setwd("/Users/Kevin/Documents/Business Analytics/Bachelor Jaar 2/Machine Learning")

library(dplyr)

library(ggplot2)

library(ggmap)

library(readr)

library(RgoogleMaps)

library(cluster)

library(lubridate)

library(e1071)

library(kknn)

library(class)

# trains

sample = read.csv("sampleSubmission.csv")

train = read.csv("train.csv")

test = read.csv("test.csv")

# Cleaning The Datasets

train$Y = as.numeric(train$Y)

train = train[train$Y != 90, ]

# Separating the attributes in the train.

dates = as.POSIXct(train$Dates, "%Y-%m-%d %H:%M:%S")

category = as.character(train$Category)

descript = as.character(train$Descript)

dayOfWeek = as.character(train$DayOfWeek)

dayOfWeek = as.integer(factor(train$DayOfWeek, levels = c("Monday", "Tuesday", "Wednesday",

"Thursday", "Friday", "Saturday", "Sunday")))

pdDistrict = as.character(train$PdDistrict)

resol = as.character(train$Resolution)

address = as.character(train$Address)

X = as.numeric(train$X)

Y = as.numeric(train$Y)

# Give features Discrete values

labels = unique(category)

labels

nr.crimes = length(labels)

nr.crimes

Target = factor(category, labels = c(1:39), levels= labels)

Target

district = unique(pdDistrict)

district

pdDistrict = factor(pdDistrict, labels = c(1:10), levels = district)

pdDistrict

address.unique = unique(address)

address.unique

Address = factor(address, labels = c(1:length(address.unique)), levels = address.unique)

Address

# Location Attributes

lon = X

lat = Y

lat.range = range(lat)

lon.range = range(lon)

lat.mean = mean(lat)

lon.mean = mean(lon)

# Feature Extraction

str(dates)

dateTime = data.frame(

Date = format(dates, "%Y-%m-%d"), Year = year(dates), Month = month(dates), day = day(dates),

Time = format(dates, "%H:%M:%S"), Hour = hour(dates), Minute = minute(dates), Second = second(dates)

)

# Create a matrix of the Features.

features = data.frame(train$Dates, dateTime$Date, dateTime$Year, dateTime$Month, train$DayOfWeek, dateTime$Time, dateTime$Hour, dateTime$Minute, train$PdDistrict, train$X, train$Y, train$Address)

colnames(features) = c("Dates", "Date", "Year", "Month", "DayOfWeek", "Time", "Hour", "Minute", "PdDistrict", "Longitutde", "Latitude", "Address")

input = data.frame(dateTime$Year, dateTime$Month, dayOfWeek, dateTime$Hour, dateTime$Minute, district, X, Y, category)

colnames(input) = c("Year", "Month", "DayOfWeek", "Hour", "Minute", "PdDistrict", "Longitutde", "Latitude", "Category")

# input

input = data.frame(dateTime$Year, dateTime$Month, dayOfWeek, dateTime$Hour, dateTime$Minute, pdDistrict, X, Y, Address, Target)

colnames(input) = c("Year", "Month", "DayOfWeek", "Hour", "Minute", "PdDistrict", "Longitutde", "Latitude", "Address", "Category")

## 75% of the sample size

smp\_size = floor(0.75 \* nrow(input))

## set the seed to make your partition reproductible

set.seed(123)

train\_ind = sample(seq\_len(nrow(input)), size = smp\_size)

X.train = input[train\_ind, ]

X.test = input[-train\_ind, ]

# Models

# Use tune for Cross-Validation to determine best cost.

svm.model = svm(Category~., data = input, kernel = "linear", cost = 10, scale = FALSE)

plot(svm.model, input)

tune.out = tune(svm, Category~., data = input, kernel = "linear", ranges =

list( cost = c(.001, 0.01, 0.1, 1, 5, 10, 100)))

naiveBayes.model = naiveBayes(Category ~., data = input)

naiveBayes.model

summary(naiveBayes.model)

prediction = predict(naiveBayes.model, as.data.frame(input))

summary(prediction)

table(prediction, input$Category)

kNearestNeigh.model = kknn(formula = formula(train), train )

# Actual Models

input = data.frame(dateTime$Year, dateTime$Month, dayOfWeek, dateTime$Hour, dateTime$Minute, pdDistrict, X, Y, Target)

colnames(input) = c("Year", "Month", "DayOfWeek", "Hour", "Minute", "PdDistrict", "Longitutde", "Latitude", "Category")

model = naiveBayes(input$Category~., data = input)

predictionModel = predict(model, input[,1:8])

tab = table(predictionModel, input$Category)

summary(predictionModel)

accuracy.nb = sum(tab[row(tab)==col(tab)])/sum(tab)

accuracy.nb

#K Nearest Neighbour

class = as.factor(X.train[['Category']])

knn.model = knn(train = X.train, test = X.test, cl = class, k = 100)

knn.modelcv = knn.cv(train = X.train, cl = class, k = 10)

attributes(.Last.value)

summary(knn.model)

tab.kn = table(knn.model, X.test[['Category']])

accuracy.k100 = sum(tab.kn[row(tab.kn)==col(tab.kn)])/sum(tab.kn) # Accuracy

accuracy.k100

# Log Loss

MultiLogLoss <- function(act, pred)

{

eps = 1e-15;

nr <- nrow(pred)

pred = matrix(sapply( pred, function(x) max(eps,x)), nrow = nr)

pred = matrix(sapply( pred, function(x) min(1-eps,x)), nrow = nr)

ll = sum(act\*log(pred) + (1-act)\*log(1-pred))

ll = ll \* -1/(nrow(act))

return(ll);

}

# Map Plotting

SFMap = qmap(location = "san fransisco", zoom = 12, color = "bw", source = "osm")

SFMap

SFmap = GetMap(center = c(lat.mean, lon.mean), zoom = min(MaxZoom(lat.range, lon.range)), destfile = "sf.png")

PlotOnStaticMap(SFmap, lat, lon, destfile = "sf.png", cex = 1, pch = 20, col = "red")

####### FENCE #######

# What patterns or interesting findings emerged in the train?

# - Regions that (certain) crimes concentrate in

# - Time Interval in which the most crimes occur

# - Are there correlations between crimes and time of day/day of the week?

# The main objective is to predict the class of the crimes.

# The task is obviusly a multi-class classification problem.

# Therefore, we could use the classification algorithms like:

# \* Support Vector Machines

# \* Naiïve Bayes

# \* k-NN (k-Nearest Neighbour)

# \* Gradient Tree Boosting

# \* etc.

# It depends on what Hypothesis we choose, and if we want our algorithm to learn under supervision or not.

# Obviously we cannot use Support Vector Machine as, SVM has some limitations.

# Speed and size is definitely one of the biggest limitation of SVM.

# "The most serious problem with SVM is the algorithm complexity

# and extensive memory requirements of the required quadratic programming in large-scale tasks."

# Horváth (2003) in Suykens et al. p 392

# SVMs are hard to scale to large datasets, using our training set of 800,000 entries

# the SVM will not be able to fit.