Data Analysis Appendix:

"The Role of Latency and Task Complexity in Predicting Visual Search Behavior"

3/31/2019

Setup

```
library(readr)
library(dplyr)
library(tidyr)
library(pander)
library(ggplot2)
library(rpart)
library(rpart.plot)
library(caret)
library(party)
```

Load data

```
pilot_data <- read.csv("pilot_data.csv") %>%
  mutate(latency = as.factor(latency))

continuous_data <- read.csv("continuous_data.csv")
```

Initial results

We conducted 3 initial studies to investigate the relationship between latency and behavior under various conditions. In each experiment, the user is presented with a collague of images, and asked to locate a target image with a particular semantic feature. Each collage has two target images and 5 latency conditions (0ms, 2500ms, 7000ms, 10000ms, and 1400ms). One target is designated the high-latency target, and incurs additional latency as specified by the latency condition; the targets load equally fast in the 0ms latency condition.

We use Pearson's Chi-Squared test to assess whether users in different latency conditions have a higher incidence of finding the low-latency target first.

Experiment 1: Baseline

The first experiment is a basic visual search task, designed to simulate the earliest stages of a visual search task. We do not offer the participant any additional information regarding the location of the targets, and they are left to orient themselves to the dataset on their own.

```
baseline <- pilot_data %>%
  filter(condition == "Experiment1")

chisq_baseline <- chisq.test(baseline$foundFastTargetFirst,
                             baseline$latency,
                             correct = FALSE)
```

```
pander(chisq_baseline)
```

Table 1: Pearson's Chi-squared test: `baseline$foundFastTargetFirst` and `baseline$latency` In this experiment, we did not observe a statistically-significant deviation from the expected values:

Test statistic	df	P value
2.373	4	0.6675

```
# Observed values
pander(chisq_baseline$observed)
```

	0	2500	7000	10000	14000
FALSE	8	9	6	7	5
TRUE	11	12	14	13	16

```
# Expected values
pander(chisq_baseline$expected)
```

	0	2500	7000	10000	14000
FALSE	6.584	7.277	6.931	6.931	7.277
TRUE	12.42	13.72	13.07	13.07	13.72

and so we conclude that the incidence of finding the fast target first does not vary with latency in the baseline condition.

Experiment 2: Search-Space Reduction

The second experiment is a slightly modified version of the **baseline** visual search task. In this experiment, the participant is given information regarding the general position of the target (to the left or to the right), effectively reducing the search space by half. This is designed to simulate a slightly later stage in the task, in which the user has begun to narrow down their search area and focus in on a particular subregion of the data.

```
search_space_reduction <- pilot_data %>%
  filter(condition == "Experiment2")

chisq_search_space_reduction <- chisq.test(search_space_reduction$foundFastTargetFirst,
                                           search_space_reduction$latency,
                                           correct = FALSE)

pander(chisq_search_space_reduction)
```

Table 4: Pearson's Chi-squared test: `search_space_reduction$foundFastTargetFirst` and `search_space_reduction$latency` In this experiment, we again did not observe a statistically-significant deviation from the expected values:

Test statistic	df	P value
3.055	4	0.5487

```
# Observed values
pander(chisq_search_space_reduction$observed)
```

	0	2500	7000	10000	14000
FALSE	10	6	7	6	8
TRUE	10	16	15	14	12

```
# Expected values
pander(chisq_search_space_reduction$expected)
```

	0	2500	7000	10000	14000
FALSE	7.115	7.827	7.827	7.115	7.115
TRUE	12.88	14.17	14.17	12.88	12.88

Experiment 3: Proposed Locations

The third experiment is a further modification to the visual search task. In this experiment, the participant is given specific information regarding two proposed locations for the target. This experiment simulates a later stage in the search task, whereing the user has generated a set of hypotheses about their target, and transitions to the task of verifying or refuting those specific locations.

```
proposed_locations <- pilot_data %>%
  filter(condition == "Experiment3")

chisq_proposed_locations <- chisq.test(proposed_locations$foundFastTargetFirst,
                                       proposed_locations$latency,
                                       correct = FALSE)

pander(chisq_proposed_locations)
```

Table 7: Pearson's Chi-squared test: `proposed_locations$foundFastTargetFirst` and `proposed_locations$latency` In this experiment, we **did** observe a statistically-significant deviation from the expected values:

Test statistic	df	P value
15.63	4	0.003554 * *

```
# Observed values
pander(chisq_proposed_locations$observed)
```

	0	2500	7000	10000	14000
FALSE	11	8	4	4	1
TRUE	10	12	19	18	19

```
# Expected values
pander(chisq_proposed_locations$expected)
```

	0	2500	7000	10000	14000
FALSE	5.547	5.283	6.075	5.811	5.283
TRUE	15.45	14.72	16.92	16.19	14.72

This is an interesting result, because it captures the widespread intuition that when latency is present, participants will tend to avoid exploring the high-latency area (and therefore finding the high-latency target). But why should this effect be present only when the user is further along in their search process, and why only at such high latencies? What is the "tipping point" at which latency starts to have an effect? Does this effect persist under more realistic conditions, such as when the user has some global knowledge of the data landscape?

Continuous latency results

In order to further investigate the relationship between latency and search behavior, we conducted a second round experiments in which latency is treated as a continuous variable rather than an ordered factor, and introduced a fourth condition in which the user is presented with an interface more closely resembling a real-world visual search environment.

Experiment 3.2: Proposed Locations with Uniformly-Drawn Latency

In this experiment, we revisit Experiment 3.1 in which the participant is given specific information regarding two proposed locations for the target. Rather than being randomly assigned to one of 5 latency conditions, we randomly select a maximum latency value between 0ms and 14000ms. By drawing unformly from a continuous range of possible maximum latency values, we are then able to investigate how the probability of finding the low-latency target first varies with latency using logistic regression.

```
proposed_locations_continuous <- continuous_data  %>%
  filter(condition == "Experiment3_2")  %>%
  mutate(success = ifelse(foundFastTargetFirst == TRUE, 1, 0)) %>%
  replace(., is.na(.), "not reported") %>%
  drop_na() %>%
  select(-userid, -condition, -foundFastTargetFirst)
```

Check that we're not talking about a "rare event" (i.e. <15% of our data), which would mean we'd have to be extra careful with our modeling:

```
prop.table(table(proposed_locations_continuous$success))
```

```
##
```

```
##         0         1
## 0.3068182 0.6931818
```

Good to go: roughly 30.7% of trials ended in the user finding the high-latency target first, and the remaining 69.3% ended in the user finding the low-latency target first.

Logistic Regression

```
logistic_model_proposed_locations <- glm(success ~ latency,
                          family = binomial(link = 'logit'),
                          data = proposed_locations_continuous)

summary(logistic_model_proposed_locations)
```

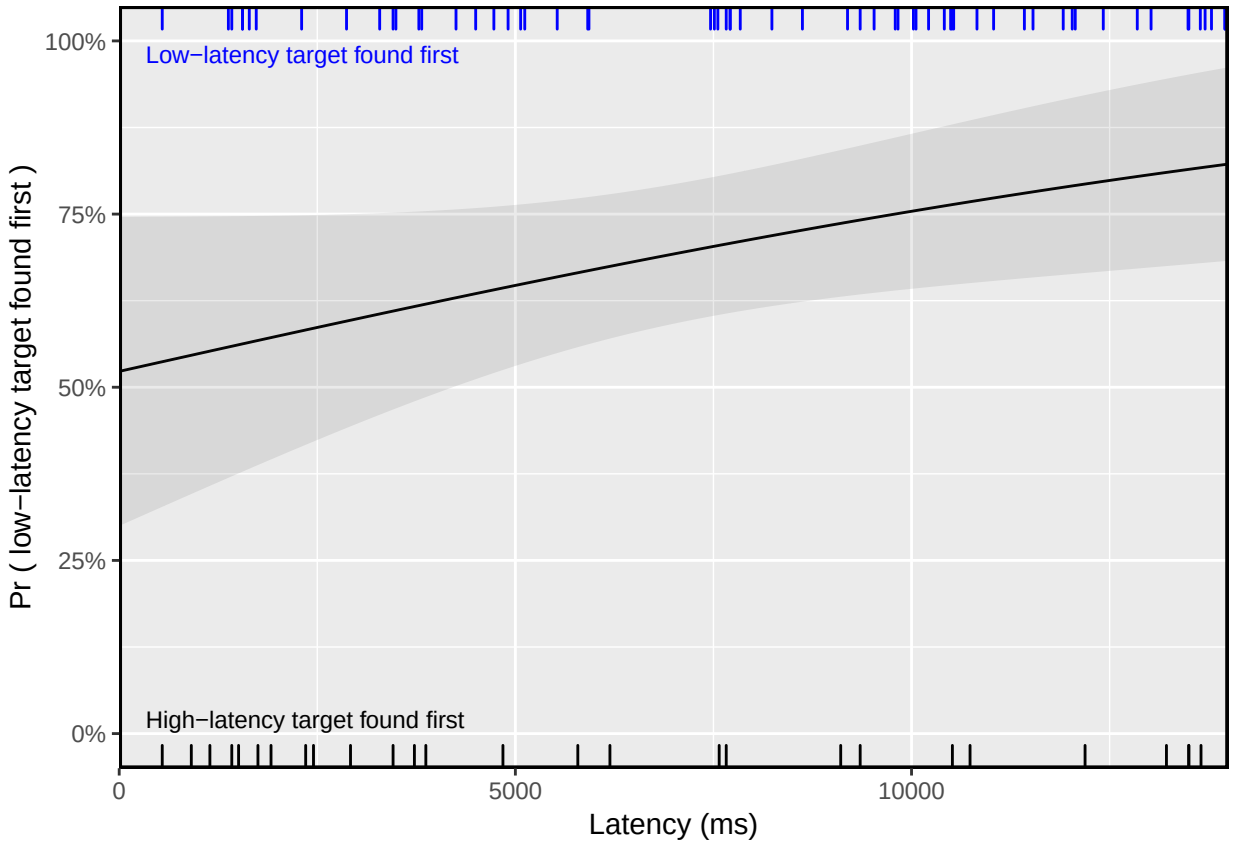
```
##
## Call:
## glm(formula = success ~ latency, family = binomial(link = "logit"),
##     data = proposed_locations_continuous)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8429  -1.2865   0.7289   0.8976   1.1154
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) 9.170e-02  4.471e-01   0.205   0.8375
## latency     1.029e-04  5.681e-05   1.810   0.0702 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 108.51  on 87  degrees of freedom
## Residual deviance: 105.10  on 86  degrees of freedom
## AIC: 109.1
##
## Number of Fisher Scoring iterations: 4
```

This relationship is significant only at the level of $p = 0.1$, which is below the generally-accepted threshold for statistical significance. When we view these results graphically, we observe the same weak effect:

```
test_data <- data.frame(latency = sample(c(0:14000), size = 1000)) %>%
  mutate(y_hat = predict(logistic_model_proposed_locations,
                                      newdata = .,
                                      type = 'response'),
         wrong_se = predict(logistic_model_proposed_locations,
                                          newdata = .,
                                          type = 'response',
                                          se.fit = TRUE)$se.fit,
         wrong_upr = y_hat + (2 * wrong_se),
                    wrong_lwr = y_hat - (2 * wrong_se))

proposed_locations_continuous_low_first <- proposed_locations_continuous %>%
  filter(success == 1)
```

```r
proposed_locations_continuous_high_first <- proposed_locations_continuous %>%
  filter(success == 0)

## plot it
ggplot(test_data, aes(x = latency, y = y_hat)) +
    geom_line() +
    # Add standard error band
    geom_ribbon(data = test_data, aes(ymin = wrong_lwr, ymax = wrong_upr),
                alpha = 0.1) +
    # Add rug plot above to denote observed low-latency-first cases
    geom_rug(aes(y = success, x = latency),
             data = proposed_locations_continuous_low_first,
             sides = "t",
             color = "blue") +
    # Add rug plot below to denote observed low-latency-first cases
    geom_rug(aes(y = success, x = latency),
             data = proposed_locations_continuous_high_first,
             sides = "b") +

    xlab("Latency (ms)") +
    ylab("Pr ( low-latency target found first )") +
    xlim(0,14000) +
    scale_y_continuous(labels = scales::percent) +
    scale_x_continuous(expand=c(0,0))+
    theme(panel.border = element_rect(colour = "black", fill=NA, size=1))+
    annotate("text", x = 0, y = 0.02,
             label = "    High-latency target found first",
             hjust = 0, size = 3) +
    annotate("text", x = 0, y = 0.98,
             label = "    Low-latency target found first",
             hjust = 0, size = 3, color = "blue")
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which
## will replace the existing scale.
```



In the absence of additional latency, there is nothing to differentiate the nominal `high-latency'' target from the nominal`low-latency" target, and so they appear as the first target found with roughly equal probability. As latency increases, we observe an increase in the probability of finding the low-latency target first. However, the standard error bands remind us that this effect is somewhat weak, suggesting that latency may not be the only factor at play. Let's consider an alternative modelling approach.

Simple recursive partition

```
set.seed(1)
inTrain <- createDataPartition(
  y = proposed_locations_continuous$success,
  p = .7,
  list = FALSE
)

proposed_locations_continuous_train = proposed_locations_continuous[inTrain,]
proposed_locations_continuous_test = proposed_locations_continuous[-inTrain,]
```

```
set.seed(1)
model <- train(factor(success,
                      levels = c(0,1),
                      labels = c("High.Latency.Target",
                                 "Low.Latency.Target")) ~ . ,
               data = proposed_locations_continuous_train,
               method='rpart',
               tuneLength=10,
```

```
              trControl=trainControl(method = 'cv',
                                     number = 10,
                                     classProbs = TRUE,
                                     summaryFunction = twoClassSummary))
```

```
## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy" was
## not in the result set. ROC will be used instead.
```

```
model
```

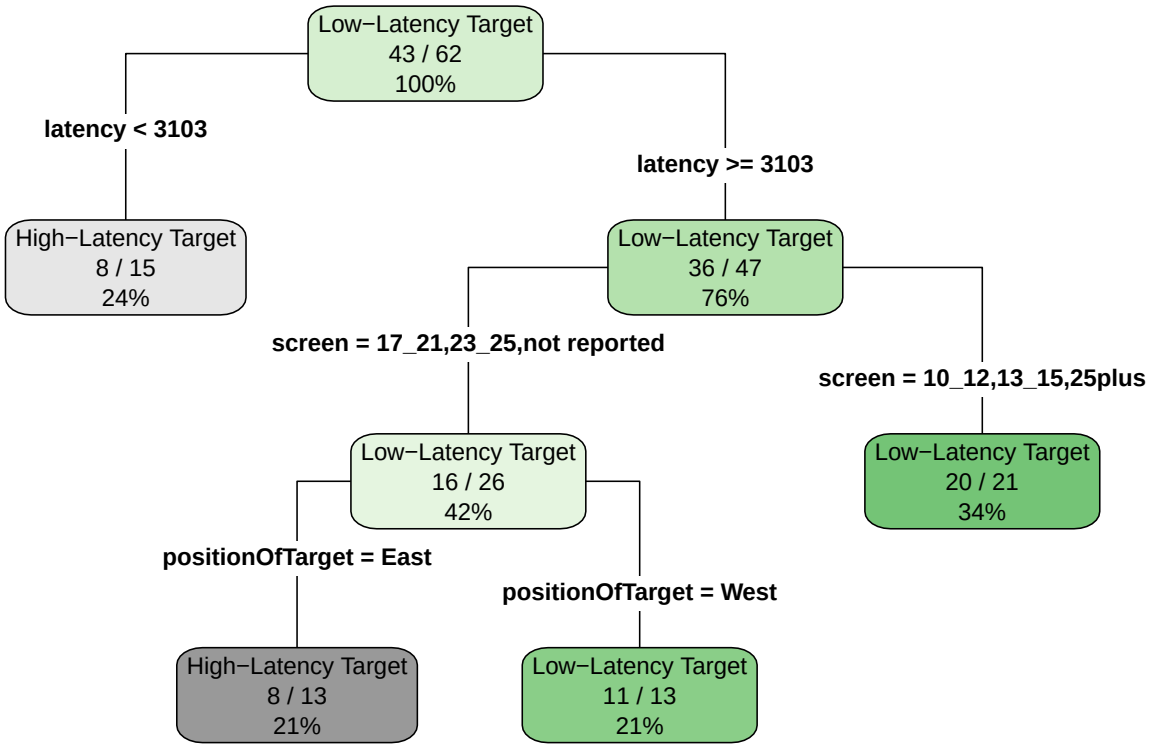
```
## CART
##
## 62 samples
## 13 predictors
##  2 classes: 'High.Latency.Target', 'Low.Latency.Target'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 55, 56, 56, 56, 55, 56, ...
## Resampling results across tuning parameters:
##
##   cp           ROC      Sens  Spec
##   0.000000000  0.37625  0.3   0.565
##   0.005847953  0.37625  0.3   0.565
##   0.011695906  0.37625  0.3   0.565
##   0.017543860  0.37625  0.3   0.565
##   0.023391813  0.37625  0.3   0.565
##   0.029239766  0.37625  0.3   0.565
##   0.035087719  0.37625  0.3   0.565
##   0.040935673  0.37625  0.3   0.565
##   0.046783626  0.37625  0.3   0.565
##   0.052631579  0.37625  0.3   0.565
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.05263158.
```

Build the tree

We'll just use the tuned parameter to build a new tree:

```
tree_model_proposed_locations = rpart(factor(success,
                                             levels = c(0,1),
                                             labels = c("High-Latency Target",
                                                        "Low-Latency Target")) ~ .,
                                      proposed_locations_continuous_train,
                                      control = rpart.control(cp = 0.05263158))

rpart.plot(tree_model_proposed_locations,
           type = 4,
           clip.right.labs = FALSE,
           extra = 102,
           box.palette = "GyGn",
           ycompress = TRUE,
           fallen.leaves = FALSE,
```

```
           branch = 1)
```



Prediction accuracy on test data:

```
predictions <- predict(tree_model_proposed_locations,
                       proposed_locations_continuous_test,
                       type = "class")
table(proposed_locations_continuous_test$success, predictions)
```

```
##    predictions
##     High-Latency Target Low-Latency Target
##   0                   6                  2
##   1                   4                 14
```

```
#(6+14)/26 = 76.9%: better than guessing!
```

Experiment 4.1: Color Clusters with Uniformly-Drawn Latency

```
color_clusters <- continuous_data  %>%
  filter(condition == "Experiment4")  %>%
  mutate(success = ifelse(foundFastTargetFirst == TRUE, 1, 0)) %>%
  replace(., is.na(.), "not reported") %>%
  drop_na() %>%
  select(-userid, -condition, -foundFastTargetFirst)
```

Sanity check that we're not talking about a "rare event":

```
prop.table(table(color_clusters$success))
```

```
##
##         0         1
```

```
## 0.4174757 0.5825243
```

Excellent: roughly 58.3% of trials ended in the user finding the low-latency target first, and the remaining 41.7% ended in the user finding the high-latency target first.

Logistic Regression

```
logistic_model_color_clusters <- glm(success ~ latency,
                          family = binomial(link = 'logit'),
                          data = color_clusters)

summary(logistic_model_color_clusters)
```

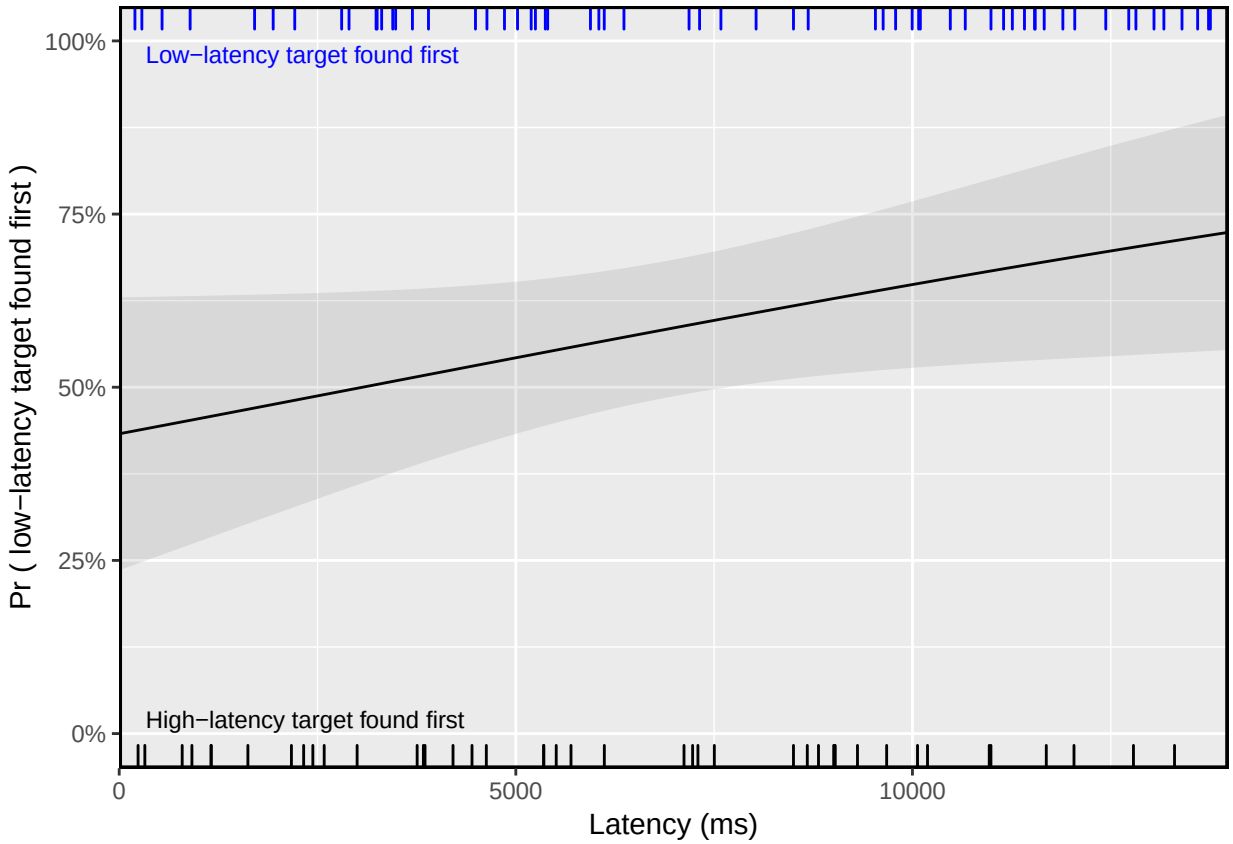
```
##
## Call:
## glm(formula = success ~ latency, family = binomial(link = "logit"),
##     data = color_clusters)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.5768  -1.2330   0.8521   1.0668   1.2865
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.704e-01  4.010e-01  -0.674   0.5002
## latency      8.818e-05  5.141e-05   1.715   0.0863 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 139.97  on 102  degrees of freedom
## Residual deviance: 136.95  on 101  degrees of freedom
## AIC: 140.95
##
## Number of Fisher Scoring iterations: 4
```

This relationship is significant again only at the level of p = 0.1. When we view these results graphically, we observe the same effect:

```
test_data_color_clusters <- data.frame(latency = sample(c(0:14000), size = 1000)) %>%
  mutate(y_hat = predict(logistic_model_color_clusters,
                                      newdata = .,
                                      type = 'response'),
         wrong_se = predict(logistic_model_color_clusters,
                                          newdata = .,
                                          type = 'response',
                                          se.fit = TRUE)$se.fit,
         wrong_upr = y_hat + (2 * wrong_se),
                    wrong_lwr = y_hat - (2 * wrong_se))

color_clusters_low_first <- color_clusters %>%
  filter(success == 1)
```

```r
color_clusters_high_first <- color_clusters %>%
  filter(success == 0)

## plot it
ggplot(test_data_color_clusters, aes(x = latency, y = y_hat)) +
    geom_line() +
    # Add standard error band
    geom_ribbon(data = test_data_color_clusters, aes(ymin = wrong_lwr, ymax = wrong_upr),
                alpha = 0.1) +
    # Add rug plot above to denote observed low-latency-first cases
    geom_rug(aes(y = success, x = latency),
             data = color_clusters_low_first,
             sides = "t",
             color = "blue") +
    # Add rug plot below to denote observed low-latency-first cases
    geom_rug(aes(y = success, x = latency),
             data = color_clusters_high_first,
             sides = "b") +

    xlab("Latency (ms)") +
    ylab("Pr ( low-latency target found first )") +
    xlim(0,14000) +
    scale_y_continuous(labels = scales::percent) +
    scale_x_continuous(expand=c(0,0))+
    theme(panel.border = element_rect(colour = "black", fill=NA, size=1))+
    annotate("text", x = 0, y = 0.02,
             label = "    High-latency target found first",
             hjust = 0, size = 3) +
    annotate("text", x = 0, y = 0.98,
             label = "    Low-latency target found first",
             hjust = 0, size = 3, color = "blue")
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which
## will replace the existing scale.
```



As latency increases, we observe an increase in the probability of finding the low-latency target first.

Recursive Partitioning (10-fold cross validation to select best cp)

```
set.seed(1)
inTrain <- createDataPartition(
  y = color_clusters$success,
  p = .7,
  list = FALSE
)

color_clusters_train = color_clusters[inTrain,]
color_clusters_test = color_clusters[-inTrain,]
```

```
model <- train(factor(success,
                      levels = c(0,1),
                      labels = c("High.Latency.Target",
                                 "Low.Latency.Target")) ~ . ,
               data = color_clusters_train,
               method='rpart',
               tuneLength=10,
               trControl=trainControl(method = 'cv',
                                      number = 10,
                                      classProbs = TRUE,
                                      summaryFunction = twoClassSummary))
```

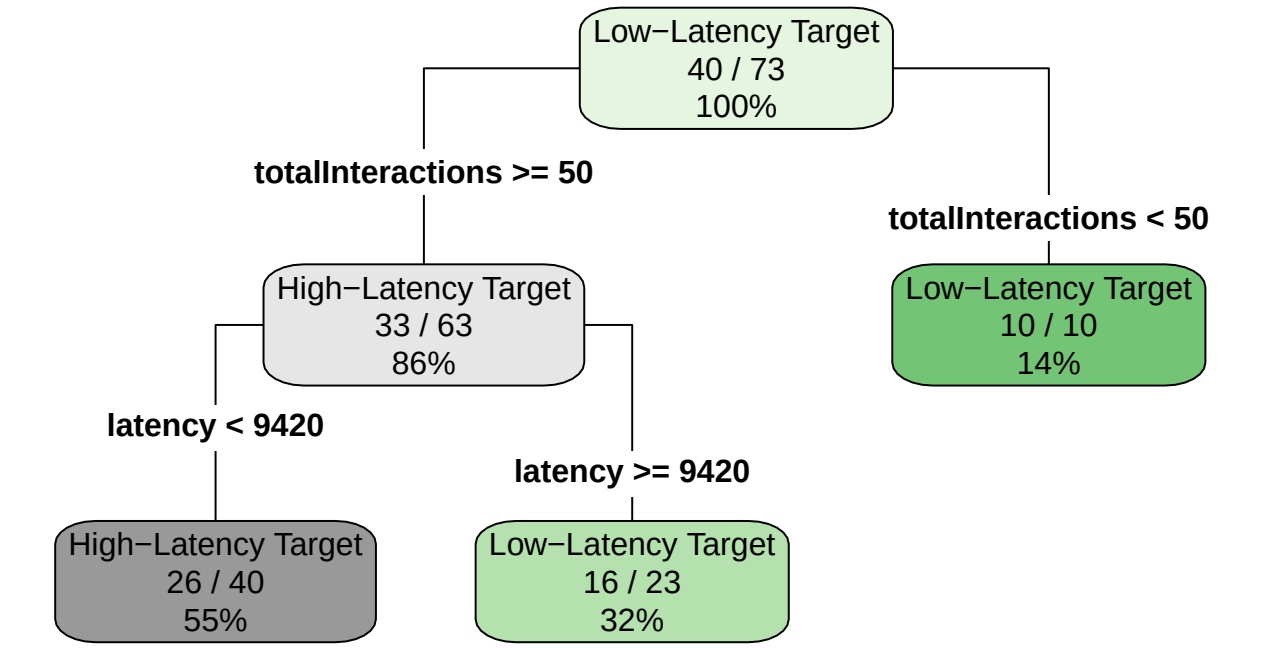
```
## Warning in train.default(x, y, weights = w, ...): The metric "Accuracy" was
```

```
## not in the result set. ROC will be used instead.
```

```
model
```

```
## CART
##
## 73 samples
## 13 predictors
##  2 classes: 'High.Latency.Target', 'Low.Latency.Target'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 66, 65, 65, 66, 66, 66, ...
## Resampling results across tuning parameters:
##
##   cp          ROC        Sens       Spec
##   0.00000000  0.5760417  0.4416667  0.600
##   0.02020202  0.5760417  0.4416667  0.600
##   0.04040404  0.5802083  0.5416667  0.625
##   0.06060606  0.5802083  0.5416667  0.625
##   0.08080808  0.5781250  0.6000000  0.550
##   0.10101010  0.5781250  0.6000000  0.550
##   0.12121212  0.6031250  0.6500000  0.575
##   0.14141414  0.6031250  0.6500000  0.575
##   0.16161616  0.5833333  0.6083333  0.550
##   0.18181818  0.5208333  0.3333333  0.700
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.1414141.
```

```
tree_model_color_clusters = rpart(factor(success,
                                         levels = c(0,1),
                                         labels = c("High-Latency Target",
                                                    "Low-Latency Target")) ~ .,
                                  color_clusters_train,
                                  control = rpart.control(cp = 0.1414141))

rpart.plot(tree_model_color_clusters,
           type = 4,
           clip.right.labs = FALSE,
           extra = 102,
           box.palette = "GyGn",
           ycompress = TRUE,
           fallen.leaves = FALSE,
           branch = 1,
           main="Recursive Partitioning of Experiment 4.1 to Predict First Target Located",
           legend.x=0,
           legend.y=0.1)
```



#Prediction accuracy on test data:

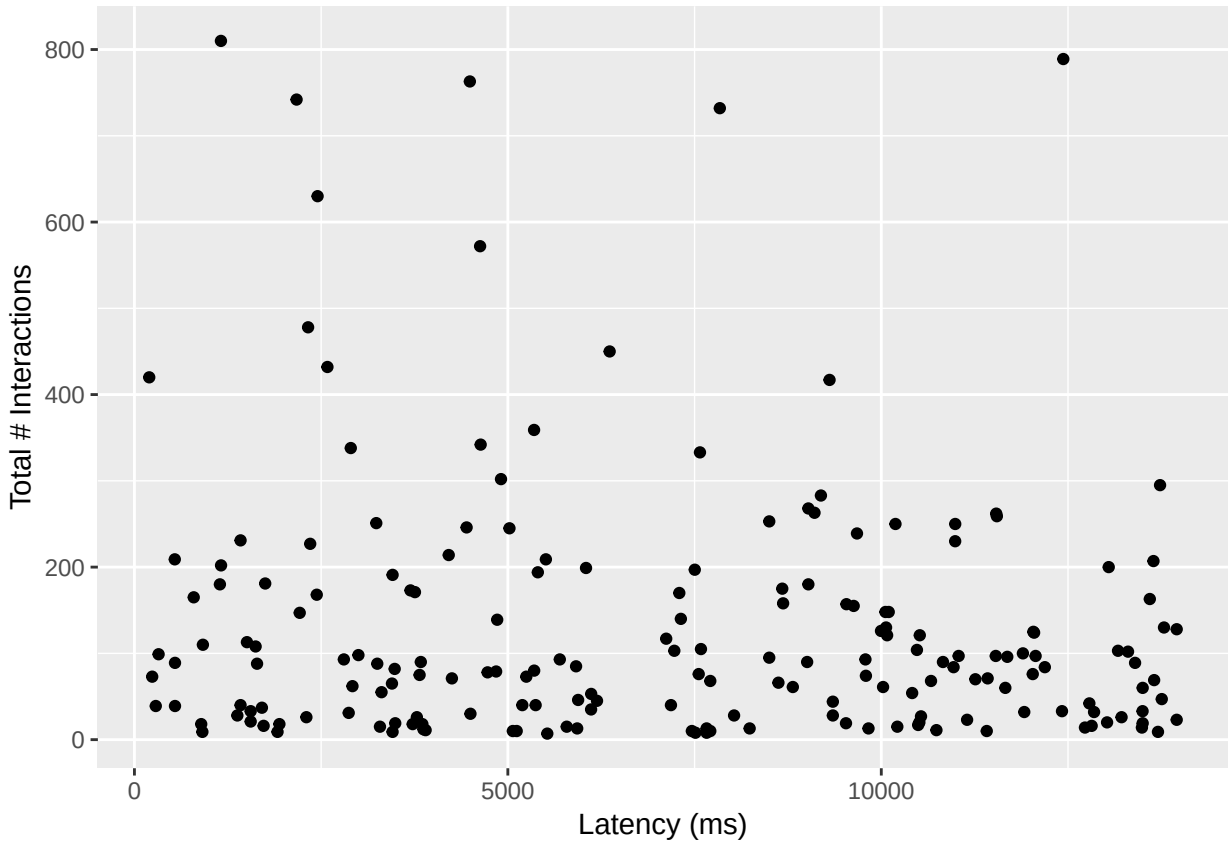
```
predictions <- predict(tree_model_color_clusters,
                       newdata = color_clusters_test,
                       type = "class")
pander(table(color_clusters_test$success, predictions))
```

	High-Latency Target	Low-Latency Target
0	7	3
1	9	11

```
#(7+11)/30 = 60%: ever-so-slightly better than guessing success=1 for all trials
```

Check correlation between total number of interactions and latency

```
rbind(color_clusters, proposed_locations_continuous) %>%
  ggplot(aes(x = latency, y = totalInteractions)) +
  geom_point() +
  xlab("Latency (ms)") +
  ylab("Total # Interactions")
```




```
cor(color_clusters$latency, color_clusters$totalInteractions)
```

```
## [1] -0.1450301
```

Search Strategy Analysis

One challenge to performing this kind of analysis is the relance on a binary outcome to codify the entirety of an interactive process. As demonstrated above, latency is just one of many factors that may influence a user's behavior on a visual search task, and the ultimate outcome is a relatively blunt instrument for measuring effect. In this section, we dig deeper into various elements of **search strategy**.

Experiments 1-3: Pilot (revisited)

```
search_strategies_pilot <- pilot_data %>%
  mutate(strategySwitch = if_else(search_strategy == "strategy_switch", TRUE, FALSE))
```

Relationship between strategy and condition:

```
pander(table(search_strategies_pilot$search_strategy, search_strategies_pilot$condition))
```

	Experiment1	Experiment2	Experiment3
direct	4	4	61

	Experiment1	Experiment2	Experiment3
opportunistic_random	24	42	16
strategy_switch	35	18	14
structured_search	38	40	15

Relationship between strategy and success

```
pander(chisq.test(search_strategies_pilot$search_strategy,
                  search_strategies_pilot$foundFastTargetFirst,
                  correct = FALSE))
```

Table 12: Pearson's Chi-squared test: `search_strategies_pilot$search_strategy` and `search_strategies_pilot$foundFastTargetFirst`

Test statistic	df	P value
0.7287	3	0.8664

Relationship between strategy and latency condition

```
pander(chisq.test(search_strategies_pilot$search_strategy,
                  search_strategies_pilot$latency,
                  correct = FALSE))
```

Table 13: Pearson's Chi-squared test: `search_strategies_pilot$search_strategy` and `search_strategies_pilot$latency` ### Relationship between strategy and condition

Test statistic	df	P value
19.17	12	0.08448

```
pander(chisq.test(search_strategies_pilot$search_strategy,
                  search_strategies_pilot$condition,
                  correct = FALSE))
```

Table 14: Pearson's Chi-squared test: `search_strategies_pilot$search_strategy` and `search_strategies_pilot$condition`

Test statistic	df	P value
129.4	6	1.681e-25 * * *

Experiment 3.2: Proposed Locations with Uniformly-Drawn Latency (revisited)

```
proposed_locations_continuous <- proposed_locations_continuous %>%
  mutate(strategySwitch = if_else(search_strategy == "strategy_switch", TRUE, FALSE))
```

Sanity check that we're not talking about a "rare event":

```
prop.table(table(proposed_locations_continuous$strategySwitch))
```

```
##
##     FALSE      TRUE
## 0.8636364 0.1363636
```

Uh-oh... we don't really have enough examples of strategy switches for regular Logistic Regression (max. likelihood) to be accurate. Instead, we'll use Firth's method (which use penalized likelihood):

Logistic Regression

```
library(brglm)
```

```
## Loading required package: profileModel
```

```
## 'brglm' will gradually be superseded by 'brglm2' (https://cran.r-project.org/package=brglm2), which
```

```
logistic_model_search_strategies_proposed_locations_continuous <- brglm(strategySwitch ~ latency,
                          data = proposed_locations_continuous)

summary(logistic_model_search_strategies_proposed_locations_continuous)
```

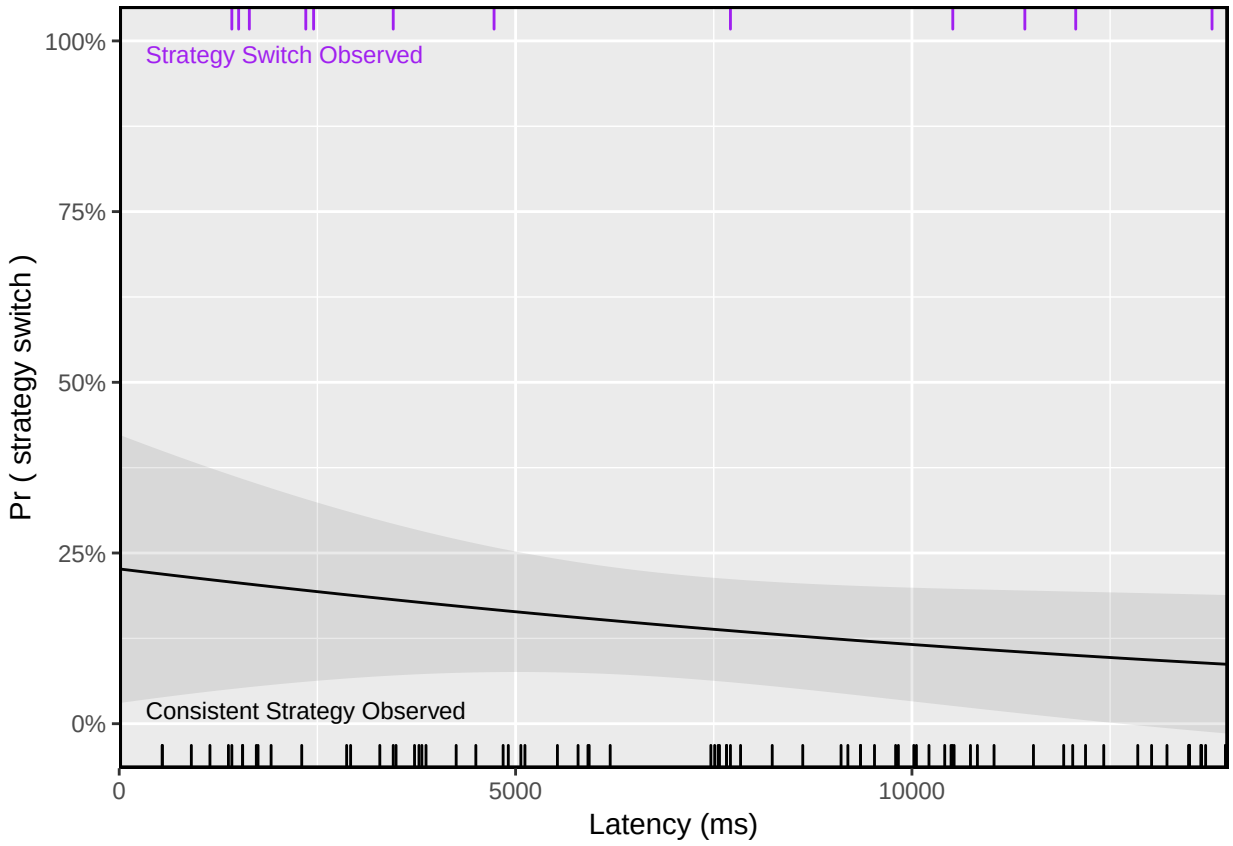
```
##
## Call:
## brglm(formula = strategySwitch ~ latency, data = proposed_locations_continuous)
##
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.227e+00  5.605e-01  -2.189   0.0286 *
## latency     -8.037e-05  7.370e-05  -1.091   0.2755
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 63.493  on 87  degrees of freedom
## Residual deviance: 68.836  on 86  degrees of freedom
## Penalized deviance: 47.43275
## AIC:  72.836
```

Graphically:

```
test_data_search_strategies_proposed_locations_continuous <- data.frame(latency = sample(c(0:14000),
                                                                                          size = 1000)) %
  mutate(y_hat = predict(logistic_model_search_strategies_proposed_locations_continuous,
                                    newdata = .,
```

```
                                        type = 'response'),
         wrong_se = predict(logistic_model_search_strategies_proposed_locations_continuous,
                                           newdata = .,
                                           type = 'response',
                                           se.fit = TRUE)$se.fit,
         wrong_upr = y_hat + (2 * wrong_se),
                     wrong_lwr = y_hat - (2 * wrong_se))

proposed_locations_continuous_strategy_switch <- proposed_locations_continuous %>%
  filter(strategySwitch == TRUE)

proposed_locations_continuous_strategy_persistent <- proposed_locations_continuous %>%
  filter(strategySwitch == FALSE)

## plot it
ggplot(test_data_search_strategies_proposed_locations_continuous,
       aes(x = latency, y = y_hat)) +
    geom_line() +
    # Add standard error band
    geom_ribbon(data = test_data_search_strategies_proposed_locations_continuous,
                aes(ymin = wrong_lwr, ymax = wrong_upr),
                  alpha = 0.1)+
    # Add rug plot above to denote observed strategy switch cases
    geom_rug(aes(y = as.numeric(strategySwitch), x = latency),
             data = proposed_locations_continuous_strategy_switch,
             sides = "t",
             color = "purple") + # Add rug plot below to denote observed single strategy
    geom_rug(aes(y = as.numeric(strategySwitch), x = latency),
             data = proposed_locations_continuous_strategy_persistent,
             sides = "b") +

    xlab("Latency (ms)") +
    ylab("Pr ( strategy switch )") +
    xlim(0,14000) +
    scale_y_continuous(labels = scales::percent) +
    scale_x_continuous(expand=c(0,0))+
    theme(panel.border = element_rect(colour = "black", fill=NA, size=1))+
    annotate("text", x = 0, y = 0.02,
             label = "    Consistent Strategy Observed",
             hjust = 0, size = 3) +
    annotate("text", x = 0, y = 0.98,
             label = "    Strategy Switch Observed",
             hjust = 0, size = 3, color = "purple")
```

```
## Scale for 'x' is already present. Adding another scale for 'x', which
## will replace the existing scale.
```



Experiment 4.1: Color Clusters with Uniformly-Drawn Latency (revisited)

```
color_clusters <- color_clusters %>%
  mutate(strategySwitch = if_else(search_strategy == "strategy_switch", TRUE, FALSE))
```

Sanity check that we're not talking about a "rare event":

```
table(color_clusters$strategySwitch)
```

```
##
## FALSE  TRUE
##    67    36
```

Logistic Regression (using latency to predict strategy switches)

```
logistic_model_search_strategies_color_clusters_latency <- glm(strategySwitch ~ latency,
                          family = binomial(link = 'logit'),
                          data = color_clusters)

summary(logistic_model_search_strategies_color_clusters_latency)
```

```
##
## Call:
## glm(formula = strategySwitch ~ latency, family = binomial(link = "logit"),
##     data = color_clusters)
```

```
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.9577  -0.9349  -0.9118   1.4297   1.4859
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -7.034e-01  4.178e-01  -1.684   0.0922 .
## latency      1.177e-05  5.174e-05   0.227   0.8201
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 133.31  on 102  degrees of freedom
## Residual deviance: 133.26  on 101  degrees of freedom
## AIC: 137.26
##
## Number of Fisher Scoring iterations: 4
```

```
test_data_search_strategies_color_clusters_latency <- data.frame(latency = sample(c(0:14000), size = 10
  mutate(y_hat = predict(logistic_model_search_strategies_color_clusters_latency,
                                     newdata = .,
                                     type = 'response'),
         wrong_se = predict(logistic_model_search_strategies_color_clusters_latency,
                                          newdata = .,
                                          type = 'response',
                                          se.fit = TRUE)$se.fit,
         wrong_upr = y_hat + (2 * wrong_se),
                   wrong_lwr = y_hat - (2 * wrong_se))

color_clusters_strategy_switch <- color_clusters %>%
  filter(strategySwitch == TRUE)

color_clusters_strategy_persistent <- color_clusters %>%
  filter(strategySwitch == FALSE)

## plot it
ggplot(test_data_search_strategies_color_clusters_latency, aes(x = latency, y = y_hat)) +
    geom_line() +
    # Add standard error band
    geom_ribbon(data = test_data_search_strategies_color_clusters_latency,
                aes(ymin = wrong_lwr, ymax = wrong_upr),
                  alpha = 0.1) +
    # Add rug plot above to denote observed strategy switches
    geom_rug(aes(y = as.numeric(strategySwitch), x = latency),
             data = color_clusters_strategy_switch,
             sides = "t",
             color = "purple") +
    # Add rug plot below to denote observed single strategy
    geom_rug(aes(y = as.numeric(strategySwitch), x = latency),
             data = color_clusters_strategy_persistent,
             sides = "b") +
```

```
    xlab("Latency (ms)") +
    ylab("Pr ( strategy switch )") +
    #xlim(0,14000) +
    scale_y_continuous(labels = scales::percent) +
    scale_x_continuous(expand=c(0,0))+
    theme(panel.border = element_rect(colour = "black", fill=NA, size=1))+
    annotate("text", x = 0, y = 0.02,
             label = "    Consistent Strategy Observed",
             hjust = 0, size = 3) +
    annotate("text", x = 0, y = 0.98,
             label = "    Strategy Switch Observed",
             hjust = 0, size = 3, color = "purple")
```