Alabama Roads Project

Background

This is a 'proof of concept' exercise that will attempt the predict road safety using the Google streetview api and a retrained (transfer learning based) version of Tensorflow to enable mass prediction of road safety on large datasets. The ultimate output would be a heat map/scoring of predicted road saftey across Alabama.

Step 1: Take streetview photos

Import roads

To do this we will use the road shapefile in the raw project data folder ~/Dropbox/pkg.data/alabama_roads/raw/us82erik/us82_dissolve. If you do not currently have access to the folder, you can download the folder by going to this Dropbox web link. Ideally, you should be able to dynamically link locally by using Dropbox desktop. . . r

To build the streetview folders, the first step is to discretize the road shapefile and then use the google streetview metadata api to get locations of nearest google streetview picture (henceforth called pano_id). In this case the picture will be manually tuned to have the car driving in the 'correct' direction. In the future, this can easily be done with tensorflow as well, by simply ensuring that the photo taken is on the right (not the left) side of the road. Again, this can be easily automated in tensorflow in the future for bigger data applications.

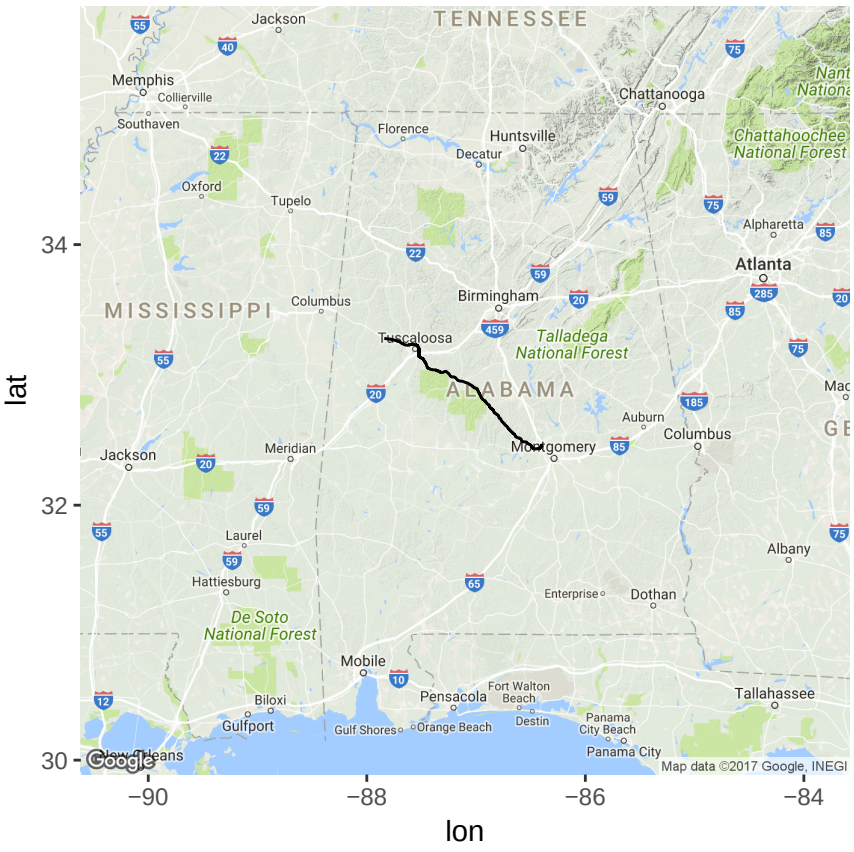
```
if (!(file.exists(roads.clean.location))) {
    roads <- rgdal::readOGR(dsn = path.expand(roads.raw.location),
        layer = "us82d", verbose = FALSE)
    roads <- sp::spTransform(roads, CRSobj = CRS(proj.env))
    saveRDS(roads, file = roads.clean.location)
} else {
    roads <- readRDS(roads.clean.location)
}
# Plot
map.center <- as.numeric(geosphere::centroid(rgeos::gBuffer(roads,
    width = 0.1)))
```

```
## Warning in rgeos::gBuffer(roads, width = 0.1): Spatial object is not
## projected; GEOS expects planar coordinates
```

```
map <- ggmap::get_googlemap(center = c(lon = map.center[1],
    lat = map.center[2]), zoom = 7)
```

```
## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=32.890998,-87.106853&zoom=7&size=
```

```
roads_fortify <- ggplot2::fortify(roads)
p <- ggmap(map) + geom_line(data = roads_fortify,
    aes(x = long, y = lat, group = group))
print(p)
```



Discretize roads to points

This discretizes the roads into points creates the points to find the panoids. The goal of the road discretization is solely to build a set of points with which to find the nearest pano_id.

```
if (!(file.exists(road.points.location))) {
    road.points <- suppressWarnings(sample.line(roads,
        sdist = 0.0075))  ##
    road.points$long <- road.points@coords[, 1]
    road.points$lat <- road.points@coords[, 2]
    saveRDS(road.points, file = road.points.location)
} else {
    road.points <- readRDS(road.points.location)
}
knitr::kable(head(road.points@data))
```

ID	long	lat
1	-87.84014	33.28436
2	-87.84006	33.28435
3	-87.83999	33.28434
4	-87.83991	33.28434
5	-87.83984	33.28433
6	-87.83976	33.28433

Points to panoids with camera bearings

```
if (!(file.exists(panoids.location))) {
    points.ids.full <- road.points$ID
    points.files <- list.files(road.points.panoid.location)
    points.ids.done <- sapply(points.files, function(x) stringr::str_extract(x,
        stringr::regex("(?<=id\\:).+(?=\\;)",
            perl = TRUE)))
    points.ids.not.done <- points.ids.full[!(points.ids.full %in%
        points.ids.done)]

    while (length(points.ids.not.done) > 0) {
        if (length(points.ids.not.done) > 1) {
            points.ids <- sample(points.ids.not.done,
                10)
        } else {
            points.ids <- points.ids.not.done
        }
        cat("Assigning", length(points.ids), "points to panoids.",
            length(points.ids.not.done), "remain \n")
        l.panoids <- lapply(points.ids, function(x) streetview.metadata(data.table::as.data.table(road.
            x), ]), api_key = api.key, save.location = road.points.panoid.location))

        points.files <- list.files(road.points.panoid.location)
        points.ids.done <- sapply(points.files,
            function(x) stringr::str_extract(x,
                stringr::regex("(?<=id\\:).+(?=\\;)",
                  perl = TRUE)))
        points.ids.not.done <- points.ids.full[!(points.ids.full %in%
            points.ids.done)]
    }

    # Combine all ponoids into a data.table
    files.to.load <- points.files <- list.files(road.points.panoid.location,
        full.names = TRUE)
    dt.file <- readRDS(files.to.load[1])
    for (iLoad in 2:length(files.to.load)) {
        f.to.load <- files.to.load[iLoad]
        f <- readRDS(f.to.load)
        l.file <- list(dt.file, f)
        dt.file <- rbindlFist(l.file, use.names = TRUE,
            fill = TRUE)
    }
    dt_pano_ids <- dt.file
    # Unique record for each pano Id and retain
    # order from roads.point
    dt_pano_ids$roads.point.id <- as.integer(dt_pano_ids$roads.point.id)
    setkey(dt_pano_ids, roads.point.id)
    # Find unique panoids
    dt_pano_ids[, `:=`(pano_id_order, .GRP), by = pano_id]
    dt_pano_ids <- unique(dt_pano_ids[, .(date,
        location.lat, location.lng, pano_id, pano_id_order,
        status)])
```

```
    dt_pano_ids$location.lng <- as.numeric(dt_pano_ids$location.lng)
    dt_pano_ids$location.lat <- as.numeric(dt_pano_ids$location.lat)
    setkey(dt_pano_ids, pano_id_order)
    # Get bearings for pictures (ahead and behind)
    # (Simply aim at panoids in front and behind.)
    dt_pano_ids$lat.lead <- data.table::shift(dt_pano_ids$location.lat,
        1, type = "lead")
    dt_pano_ids$lng.lead <- data.table::shift(dt_pano_ids$location.lng,
        1, type = "lead")
    dt_pano_ids$lat.lag <- data.table::shift(dt_pano_ids$location.lat,
        1, type = "lag")
    dt_pano_ids$lng.lag <- data.table::shift(dt_pano_ids$location.lng,
        1, type = "lag")
    dt_pano_ids <- na.omit(dt_pano_ids)
    dt_pano_ids$bearings.lead <- sapply(1:nrow(dt_pano_ids),
        function(x) geosphere::bearing(c(dt_pano_ids[x,
            location.lng], dt_pano_ids[x, location.lat]),
            c(dt_pano_ids[x, lng.lead], dt_pano_ids[x,
                lat.lead])))
    dt_pano_ids$bearings.lag <- sapply(1:nrow(dt_pano_ids),
        function(x) geosphere::bearing(c(dt_pano_ids[x,
            location.lng], dt_pano_ids[x, location.lat]),
            c(dt_pano_ids[x, lng.lag], dt_pano_ids[x,
                lat.lag])))
    saveRDS(dt_pano_ids, file = panoids.location)
} else {
    dt_pano_ids <- readRDS(panoids.location)
}
knitr::kable(head(dt_pano_ids[, .(date, pano_id,
    pano_order = pano_id_order, lat = location.lat,
    lng = location.lng, bear.lead = bearings.lead,
    bear.lag = bearings.lag)]))
```

date	pano_id	pano_order	lat	lng	bear.lead	bear.lag
2016-05	lk4vKW941erchBtIeRqenA	2	33.28436	-87.84009	92.64210	-87.35684
2016-05	8sR6h5ILqzn3NSvXhYhP_g	3	33.28435	-87.84000	100.73244	-87.35785
2016-05	5YeX5au5BuKvr-VE8mnHYg	4	33.28434	-87.83991	98.83936	-79.26751
2016-05	7ixd4IHZEO1aUZSMVo_OcQ	5	33.28433	-87.83982	97.23829	-81.16059
2016-05	LMB6s5_qe2463PujYfHafw	6	33.28432	-87.83973	97.02418	-82.76167
2016-05	6TTjtyYJ1NmTOpE0T-_Yzg	7	33.28431	-87.83964	96.80426	-82.97577

This can be automated in the future to adjust the sdist parameter in the road.points chunk to optimally collect all panoids while minimizing the number of repeats samples. This will speed up the meta_data api calls. For now we will just fill in the data data with the 24529 road points that we have sampled. Two camera bearings are assigned for each panoid. This is based on aiming the camera at the previous and next pano_id. Order is based on the roads.points file and retained in the field **dt_pano_ids$pano_id_order**.

There are 20104 observations in the **dt_pano_ids** dataset.

Step 2: Streetview Snapshots

```
pano_samples <- sample(nrow(dt_pano_ids), size = nrow(dt_pano_ids),
    replace = FALSE)

for (iSample in pano_samples) {
    if ((iSample%%100) == 0)
        cat(iSample, "\\n")
    dt_pano_id <- dt_pano_ids[iSample, ]

    fDest.lead <- paste0(snapshots.location, "pano_id:",
        dt_pano_id$pano_id, "_lead.jpg")
    fDest.lag <- paste0(snapshots.location, "pano_id:",
        dt_pano_id$pano_id, "_lag.jpg")
    g.lat <- dt_pano_id$location.lat
    g.lng <- dt_pano_id$location.lng
    g.bearings <- unlist(dt_pano_id[, .(bearings.lead,
        bearings.lag)])

    api <- list()
    api$head <- "https://maps.googleapis.com/maps/api/streetview?size=600x400"
    api$location <- paste0("&location=", g.lat,
        ",", g.lng)
    api$fov <- paste0("&fov=", 30)
    api$heading <- paste0("&heading=", g.bearings[1])
    api$pitch <- paste0("&pitch=", 0)
    api$api.key <- paste0("&key=", api.key)
    api.url <- paste0(unlist(api), collapse = "")
    if (!(file.exists(fDest.lead))) {
        streetShot <- download.file(api.url, fDest.lead,
            quiet = TRUE)
    }
    api$heading <- paste0("&heading=", g.bearings[2])
    api.url <- paste0(unlist(api), collapse = "")
    if (!(file.exists(fDest.lag))) {
        streetShot <- download.file(api.url, fDest.lag,
            quiet = TRUE)
    }
}
snap.list <- list.files(path = snapshots.location,
    full.names = TRUE)
snap.samp <- sample(snap.list, 10)
knitr::include_graphics(snap.samp[1])
knitr::include_graphics(snap.samp[2])
```

Step 3: Human classification to use for machine training

Step 4: Machine Training

Traning data must have unique filenames.

```
training_dirs <- c('guardrail', 'rumble', 'signals')
for (training_dir in training_dirs){

filez <- list.files(paste0('~/Dropbox/pkg.data/alabama_roads/raw/training_data/' training_dir), full.nam
sapply(filez,FUN=function(eachPath){
      file.rename(from=eachPath,to=stringr::str_replace(eachPath, pattern = '\\.jpg', replacement = past
})
}
```

Now actually compile and train tensorflow.

```
cd ~/tensorflow
# First build inception for retraining (estimated time ~25 minutes on Linux box)
# Do not need to run this more than once on the box.
bazel build --config opt tensorflow/examples/image_retraining:retrain

# retrain/transfer learning on roads data (~5-10 minutes)
# need to rerun this if want to retrain.
bazel-bin/tensorflow/examples/image_retraining/retrain --image_dir /home/ebjohnson5/Dropbox/pkg.data/ala
# To visualize training output in tensorboard
tensorboard --logdir /tmp/retrain_logs
```

Step 5: Machine Classification

This is the main bottleneck in the process right now. This can be parallelized in python using tensorflow/serving https://github.com/tensorflow/serving. For now, this works but takes awhile.

```
tf_classify <- tf_classification()
```

```
# Guardrail example
# bazel-bin/tensorflow/examples/image_retraining/label_image
# --graph=/tmp/output_graph.pb
# --labels=/tmp/output_labels.txt
# --output_layer=final_result:0
# --image=$HOME/Dropbox/pkg.data/alabama_roads/raw/snapshots/pano_id:r2NkANifYabJWFCEi2CXUQ_lag.jpg

# bazel-bin/tensorflow/examples/image_retraining/label_image
# --graph=/tmp/output_graph.pb
# --labels=/tmp/output_labels.txt
# --output_layer=final_result:0
# --image=$HOME/Dropbox/pkg.data/alabama_roads/raw/snapshots/pano_id:r0ExJrw5e5qPwp7qcTU9rg_lag.jpg

# Need to add to training
# bazel-bin/tensorflow/examples/image_retraining/label_image
# --graph=/tmp/output_graph.pb
# --labels=/tmp/output_labels.txt
# --output_layer=final_result:0
# --image=$HOME/Dropbox/pkg.data/alabama_roads/raw/snapshots/pano_id:r0fW5VldKtkNNetToCeLGQ_lag.jpg

# Traffic signal
# bazel-bin/tensorflow/examples/image_retraining/label_image
# --graph=/tmp/output_graph.pb
# --labels=/tmp/output_labels.txt
# --output_layer=final_result:0
```

```
# --image=$HOME/Dropbox/pkg.data/alabama_roads/raw/snapshots/pano_id:QZkFVmowjPqOztNwXXOhdg_lead.jpg

# Null
# bazel-bin/tensorflow/examples/image_retraining/label_image
# --graph=/tmp/output_graph.pb
# --labels=/tmp/output_labels.txt
# --output_layer=final_result:0
# --image=$HOME/Dropbox/pkg.data/alabama_roads/raw/snapshots/pano_id:QZkFVmowjPqOztNwXXOhdg_lag.jpg
```

Step 6: Output visualization

This is where the QGis component goes.