Ref:

1. Buda, M., Maki, A. and Mazurowski, M. A. (2017) ‘A systematic study of the class imbalance problem in convolutional neural networks.’ *Neural networks.*106, pp.249-259.
2. Chawla, N. V., Bowyer, K., Hall, L. O. and Kegelmeyer, W. P. (2002) ‘SMOTE: synthetic minority over-sampling technique’ *Journal of Artificial Intelligence Research.* 16(1), pp.321-357.
3. He, H., Bai, Y., Garcia, E. A. and Li, S. (2008) ‘ADASYN: Adaptive synthetic sampling approach for imbalanced learning’ *2008 IEEE International Joint Conference on Neural Networks (IJCNN).* Pp.1322-1328.
4. Thabtah, F., Hammoud, S., Kamalov. F. and Gonsalves, A. (2020) ‘Data imbalance in classification: Experimental evaluation’ *Information Science.* 513(1), pp.429-441.
5. Zhang, S., Li, X., Zong, M., Zhu, X. and Cheng, D (2017). ‘Learning k for kNN Classification’ *ACM Transactions on Intelligent Systems and Technology.* 8(3), pp.1-19.

According to the original dataset, a significant imbalance of classes of data is observed, where in the total 6819 cases, 6599 (96.77%) of which are negative while only 220 (3.23%) is positive. Therefore, it is essential to solve the data imbalance issue in case the model is not trained with bias.

Thabtah et al (2020) suggested a framework of data imbalance solutions. From the data’s aspect, there are two notions to rebalance data, which are under sampling and over sampling. Under sampling means to reduce the cases of major class, while over sampling is to generate synthetic data of minor class till the two classes are balanced. Considering the minor class of this dataset only has 220 cases, using the over sampling method is recommended.

Common over sampling methods include random over sampling, SMOTE, ADASYN, etc (Chawal et al, 2002; Buda et al, 2017; He et al, 2008). Random over sampling method replicates the minor cases randomly, which generates same data and causes bias. SMOTE is based on nearest neighbour algorithm and generates data that are not exactly same as the neighbour cases. ADASYN, also based on nearest neighbour algorithm, focuses more on difficult examples (points at the border). This is considered a main difference to SMOTE, which could reduce bias from the data. Therefore, in this project we use ADASYN to solve data imbalance.

As mentioned above, ADASYN is based on nearest neighbour algorithm to choose and thus generate data. Therefore, same to KNN classification, the performance of ADASYN is essentially influenced by *k* value. To find the optimal *k* value for this project, we applied simple SVM model to the training set data and compare the performances of model by setting *k* = 3, 5, 7, 9…73. The *k* values are odd numbers because according to He et al (2008), an even number of nearest neighbours may cause difficulty to decide the class of data when not having majority class of neighbours. Note that according to Zhang et al (2017), a typical *k* value for ADASYN model is the square root of data size that requires to be augmented. In this case, it equals the square root of 5073, equalling 71.22, approximately 71. The accuracy is tested based on cross validation, by setting the number of folds equalling 5. Each augmented train set is fitted into a default Support Vector Machine for calculating the accuracy. Below are the steps of entire solution.

Step 1: data preparation

set.seed(123)

# load datasets

train = as.data.frame(read.csv("processed\_train\_final\_v2\_with\_18\_variables.csv"))

test = as.data.frame(read.xlsx("selected\_test.xlsx"))

# list of k values

k <- seq(3, 73, by = 2)

# empty list to store accuracy

accuracy <- list()

step 2: generate ADASYN augmented data and fit into SVM

# apply cross validation

trControl <- trainControl(method = 'cv', number = 5)

# apply adasyn to train set and use svm to test the performance of changing k value

for (i in 1:36) {

# apply adasyn and generate rebalanced train set

adasyn <- ADAS(train, target = train$y1, K = k[i])

adasyn\_train <- subset(adasyn$data, select = -class)

adasyn\_train$y1 <- as.factor(adasyn\_train$y1)

# fit the rebalanced train set to svm model

# all hyperparams are set default

model <- train(y1 ~ .,

data = adasyn\_train,

trControl = trControl,

method = "svmLinear",

)

accuracy <- append(accuracy, model$results$Accuracy)

}

Step 3: interpret the optimal k value

# reshape the accuracy list into 2-cols data frame

# col 1: k values; col 2: accuracy

accuracy <- transpose(as.data.frame(accuracy))

colnames(accuracy) <- "accuracy"

k <- as.data.frame(k)

colnames(k) <- "k value"

accuracy <- cbind(k, accuracy)

# k value of max accuracy and max accuracy

accuracy$`k value`[which(accuracy$accuracy == max(accuracy$accuracy), arr.ind = T)]

max(accuracy$accuracy)

The result of the three steps is shown in Fig1 below.

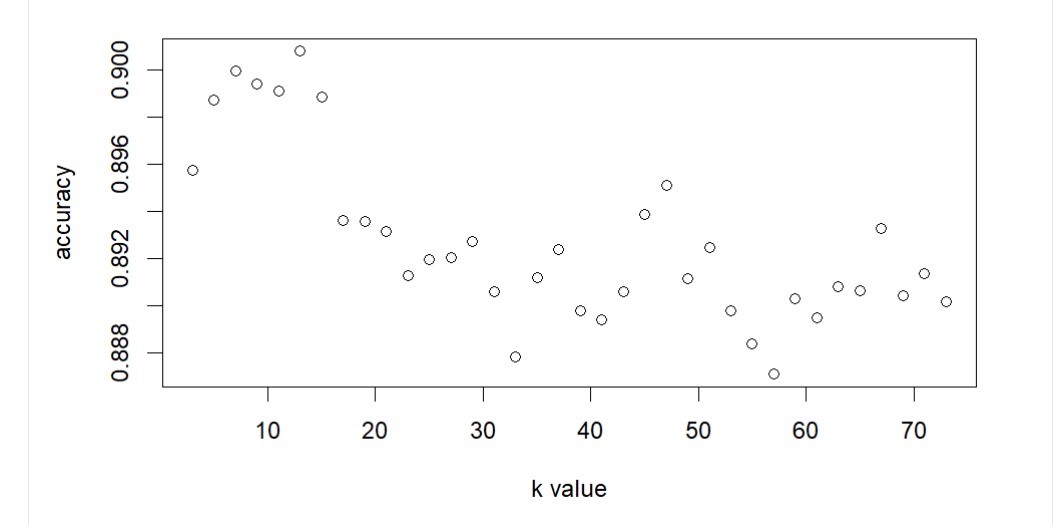


Fig 1: accuracy to k value scatter plot

From Fig 1 we can see that the maximum accuracy occurs when *k* = 13, at 0.9008. Therefore, *k* = 7 is picked for ADASYN. We apply ADASYN with *k* = 13 only to train set and get a training set of 5275 class 1 and 5309 class 2 cases, and the proportion of two classes reaches 99.36%.