Analytics Notes

Gonzalo Lezma

October 7, 2020

Chapter 1

Introduction to Machine Learning

1.1 Where do machine learning sits

Artificial inteligence is a broad term that defnes all programs with the ability to imite the cognitive skills of humans. This definition includes proggrams in which the rules have been explicitly programmed. The term Machine Learning denotes programs with the ability to learn without being explicitly programmed. Neural networks are functions that translate input data in output data. This functions have the ability to be flexible enough to learn any pattern that you throw to them. Given enough data they can learn any data pattern. Deep learning is the study and applications of neural network that uses many layers of neural networs. In this arrangement the output of the neural network in a given layer becomes the input of another neural network in the next layer. This type of arrangement has proven to be specially effective when learning complex patterns.

1.2 Classification of models

The most usual classification is based on whether or not the models are trained with or without human supervision:

1. **Supervised learning:** Includes algorithms that are used to predict a dependent variable using data that has been previously labeled. Examples include: Linear regression, logistic regression, k-Nearest Neigghbors, Support Vector Machines (SVMs), Decision Trees and Random forest, Neural Networks.

2. **Unsupervised learning:** Here the system will try to learn patterns without using any labeled data. Examples include: Clustering (K-Means, DBScan , Hierarchical Cluster Analysis (HCA), Anomaly detection, Visualization (t-SNE, PCA) , Association rule learning (Apriori algorithm, Eclat), etc.

3. **Reinforcement learning:** Here the learning system is called an agent that can observe the environmet, perform actions and get rewards in return. The agent should learn the best strategy called policy to get the most reward over time. Examples are Bamdits algorithms, and applications such as AlphaZero or AlphaStar from the firm DeepMind.

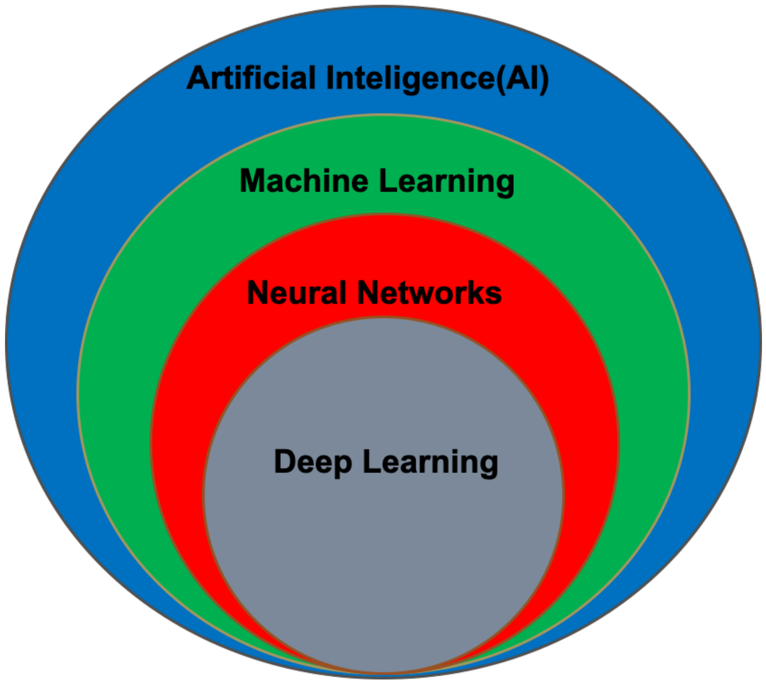

Figure 1.1: Scikit-Learn flow to model selection

Other used classifications are: the ones grouping the so called Classical Methods for ML (Including Regression, regression trees, etc) and the newer Neural networks Methods for Machine Learning (Including all variants, Standard Neural Networks, CNN, Recurrent Neural Networks, Long Short Term Memory, etc.). The ones grouping methods according to its deployment nature. For example some methods can be deploy in a continuous learning process. Some systems can only be maintained and learn in batch mode.

1.3 The data steps to a ML project

1. **Gathering data:** The first step in our analytics journey will be to data from datasources such as a webpage, the API of our Marketing management Platform, CRM, Datawarehouses, a Datamart, Social Networks, Google Analytics, etc. Usually the data is collected by a data unit (composed by data engineers) getting information that was send from different platforms, in the case of a completely online business the information is collected and sent by a javascript script detecting user interactions with the webpage. In big organizations, data engineers not only create processes for data gathering but also build and configure the data storage and access center s which are usually distributed into several servers and nodes optimized for the use of Big Data Techniques.

2. **Preparing that data:** Data prep includes tree steps. Firstly, the raw data has to be formed into a usable table that can be consumed by the data science teams in a process usually refered to as Data Wrangling or Data Munging. In big organizations the experts performing this tasks are Data Analysts. These are people who know the

table structure of the organization and make extensive use of SQL-like languages as the main tool. Secondly the sampling bias should be assessed. It is possible to have some sample bias due to a selection bias of the firm?. In this case the oversampling methods should be used to avoid spurious findings. Lastly, a data partition should be set composed of a training and a test set. Sometimes it is also convenient to use a development set. The training set is the partition of data on which we perform the model estimation. We will use the test set to evaluate our model on data the model have never seen before. The development set is used to estimate the best model when its estimation depends on numerical hyper-parameters such as LASSO, Ridge regression, etc.

3. **Descriptive statistics on the training data:** Visualize and get descriptive statistic to gain insigths on data:

4. **Choosing a model:** In this step we will choose a model based on the nature of the problem and the availability of data. The first category we can chose from is supervised learning models. In such models the outcome is known or labeled [1] so we continuously refine the model itself until our output reaches the desired performance. Models that can be used here are copious in number, going from basic regression (linear ,logistic, etc) to the more complex and state of the art neural networks. If the outcome is unknown (not labeled) and we need classification to be done then the second category, unsupervised learning, is used. Examples of unsupervised learning include K-means and DBSCAN. The third category is reinforcement learning. This category focuses on learning on the basis of trial and error. Bandit algorithms and markov chain models can be used in this category.

5. **Training** This step includes the following parts. First, selecting and transforming variables to include in our model. This process is called feature engineering. This process can include generating different variables as simple transformations of other variables, converting categorical variables to numeric codes, using encodings and embeddings to represent words or concepts, etc. Finally performing the actual estimation using optimal estimators for each case (Gradient descent for neural networks, OLS for regression, Maximun Likelihood for logistic regression, etc.). If the model includes a numerical hyperparameter it will be necessary to use the dev set to tune up those parameters.

6. **Evaluation** The evaluation step consists in finding out the performance of our model in the test data. This evaluation criteria can be simple such as an RMSE or complex in terms of a ratios derived from a confusion matrix. Taking into account accuracy, precission, recall, etc.

7. **Hyperparameter tuning** After a model is selected in the evaluation process we could still tune up additional hyperparameters such as the ones corresponding to the initial values in the estimation process or using a better algorithm to improve the model performance.

[1] A great resource to label data is available at https://www.mturk.com/

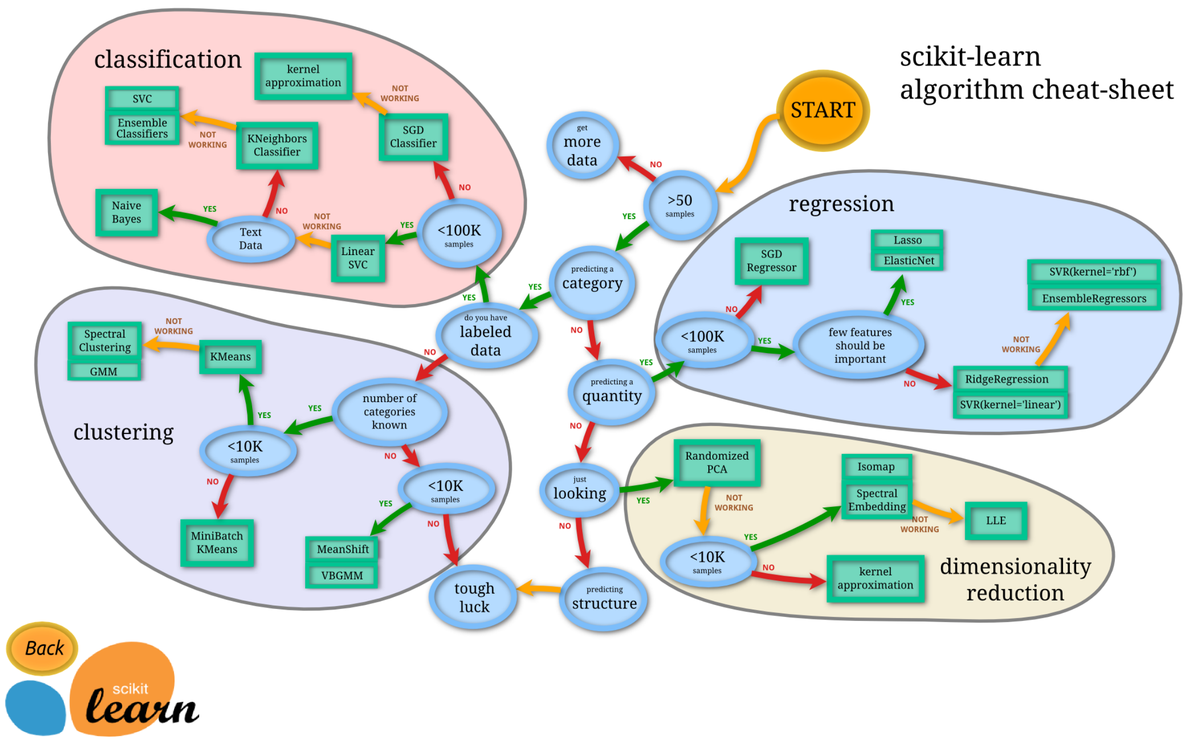

Figure 1.2: Scikit-Learn flow to model selection

8. **Launch, monitor and maintain your system** The last step is when the model is pass to production and its living independently of the modeler. Any ML system should live in a test setting. The so called champion-challenger approach, promote the current system to be constantly challenged by other models, if the challenger turns out to have better performance it will become the new champion.

Chapter 2

Supervised Learning or Predictive Modeling

There's 2 types of predictive models: Classification and Regression. In classification the model predicts a discrete variable whereas in Regression the predicted variable is continuous. Sometimes it is said that classification is meant to predict Qualitative variables and Regression can be used to predict quantitative variables, however, this claim is not exactly correct since it is very simple to turn a regression model into a Classification model by discretizing the dependent variable.

2.1 The over-fitting and Cross Validation

To compute the goodness of fit of a model it is common to use the RMSE (Root Mean Squared Error). Based on this metric we can select the best model that adjusts to our data however this usually leads to be too optimistic with respect to the results. It is usual to see a model that performs well in the training sample but does not generalize well when used to predict out of the sample. This problem is called over-fitting.

Splitting our sample in a train and test sets once, has the advantage of being simple and allows us to estimate the out of sample error of our model. However, any outlier and the randomness asociated with our sample partition could bias our out-of-sample error estimatin. In order to avoid that mistake, cross-validation (CV) is a great alternative. CV performs several train-test partitions in our data with the restriction that every data sample is only one time in the test data set.

CV is only use to estimate the out of sample error to compare between models. After a model is selected, the estimation should be performed on the entire dataset

2.2 Linear Regression

```
# 1) Fitting a linear model to in the mtcars data
# Fitting a linear model to the mtcars data
data(mtcars)
```

```r
model <-lm(mpg ~ hp, mtcars[1:20,])
# Predict in-sample (Getting RMSE for the training sample)
predicted <- predict(
  model, mtcars[1:20,],type='response'
)

# Computing rmse
actual <- mtcars[1:20,'mpg']
sqrt(mean((predicted-actual)^2))

# 2) Fitting a linear model to in the diamonds data
# Fit lm model: model
model <- lm(price ~ . ,diamonds)

# Predict on full data: p
predicted <- predict(model,
                     diamonds, type='response')

# Compute errors: error
actual <- diamonds$price
errors = predicted-actual

# Calculate RMSE
sqrt(mean(errors^2))

data(mtcars)
model <- lm(mpg ~hp, mtcars[1:20,])

# Prediction out of sample
predicted <- predict(
  model, mtcars[21:32,],type = "response"
)

# compite mse error
actual <- mtcars[21:32, "mpg"]
sqrt(mean((predicted - actual) ^ 2))

# ORDER A DF RANDOMLY
# Set seed
set.seed(42)


# Shuffle row indices: rows

rows <- sample(nrow(diamonds))

# Randomly order data
shuffled_diamonds <- diamonds[rows,]


# DO AN 80/20 SPLIT

# Determine row to split on: split
split <- round(nrow(diamonds) * 0.80)
```

```r

# Create train
train <- diamonds[1:split, ]

# Create test
test <-diamonds[(split + 1):nrow(diamonds), ]


# PREDICT ON A TEST SET

# Fit lm model on train: model
model <- lm(price~.,train)

# Predict on test: p
p <- predict(model, test)

# Compute errors: error
error <- p-test$price

# Calculate RMSE
sqrt(mean(error^2))

library(caret)
# CROSS VALIDATION
set.seed(42)

# Fit linear  re gresssion model
model <- train(
  mpg ~ hp, mtcars,
  method = 'lm',
  trControl = trainControl(
    method = 'cv',
    number = 10,
    verboseIter = TRUE
  )
)

model <- train(
  mpg ~ hp,
  mtcars,
  method = "lm",
  trControl = trainControl(
    method = "repeatedcv",
    number = 5,
    repeats = 5,
    verboseIter = TRUE
  )
)
summary(model)
```

Listing 2.1: Logistic Regression using Caret

2.3 Logistic Regression

```r
# Get the number of observations
n_obs = nrow(Sonar)

# Shuffle row indices: permuted_rows
permuted_rows <- sample(n_obs)

# Randomly order data: Sonar
Sonar_shuffled <- Sonar[permuted_rows,]

# Identify row to split on: split
split <- round(n_obs * 0.6)

# Create train
train <- Sonar_shuffled[1:split,]

# Create test
test <- Sonar_shuffled[(split+1):n_obs,]

# Logistic reggresion


# Fit glm model: model
model <-glm(Class ~.,family = "binomial", train)

# Predict on test: p
p<-predict(model, test, type = "response")


# CONFUSUION MATRIX

# If p exceeds threshold of 0.5, M else R: m_or_r
m_or_r <- ifelse(p>0.5,"M","R")

# Convert to factor: p_class
p_class <- factor(m_or_r,levels = levels(test[['Class']]))

# Create confusion matrix
confusionMatrix(p_class,test[["Class"]])
```

Listing 2.2: Logistic Regression using Caret

2.4 CART (Classification and Regression Trees

```r
library(tidyverse)
library(caret)
library(rpart)
library(mlbench)

# Load the data and remove NAs
data("PimaIndiansDiabetes2", package = "mlbench")
PimaIndiansDiabetes2 <- na.omit(PimaIndiansDiabetes2)
# Inspect the data
sample_n(PimaIndiansDiabetes2, 3)
```

```r
# Split the data into training and test set
set.seed(123)
training.samples <- PimaIndiansDiabetes2$diabetes %>%
  createDataPartition(p = 0.8, list = FALSE)
train.data  <- PimaIndiansDiabetes2[training.samples, ]
test.data <- PimaIndiansDiabetes2[-training.samples, ]




# Build the model
set.seed(123)
model1 <- rpart(diabetes ~., data = train.data, method = "class")
# Plot the trees
par(xpd = NA) # Avoid clipping the text in some device
plot(model1)
text(model1, digits = 3)


# Make predictions on the test data
predicted.classes <- model1 %>%
  predict(test.data, type = "class")
head(predicted.classes)


# Compute model accuracy rate on test data
mean(predicted.classes == test.data$diabetes)


# Fit the model on the training set
set.seed(123)
model2 <- train(
  diabetes ~., data = train.data, method = "rpart",
  trControl = trainControl("cv", number = 10),
  tuneLength = 10
)
# Plot model accuracy vs different values of
# cp (complexity parameter)
plot(model2)


# Print the best tuning parameter cp that
# maximizes the model accuracy
model2$bestTune


# Plot the final tree model
par(xpd = NA) # Avoid clipping the text in some device
plot(model2$finalModel)
text(model2$finalModel,  digits = 3)

# Decision rules in the model
```

```
model2$finalModel
```

Listing 2.3: Classification tree with rpart

Chapter 3

Unsupervised Learning

The role of unsupervised learning is to identify patterns in data without needing to use labels. The most common application for unsupervised learning is clusterization. Clusterization is the problem in which we need to group customers according to similar characteristics. For example consider we need to identify several group of customers according to age, geolocation, language, gender, risk, purchasing behaviour, income, education, etc. etc. Enumerating the different combinations of such many variables would be impractical and useless. We need an algorithm that tells us, based only on the data, what are the clusters we need to consider.

3.1 Kmeans

K-means is a clustering algorithm for identifying homogeneous groups. It considers a pre-defined number of clusters and breaks the observations into those clusters. The algorithm's result depends on the initial condition that can be defined by the user. If you had any business prior to select the number of clusters in advance, this makes setting the number of clusters easy. In most of the cases you will not have an idea of what is the number of clusters and you will need to compute it.

The usual approach to select the number of clusters is to run the algorithm multiple times, each time with a different number of clusters. From this exercise you can observe how a measure of model quality changes with the number of clusters. The ideal plot will show an elbow. The elbow point indicates the number of clusters at which adding one more cluster does not improve the model substantially. Therefore the elbow point usually indicates the number of clusters you should use.

```

# k-means
library(tidyverse)
x<-read_csv("data_clustering.csv")
kmeans(x, centers = 3, nstart = 20)


# Create the k-means model: km.out
km.out <-kmeans(x,centers=3,
                nstart=20)
```

```r

# Inspect the result
summary(km.out)
# Print the cluster membership component of the model
print(km.out$cluster)

# Print the km.out object
print(km.out)

# ploting
# Scatter plot of x
plot(x,col=km.out$cluster,main="k-means with 3 clusters",ylab="",xlab="")


# K-means has a random component which is important to understand

# Set up 2 x 3 plotting grid
par(mfrow = c(2, 3))

# Set seed
set.seed(1)

for(i in 1:6) {
  # Run kmeans() on x with three clusters and one start
  km.out <- kmeans(x, centers=3,nstart=1)

  # Plot clusters
  plot(x, col = km.out$cluster,
       main = km.out$tot.withinss,
       xlab = "", ylab = "")
}

# The elbow

# Initialize total within sum of squares error: wss
wss <- 0

# For 1 to 15 cluster centers
for (i in 1:15) {
  km.out <- kmeans(x, centers = i, nstart = 20)
  # Save total within sum of squares to wss variable
  wss[i] <- km.out$tot.withinss
}

# Plot total within sum of squares vs. number of clusters
plot(1:15, wss, type = "b",
      xlab = "Number of Clusters",
      ylab = "Within groups sum of squares")

# Set k equal to the number of clusters corresponding to the elbow
   location
k <- 2
Xraw<-read_csv('')
library(Rtsne)
```

```
X<-normalize_input(Xraw)
tsne_out<-Rtsne(X, dims = 2)

library(ggplot2)
tsne_plot <- data.frame(x = tsne_out$Y[,1], y = tsne_out$Y[,2], col = iris
    _unique$Species)
ggplot(tsne_plot) + geom_point(aes(x=x, y=y, color=col))
```

Listing 3.1: Logistic Regression using Caret

Chapter 4

Testing

4.1 Preliminary definitions

- Type I error (α , also called significance level): the probability to reject H_0 (the null hypothesis) when it is true. (False positive)

- Confidence level $(1 - \alpha)$: ability to produce accurate intervals that include the true parameter value if many samples were to be generated

- Type II error (β): the probability to FAIL to reject H_0 when it is false.(False negative)

- Power of the statistical test $(1 - \beta)$:the probability to reject H_0 when it is false. The power of the test is directly related to the sample size required to detect an effect of a particular magnitude. Power gives a method of discriminating between competing test of the same hypothesis. The preferred test is always the one with more power.

The usual values chosen for an statistical test under the frequentist point of view are $\alpha = 0.05$ and $\beta = 0.2$ (i.e significance of 5% and power of 80%. [1]

4.2 AB testing

The fundamental objective of an A/B test is that we consider a counterfactual universe where the population under study is identical but subject to another experience (page design,treatment, etc). Also, we are interested in the probability of conversion (purchase, effectiveness of drug, etc). So our goal should be to determine if B has a higher probability of conversion. In the case of a webpage test, we need to show different experiences to randomly different visitors.

[1]For an illustration of how β and α al related visit the interactive animation created by Kristoffer Magnusson at https://rpsychologist.com/d3/nhst/

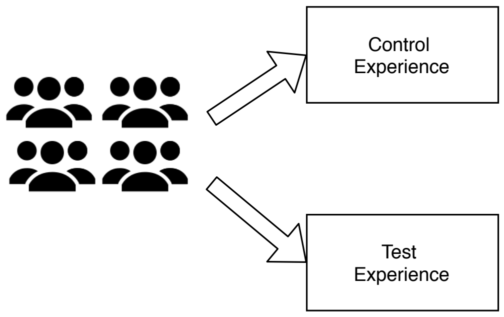

Figure 4.1: AB testing Flow

The Obama Presidential Campaign

”In 2007, Google product manager Dan Siroker took a leave of absence to join the presidential campaign of then senator Barack Obama in Chicago. Heading the New Media Analytics team, Siroker brought one of Google web practices to bear on the campaign bright-red DONATE button. The result was nothing short of astonishing: $57 million of additional donations were raised as a direct result of his work. What exactly did he do to that button? He A/B tested it.

A/B testing works as follows: a company drafts several different versions of a particular webpage. Perhaps they try different colors or images, or different headlines for a news article, or different arrangements of items on the screen. Then they randomly assign incoming users to these various pages, usually in equal numbers. One user may see a red button, while another user may see a blue one; one may see DONATE and another may see CONTRIBUTE. The relevant metrics (e.g., click-through rate or average revenue per visitor) are then monitored. After a period of time, if statistically significant effects are observed, the winning version is typically locked into place or becomes the control for another round of experiments.

In the case of Obama’s donation page, Siroker’s A/B tests were revealing. For first-time visitors to the campaign site, a DONATE AND GET A GIFT button turned out to be the best performer, even after the cost of sending the gifts was taken into account. For longtime newsletter subscribers who had never given money, PLEASE DONATE worked the best, perhaps appealing to their guilt. For visitors who had already donated in the past, CONTRIBUTE worked best at securing follow-up donations the logic being perhaps that the person had already donated but could always contributed more. And in all cases, to the astonishment of the campaign team, a simple black-and-white photo of the Obama family outperformed any other photo or video the team could come up with. The net effect of all these independent optimizations was gigantic.”

Taken from: Christian B. and Griffiths T. 2016 ”Algorithms to Live By”

4.3 The frequentist approach

4.3.1 Power Analysis

1. Set the statistical test you plan to run (mean test, proportion test). Conversion rates are usually number of purchases per visits and as you can tell they are fractions that should be modelled as a proportion test. Number of visitors should be modeled as a mean test. The distinction does not matter when there's lots of data.

2. Compute the baseline value. The value that you expect for the current control. This is just an indicative value, the actual number will be computed during the experiment.

3. Set the desired value (or desired change). I.e. 20% in number of visitors/clients (going from 1000 in the control group to 1200 in the test group) or a 25% lift in the conversion rate (going from 0.4% to 0.5% conversion rate)

4. Set the power and significance of a test

5. Compute the minimum sample so that the results are have enough power to detect a difference of δ. Spotting a small difference will required more data than spotting a big difference. A simplified version of the formula is the one proved by (Van Belle 2008). Where σ^2 is the sample variance and δ is the difference between the Treatment and Control. The difference that would matter in practice. For example you could miss detecting a difference of 0.1% in conversion but a drop of 1% is not ok. In this case 1% would be the smallest difference that is practically significant. [2]

$$n_\delta \approx \frac{16\sigma^2}{\delta^2} \tag{4.2}$$

6. Deviate web traffic. Or split the users randomly between control and test groups and wait.

```r

# install.packages("tidyverse")
library(tidyverse)
library(lubridate)
data = read_csv("abtest.csv")
data

data %>%
  summarize(conv_rate = mean(conversion))
```

[2]The actual sample sized needed for comparing the Means of 2 Normally Distributed Samples of Equal Size Using a Two-Sided Test With $\alpha = 0.05$ and power $1 - \beta = 0.8$ is:

$$n_\delta = \frac{(\sigma_1^2 + \sigma_2^2)(z_{1-\alpha/2} + z_{1-\beta})}{\delta^2} \tag{4.1}$$

Where $\delta = \mu_1 - \mu_2$

```

data %>%
  group_by(month(date)) %>%
  summarize(conversion_rate= mean(conversion))


# Compute conversion rate by week of the year
data_sum <- data %>%
  group_by(week(date)) %>%
  summarize(conversion_rate = mean(conversion))

# Build plot
ggplot(data_sum, aes(x = `week(date)`,
                           y = conversion_rate)) +
  geom_point() +
  geom_line() +
  scale_y_continuous(limits = c(0, 1))

# 1) LOGISTIC REGGRESION
# 1.1) COMPUTE POWER SAMPLE SIZE FOR LOGISTIC REGRESION
install.packages("powerMediation")
library(powerMediation)
total_sample_size <- SSizeLogisticBin(p1 = 0.2,
                                      p2 = 0.3,
                                      B = 0.5,
                                      alpha = 0.05,
                                      power = 0.8)

total_sample_size
total_sample_size/2

# EXPLORE

# Group and summarize data
data_sum <- data %>%
  group_by(experiment, month(date)) %>%
  summarize(conversion_rate = mean(conversion))
data_sum
# Make plot of conversion rates over time
ggplot(data_sum,
       aes(x = `month(date)`,
           y = conversion_rate,
           color = experiment,
           group = experiment)) +
  geom_point() +
  geom_line()

# 1.2) PERFORMING TEST
# Load package for cleaning model results
library(broom)
# View summary of results
data %>%
  group_by(experiment) %>%
  summarize(conversion_rate = mean(conversion))
```

```

# Run logistic regression
experiment_results <- glm(conversion ~ experiment,
                          family = "binomial",
                          data = data) %>%
  tidy()
experiment_results

# 2) T-TEST LINEAR REGRESSION (TIME SPEND ON EACH GROUP)
# 1.1) COMPUTE POWER SAMPLE SIZE FOR LINEAR REGRESSION
library(pwr)
pwr.t.test(power=0.8,
           sig.level=0.05,
           d=0.1
           )
t.test(time_spend ~ experiment,
       data=data)

lm(time_spend ~ experiment, data=data) %>%
  summary()
```

Listing 4.1: Logistic Regression using Caret

4.3.2 Early Stopping

According to power analysis, if we need to detect a difference of δ, we need to wait for $2n_\delta$ users/samples to come. However, what if at the time we get the $2n_\delta$ users we do not get significant results? According to power analysis $2n_\delta$ should be enough for the test. Could it be possible that we overestimated δ and the difference we are after is smaller than we thought? In this case, stopping at that moment would not be long enough.

In this cases, the usual practice is to set up different points in advanced so that we preserve a pre-specified level of type error I

```


# Sequential analysis
# Load package to run sequential analysis
library(gsDesign)

# Run sequential analysis
seq_analysis_3looks <- gsDesign(k = 2,
                                test.type = 1,
                                alpha = 0.05,
                                beta = 0.2,
                                sfu = "Pocock")
seq_analysis_3looks
# find stopping points
max_n <- 3000
max_n_per_group <- max_n / 2
stopping_points <- max_n_per_group * seq_analysis_3looks$timing
stopping_points
```

```

#install.packages("bayesAB")
```

Listing 4.2: Logistic Regression using Caret