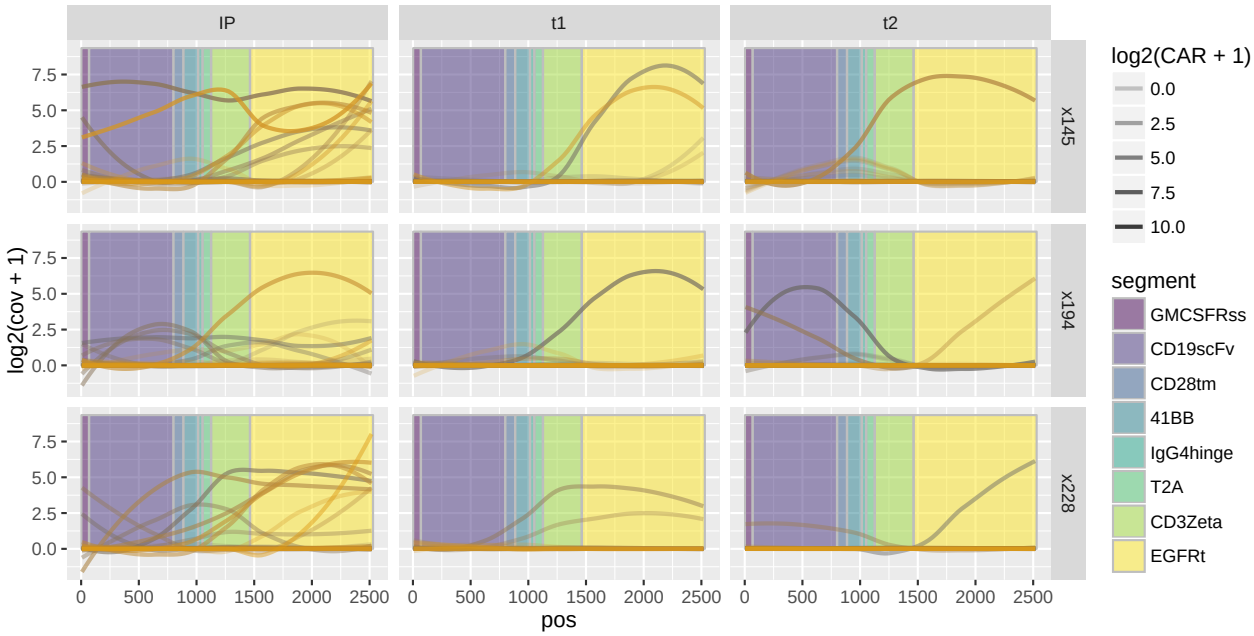
Single-cell libraries from project P85 As expected for the negative control, the MAIT cells from P85 show virtually no evidence of *CAR* expression.

donor_id	timepoint	num_libs
x145	IP	79
x145	t1	56
x145	t2	60
x194	IP	71
x194	t1	74
x194	t2	57
x228	IP	58
x228	t1	35
x228	t2	69



Inspecting trends in *CAR* coverage among single-cell P89 libs

To simplify the plots (and make it easier to distinguish between libraries), I'll just plot the fitted line of coverage for each library.

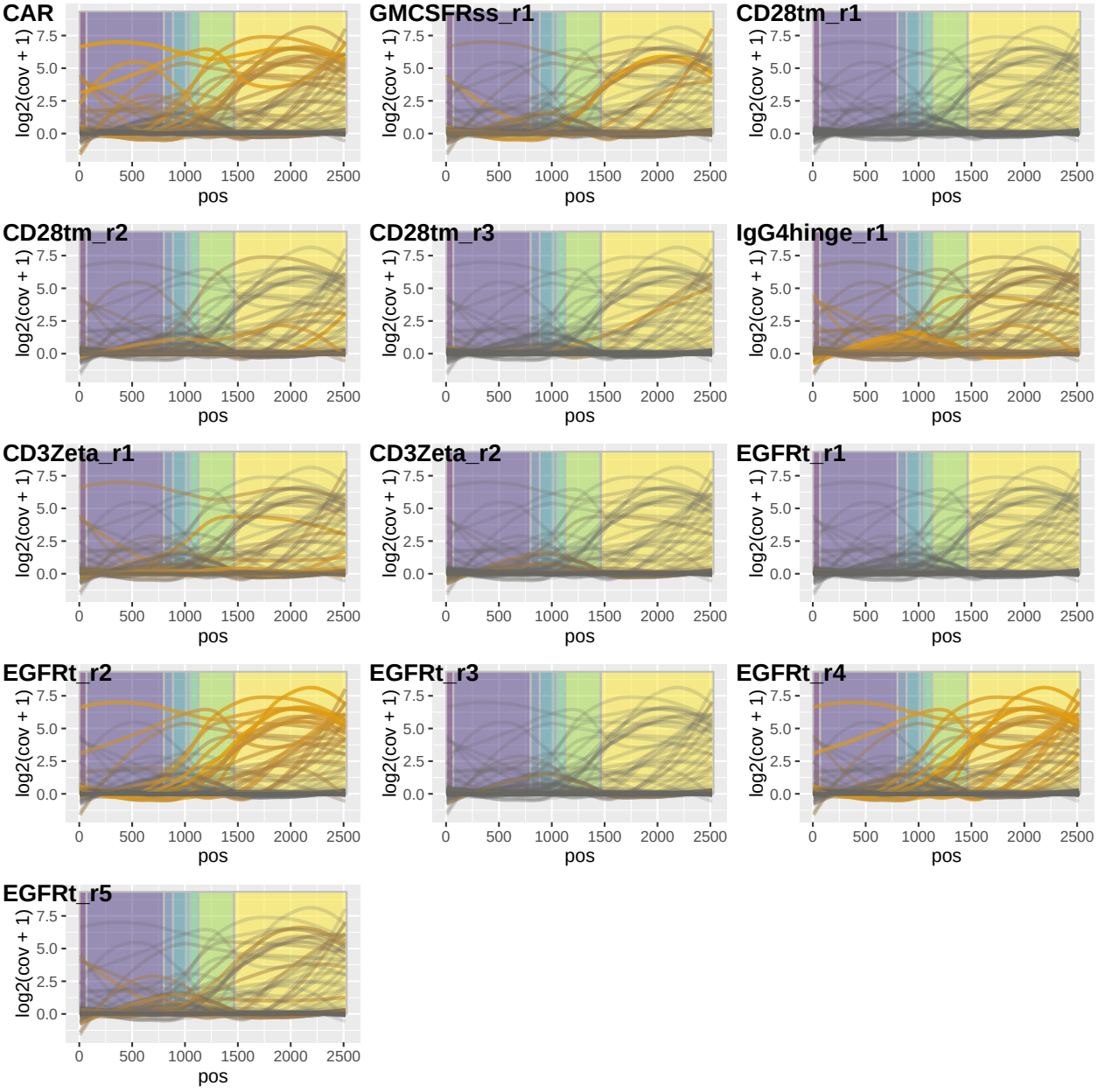


Inspecting abundance of non-*CAR* transcripts

The following human transcripts (which overlap the *CAR* sequence) were quantified by `Salmon`.

xcript_name	segment_version
CSF2	GMCSFRss_r1
CD28	CD28tm_r1
CD28	CD28tm_r2
CD28	CD28tm_r3
TNFRSF9	IgG4hinge_r1
CD247	CD3Zeta_r1
CD247	CD3Zeta_r2
EGFR	EGFRt_r1
EGFR	EGFRt_r2
EGFR	EGFRt_r3
EGFR	EGFRt_r4
EGFR	EGFRt_r5

Each plot shows *CAR* coverage across libraries, but colored based on the estimated abundance of the respective transcript.

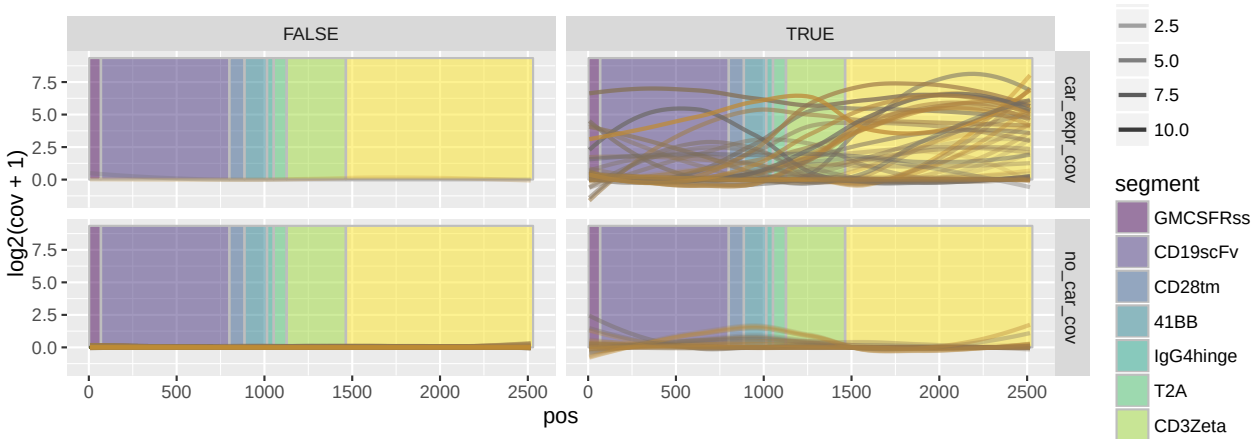


Attempting to define *CAR* detection rules

I tried to come up with a relatively simple way to classify whether *CAR* was detected in a particular library.

Coverage-based rule

> **expressed:** $\geq$ 10 positions with $>$ 0 reads in **ANY** of *CD19scFv*, *T2A*, or *EGFRt*

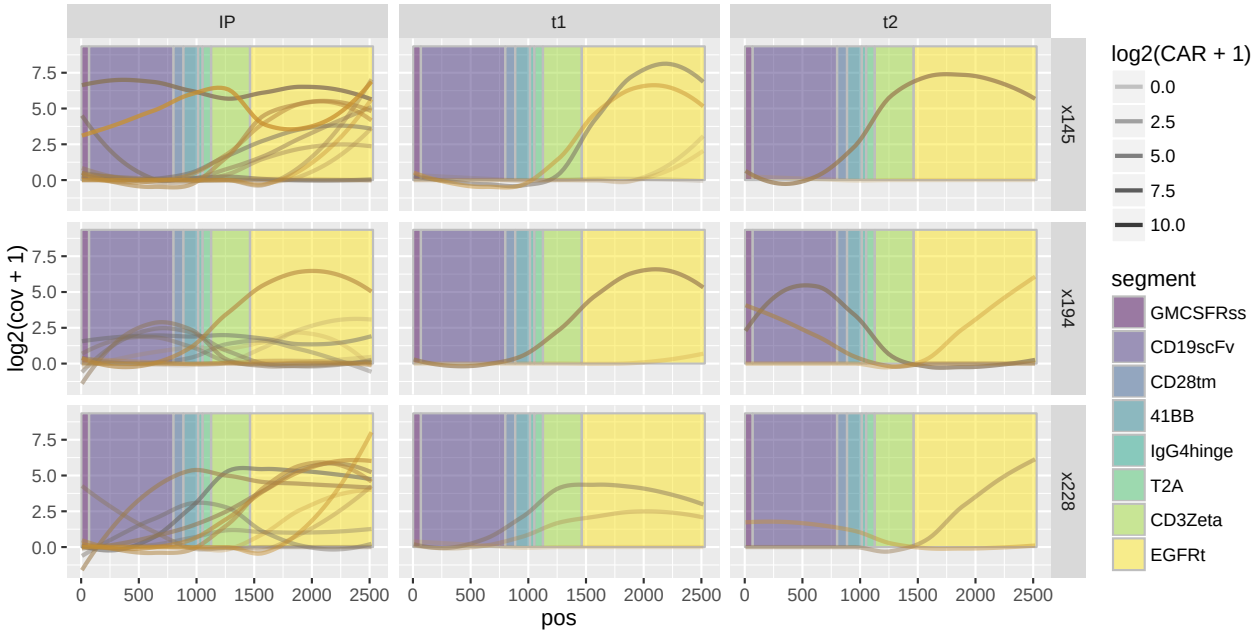


car_expr_cov	nz_cov	n_libs
car_expr_cov	FALSE	2
car_expr_cov	TRUE	53
no_car_cov	FALSE	460
no_car_cov	TRUE	39
NA	NA	5

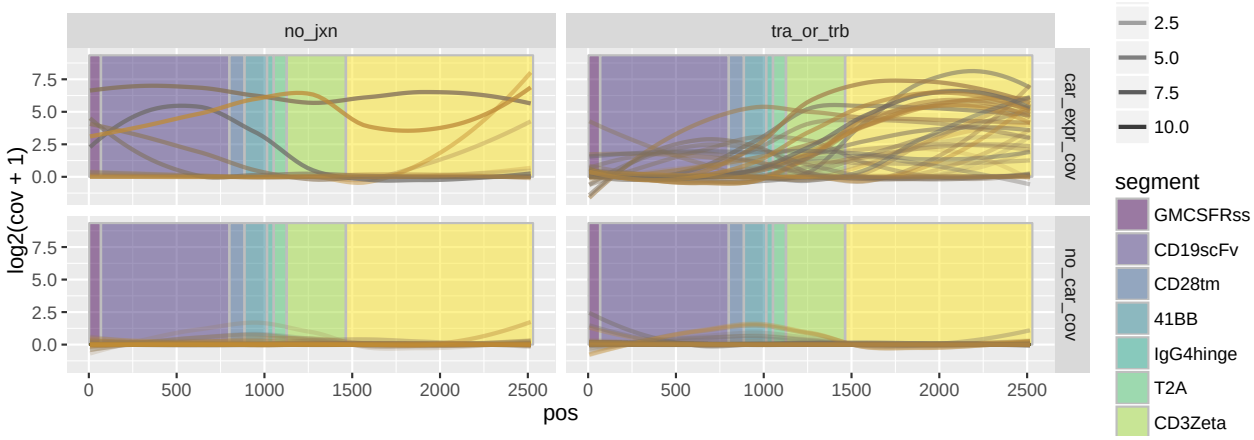
Revisiting trends

Plotting the remaining 53 libraries with *CAR* detected based on **coverage**.

Timepoint vs. donor



Functional junction detection vs. *CAR* detection



Methods

Data preprocessing

Prior to any computation with `Salmon` and `RapMap`, several steps were needed to prepare reference data and indexes:

Formatting/building reference transcriptome To convert the *CAR* sequence information from lines in a Word document to something more usable (in this case, FASTA), I wrote a script to do most of the work for me. Writing the script took a bit of extra time, but it was useful for minimizing human error and replicated effort.

```
python scripts/format_fasta.py \
    data/sequence/carGeneRaw.txt \ # unformatted sequences, copied from Word doc into text file
    CAR \ # name of gene/transcript
    data/sequence/carTranscript.fasta \ # output FASTA file
    True \ # merge individual segments into a single FASTA record
    True # output is transcript (don't add artificial 'intron' buffer between segments)
```

To convert transcript records in the hg38 gene model GTF to the required FASTA format, I used the `gffread` function included with `cufflinks`.

```
gffread \
    -w data/sequence/hg38_transcripts.fa \ # output transcriptome FASTA
    -g genome.fa \ # reference genome (hg38) from iGenomes
    genes.gtf # reference gene models for hg38 from iGenomes
```

Finally, I simply pasted the *CAR* sequence to the top of the hg38 transcriptome FASTA.

```
cat data/sequence/carTranscript.fasta data/sequence/hg38_transcripts.fa > hg38_CAR_transcripts.fa
```

Formatting/building reference gene model GTF For the purposes of visualization and some filtering tasks, I created a pseudo-GTF file with a record for each segment of the *CAR* transcript. Each segment (e.g., "CD28tm") is labeled as a unique 'transcript', and the chromosome is denoted as *CAR-1*. The start and end position of each segment is also included. I was able to use the `format_fasta.py` script for this, keeping the default option to not merge segments.

```
python scripts/format_fasta.py \
    data/sequence/carGeneRaw.txt \ # unformatted sequences, copied from Word document into text file
    CAR \ # name of gene/transcript
    data/sequence/carGeneParts.fasta \ # output FASTA file
```

Using a bit of digging and manual editing, I also added entries for each transcript in the reference human transcriptome that might overlap with *CAR* segments. In this case, the transcript ID (e.g., "NM_001243077") is used as the chromosome name, the "common" gene name (e.g., "CD28") is used as the gene ID, and the corresponding *CAR* segment along with a tag indicating record number, if multiple isoforms exist for the gene (e.g., "CD28tm_r1").

```
python scripts/gene_fasta_to_gtf.py \
    data/sequence/carGeneParts.fasta \
    CAR \
    data/annotation/carGeneParts.gtf
```

Building indexes The commands for building indexes with `RapMap` and `Salmon`. Note: `docker` was used to run `RapMap`, as the tool is not currently available for Mac OS.

```
docker run --rm -v ${PWD}/data:/home/data jaeddy/rapmap:0.1.0-pre \ # loading docker image
    rapmap quasiindex \ # command to use rapmap executable & quasiindex module
    -t data/sequence/hg38_CAR_transcripts.fa \ # input reference transcriptome
    -i data/indexes/rapmap/hg38_CAR \ # index folder/prefix
    -k 19 #
```

```
tools/SalmonBeta-0.5.0_OSX-10.10/bin/salmon index # salmon executable & index module \
    -t data/sequence/hg38_CAR_transcripts.fa \
    -i data/indexes/salmon/hg38_CAR \
    --type quasi \
    -k 19
```

Mapping & quantification

Both index building and actual mapping/quantification are included in the script `car_detect_pipe.sh`, which runs the pipeline an all library FASTQ files specified in a tab-delimited input list. The `bash` code is a bit messy, so I won't include it here, but the `RapMap` step proceeds as follows:

1. Map reads to reference transcriptome, save output SAM file
2. Convert SAM to BAM, sort, and index with `samtools`
3. Filter BAM records to only include reads mapping to *CAR* or relevant endogeneous transcripts (using `samtools`)
4. Index again with `samtools`)

For the `Salmon` step, outputs are saved as-is.

Session info

```
## R version 3.2.1 (2015-06-18)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.11.4 (unknown)
```

```
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] cowplot_0.6.1        scales_0.4.0         ggthemes_3.0.2
##  [4] rtracklayer_1.30.4   GenomicRanges_1.22.4 GenomeInfoDb_1.6.3
##  [7] IRanges_2.4.8        S4Vectors_0.8.11     BiocGenerics_0.16.1
## [10] viridis_0.3.4        ggplot2_2.1.0        dplyr_0.4.3
## [13] tidyr_0.4.1          stringr_1.0.0        knitr_1.12.3
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.4                highr_0.5.1
##  [3] futile.logger_1.4.1        formatR_1.3
##  [5] plyr_1.8.3                 XVector_0.10.0
##  [7] futile.options_1.0.0       bitops_1.0-6
##  [9] tools_3.2.1                zlibbioc_1.16.0
## [11] digest_0.6.9               lattice_0.20-33
## [13] nlme_3.1-126               evaluate_0.8.3
## [15] gtable_0.2.0               mgcv_1.8-12
## [17] Matrix_1.2-4               DBI_0.3.1
## [19] yaml_2.1.13                gridExtra_2.2.1
## [21] Biostrings_2.38.4          grid_3.2.1
## [23] Biobase_2.30.0             R6_2.1.2
## [25] XML_3.98-1.4               BiocParallel_1.4.3
## [27] rmarkdown_0.9.5            reshape2_1.4.1
## [29] lambda.r_1.1.7             magrittr_1.5
## [31] codetools_0.2-14           GenomicAlignments_1.6.3
## [33] Rsamtools_1.22.0           htmltools_0.3.5
## [35] SummarizedExperiment_1.0.2 assertthat_0.1
## [37] colorspace_1.2-6           labeling_0.3
## [39] stringi_1.0-1              lazyeval_0.1.10
## [41] RCurl_1.95-4.8             munsell_0.4.3
```