Student t test –

m−μ

t= -------

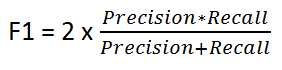
s/√n

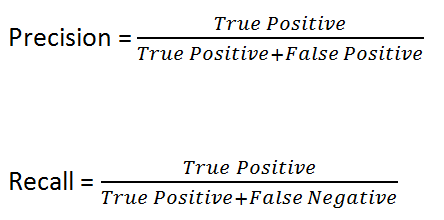
predicted

# P N

# Actual P tp FN

N FP TN





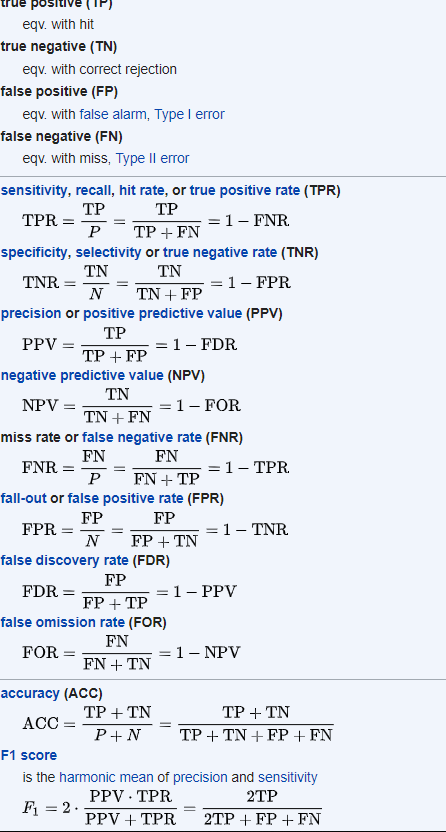
Accuracy = TP+TN/TP+FP+FN+TN

Adjusted *R*2 = 1*−* RSS*/*(*n − d −* 1)

-------------------------------

TSS*/*(*n −* 1)

Receiver operating characteristic



### User Behavior Analysis 2019

* Model to monitor and detect anomalies in user behavior to proactively mitigate attacks and compromises.
* **30% reduction** in user related incidents.

Model to determine anomalies in human behavior

Admin access, notice period, previous DLP, fund access, phishing clicks, designation

Login time, location, download/upload rates, no.of print outs , no .of blocked sites , outbound / inbound traffic,

15/30 days worth of data in 15 mins time period. Isolation forest package to determine anomalies in real time. A mail and alert is sent to SOC team.

To avoid false positives alert is generated when the cumulative score of anomalies goes beyond a threshold. Single one time anomalies wont generate an alert

**IsolationForest**(n\_estimators=100, max\_samples=’auto’, contamination=’legacy’, max\_features=1.0, bootstrap=False, n\_jobs=None, behaviour=’old’, random\_state=None, verbose=0, warm\_start=False)

**\_estimators : *int, optional (default=100)***

The number of base estimators in the ensemble.

**max\_samples : *int or float, optional (default=”auto”)***

**The number of samples to draw from X to train each base estimator.**

* If int, then draw max\_samples samples.
* If float, then draw max\_samples \* X.shape[0] samples.
* If “auto”, then max\_samples=min(256, n\_samples).
* If max\_samples is larger than the number of samples provided, all samples will be used for all trees (no sampling).

**contamination : *float in (0., 0.5), optional (default=0.1)***

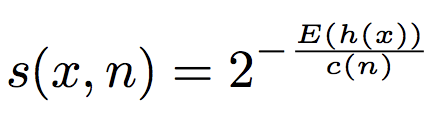
The amount of contamination of the data set, i.e. the proportion of outliers in the data set. Used when fitting to define the threshold on the decision function

**max\_features : *int or float, optional (default=1.0)***

The number of features to draw from X to train each base estimator.

* If int, then draw max\_features features.
* If float, then draw max\_features \* X.shape[1] features.

outliers are less frequent than regular observations and are different from them in terms of values (they lie further away from the regular observations in the feature space). That is why by using such random partitioning they should be identified closer to the root of the tree (shorter average path length, i.e., the number of edges an observation must pass in the tree going from the root to the terminal node), with fewer splits necessary.



where h(x) is the path length of observation x, c(n) is the average path length of unsuccessful search in a Binary Search Tree and n is the number of external nodes. More on the anomaly score and its components can be read in [1].

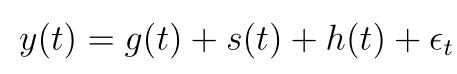
### Cyber Security Threat Hunting & Anomaly Detection 2019

* Threat Detection System to achieve real time threat identification, classification and resolution recommendation.
* Predictive Model to forecast volume/threat level of alerts. Anomaly Detection and Threat Level Rating.
* **50% reduction** in phishing and malware clickers. Reduced time to mitigate issues.
* Detected active campaign vulnerabilities in system.

Time series to forecast the acceptable traffics. Like DDoS , External mail ,incoming mails, nocustomer traffic ,

Used Phrophet package.

We use a decomposable time series model with three main model components: trend, seasonality, and holidays. They are combined in the following equation:

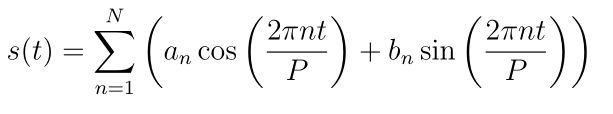


* **g(t)**: piecewise linear or logistic growth curve for modelling non-periodic changes in time series
* **s(t)**: periodic changes (e.g. weekly/yearly seasonality)
* **h(t)**: effects of holidays (user provided) with irregular schedules
* **εt**: error term accounts for any unusual changes not accommodated by the model

A parameter called changepoint\_prior\_scale could be used to adjust the trend flexibility and tackle the above 2 problems. Higher value will fit a more flexible curve to the time series.

Seasonality

To fit and forecast the effects of seasonality, prophet relies on fourier series to provide a flexible model. Seasonal effects s(t) are approximated by the following function:



P is the period (365.25 for yearly data and 7 for weekly data)

Parameters [a1, b1, ….., aN, bN] need to be estimated for a given N to model seasonality.

The fourier order N that defines whether high frequency changes are allowed to be modelled is an important parameter to set here. For a time series, if the user believes the high frequency components are just noise and should not be considered for modelling, he/she could set the values of N from to a lower value. If not, N can be tuned to a higher value and set using the forecast accuracy.

**Holidays and events**

# System Engineer - Fidelity Investments Jan 2017 – Dec 2018

### Log Pattern Anomaly Detection 2018

* Log pattern analysis and anomaly detection. System failure prediction. Dynamic system metrics alerting system.
* **80% reduction** in resource unavailable incidents. **30 hrs/week** time savings.

Three flavors to this project.

1. System metrics – CPU usage, Memory usage . Every 15 secs. Anomaly Detection . Threshold above

anomalize(data, target, method = c("iqr", "gesd" Generalized Extreme Studentized Deviate Test), alpha = 0.05,

max\_anoms = 0.2, verbose = FALSE)

**Data** A tibble or tbl\_time object.

**Target** A column to apply the function to

**Method** The anomaly detection method. One of "iqr" or "gesd". The IQR method is faster at the expense of possibly not being quite as accurate. The GESD method has the best properties for outlier detection, but is loop-based and therefore a bit slower.

**Alpha** Controls the width of the "normal" range. Lower values are more conservative while higher values are less prone to incorrectly classifying "normal" observations.

**max\_anoms** The maximum percent of anomalies permitted to be identified.

* time\_decompose(count, merge = TRUE): This performs a time series decomposition on the “count” column using seasonal decomposition. It created four columns:
  + “observed”: The observed values (actuals)
  + “season”: The seasonal or cyclic trend. The default for daily data is a weekly seasonality.
  + “trend”: This is the long term trend. The default is a Loess smoother using spans of 3-months for daily data.
  + “remainder”: This is what we want to analyze for outliers. It is simply the observed minus both the season and trend.
  + Setting merge = TRUE keeps the original data with the newly created columns.
* anomalize(remainder): This performs anomaly detection on the remainder column. It creates three new columns:
  + “remainder\_l1”: The lower limit of the remainder
  + “remainder\_l2”: The upper limit of the remainder
  + “anomaly”: Yes/No telling us whether or not the observation is an anomaly
* time\_recompose(): This recomposes the season, trend and remainder\_l1 and remainder\_l2 columns into new limits that bound the observed values. The two new columns created are:
  + “recomposed\_l1”: The lower bound of outliers around the observed value
  + “recomposed\_l2”: The upper bound of outliers around the observed value

1. Monitoring error Log – For all apps we have logs that comes in on a regular basis at certain frequency. Specific queries are written to pull the logs count for different apps. The error logs count follows a trend depending on regular scheduled maintainace activities or outages. But any new errors introduced due to code changes or unforeseen outages the number of alerts go up significantly causing in an alert being generated.
2. A dump of ims and the logs. Ims as the dependent & Logs as independent. Divide the logs into 15 mins chunks and run a supervised model. Now monitor logs at 15 mins intervals and if same series of logs have occurred as in the training set. Im is probable to come.

### Twitter Feed Analysis 2018

* Extracting IOCs and cyber intelligence from twitter API to expand internal threat indicator repository.
* Up-to-date on zero day vulnerabilities, threats and TTPs.

Twitter provides api to pull tweets for certain hastags / time line or all . using this certain hastags were monitored to see if new information of IOCs can be obtained.

### Email Mining and Analysis 2018

* Mail scrapping and NLP to categorize, analyze and prioritize work items out of thousands of mails a day.
* **10hrs/day** savings by triggering automations after deriving information from mail.

A python script to extract all the mails. Divide them according to subjects first . later apply clustering on each subdivision. Once clusters are formed they are sorted based on priority. And resolution is recommended from knowledge base.

### Security Patching Self Service Tool 2017

* A shiny app to provide a self-service tool for users to manage security patches on hundreds of systems.
* Saved **30hrs/week** time for the users.

ReactivePoll is used. check and value function

Modal function

Api can be called using httr . Or system2 command. Using curl.

### Tableau Reports 2017

* Tableau reports of compliance reports, trend analysis, gamification leaderboard and incident management.

# Associate System Engineer - Fidelity Investments July 2014 – Dec 2016

### Incident Resolution Recommendations Dashboard 2016

* A shiny dashboard to recommendation possible solutions to an incident using historical data was created.
* Reduced time of resolution of incidents across teams.

Many incidents that come to teams are repeated. An IM might have been solved by an engineer after much research. But this info might not be readily available for others to use. This tool recommends possible solutions of the similar incidents that happened in the past. This significantly reduces much of research time.

More than just recommend solutions, if a certain automation is available for a common type of IM, they can be invoked through tool to resolve it on the go.

* **TF: Term Frequency**, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:   
    
  TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).
* **IDF: Inverse Document Frequency**, which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:   
    
  IDF(t) = log\_e(Total number of documents / Number of documents with term t in it).

Jaccard - Jaccard Similarity = |A ∩ B| : |A ∪ B|

Cosine similarity is for comparing two real-valued vectors, but Jaccard similarity is for comparing two binary vectors (sets)

The Cosine similarity could be used to identify plagiarism, but will not be a good index to identify mirror sites on the internet. Whereas the Jaccard similarity will be a good index to identify mirror sites, but not so great at catching copy pasta plagiarism (within a larger document).

1. similarity takes only **unique set of words** for each sentence / document while cosine similarity takes **total length of the vectors**. (these vectors could be made from bag of words term frequency or tf-idf)
2. This means that if you repeat the word “friend” in Sentence 1 several times, cosine similarity **changes** but Jaccard similarity does not. For ex, if the word “friend” is repeated in the first sentence 50 times, cosine similarity drops to 0.4 but Jaccard similarity remains at 0.5.
3. Jaccard similarity is good for cases where duplication does not matter, cosine similarity is good for cases where duplication matters while analyzing text similarity. For two product descriptions, it will be better to use Jaccard similarity as repetition of a word does not reduce their similarity.

Word2Vec is a method to construct such an embedding. It can be obtained using two methods (both involving Neural Networks): Skip Gram and Common Bag Of Words (CBOW)

CBOW Model:This method takes the context of each word as the input and tries to predict the word corresponding to the context.

SKIP Gram

We input the target word into the network. The model outputs C probability distributions.

### Fidelity Helpdesk Call Analysis 2016

* Importing, cleaning, analyzing and prescribing solutions to the Fidelity call center text data which are in range of 30 million a year using text mining and NLP.
* Helped the management to reduce call volume and find automated solutions for most frequent issues.

8 gb of data was given of the call center call details for Direct Contribution clients. All of it text data. Problem statement was to determine the most common problems / calls that were coming in. The entire corpus was divided into most common problems and their count. Which helped the business to take apt steps to inculcate solutions into the webpage /app.

bigmemory is part of the “big” family which consists of several packages that perform analysis on large data sets. bigmemory uses several matrix objects but we will only focus on big.matrix.

big.matrix is a R object that uses a pointer to a C++ data structure. The location of the pointer to the C++ matrix can be saved to the disk or RAM and shared with other users in different sessions.

By loading the pointer object, users can access the data set without reading the entire set into R.

The following sample code will give a better understanding of how to use bigmemory:

**library**(bigmemory)

**library**(biganalytics)

**library**(bigtabulate)

*#Create big.matrix*

setwd("/Users/sundar/dev")

school.matrix <- read.big.matrix(

"./numeric\_matrix\_SAT\_\_College\_Board\_\_2010\_School\_Level\_Results.csv",

type ="integer", header = TRUE, backingfile = "school.bin",

descriptorfile ="school.desc", extraCols =NULL)

*# Get the location of the pointer to school.matrix.*

desc <- describe(school.matrix)

str(school.matrix)

### Mainframe Batch Job Runtime Prediction 2015

* Run time of a batch cycle is forecasted using Random Forest ensemble model helping mainframe team

**Save 10K dollars** on optimizing the mainframe availability for a batch cycle.

Random forest was used. As it was institutional data there was a good trend. All features were extracted and lag terms were added. First the entire batch cycle run time and volume is predicted. Then sequential batch jobs runtime was predicted.

randomForest(x, y=NULL, xtest=NULL, ytest=NULL, ntree=500,

mtry=**if** (!is.null(y) && !is.factor(y))

max(floor(ncol(x)/3), 1) **else** floor(sqrt(ncol(x))),

replace=TRUE, classwt=NULL, cutoff, strata,

sampsize = **if** (replace) nrow(x) **else** ceiling(.632\*nrow(x)),

nodesize = **if** (!is.null(y) && !is.factor(y)) 5 **else** 1,

maxnodes = NULL,

importance=FALSE, localImp=FALSE, nPerm=1,

proximity, oob.prox=proximity,

norm.votes=TRUE, do.trace=FALSE,

keep.forest=!is.null(y) && is.null(xtest), corr.bias=FALSE,

keep.inbag=FALSE,

**ntree**

Number of trees to grow. This should not be set to too small a number, to ensure that every input row gets predicted at least a few times.

**Mtry**

Number of variables randomly sampled as candidates at each split. Note that the default values are different for classification (sqrt(p) where p is number of variables in x) and regression (p/3)

**Nodesize**

Minimum size of terminal nodes. Setting this number larger causes smaller trees to be grown (and thus take less time). Note that the default values are different for classification (1) and regression (5).

**Maxnodes**

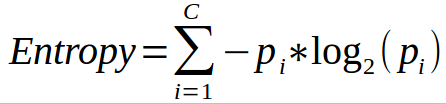
Maximum number of terminal nodes trees in the forest can have. If not given, trees are grown to the maximum possible (subject to limits by nodesize). If set larger than maximum possible, a warning is issued.

**Importance**

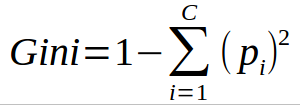
Should importance of predictors be assessed?

**nPerm**

Number of times the OOB data are permuted per tree for assessing variable importance. Number larger than 1 gives slightly more stable estimate, but not very effective. Currently only implemented for regression.



* Favors splits with small counts but many unique values.
* Weights probability of class by log(base=2) of the class probability
* A smaller value of Entropy is better.  That makes the difference between the parent node’s entropy larger.
* Information Gain is the Entropy of the parent node minus the entropy of the child nodes.
* Entropy is calculated [ P(class1)\*log(P(class1),2) + P(class2)\*log(P(class2),2) + … + P(classN)\*log(P(classN),2)]



* Favors larger partitions.
* Uses squared proportion of classes.
* Perfectly classified, Gini Index would be zero.
* Evenly distributed would be 1 – (1/# Classes).
* You want a variable split that has a low Gini Index.
* The algorithm works as 1 – ( P(class1)^2 + P(class2)^2 + … + P(classN)^2)