Analysis of UMM Student Retention Data using Support Vector Machines (SVM) and Tidymodels

Elmurad Abbasov

April 29, 2022

Abstract

Retention data is an important piece of every developed organization to be in compliance with the federal law. It is a continued storage of an organization's data for business records. University of Minnesota Morris (UMM) is an example of a developed and established organization, on a university, that collects retention data for legal reasons. In this project, a specific Machine Learning model, Support Vector Machines (SVM) is going to be used to analyze UMM retention data. In additions, the model is going to be executed with help of Tidymodels packages. The analysis is going to be performed using R language.

Introduction

Because this study is built on analyzing the University of Minnesota Morris (UMM) retention data using Support Vector Machines (SMV) and Tidymodels set of packages, it is necessary to underline several definitions before stepping into the analysis part of this paper. To be specific, a description of Machine Learning, Tidymodels, and relevant methods will be discussed further.

Machine Learning (ML) is a branch of Artificial Intelligence (AI) that is focused on statistical models and algorithms that are meant to learn from its own patterns, and such learning process in Machine Learning is called training. By training on its own patters and features in the data, ML models gradually improve their accuracy. Generally speaking, ML algorithms can be broken down into three main parts: a decision process, where a machine learning algorithm is either classifying or predicting based on patterns in some input data, an error function, which evaluates the prediction and accuracy of the model, and an model optimization process, which is the process of adjusting hyper-parameters to minimize the cost function.

University student retention is one of the most important criteria for the college admissions to consider when trying to improve graduation rates and decrease a loss of tuition revenue that either drop our or transfer to another school. Besides university admissions office, students are also interested in learning more about retention data because it would help them to understand if their time and resources are worth to invest in that specific college degree. For those purposes, universities are collecting lots of data from students once they apply to their institution. To clarify, as the National Center for Education Statistics (NCES) stated, retention rates refer to the proportion of students returning to the same institution the following fall, while graduation rates are students who complete the programs in certain amount of time.

Machine Learning Background

There are currently three main types of Machine Learning algorithms. **Supervised, Unsupervised, and Reinforcement learning**. In **Supervised Learning**, the data that is used for training ML models is already labeled and defined. There are two main categories of supervised machine learning - classification, which is an algorithm that assigns test data into specific categories, and regression, which is a set of statistical processes for estimating the relationship between dependent and independent variables. ~IBM Cloud Education (2020) The weights of the model adjust as input data is fed into the model, and this is also known as a cross validation process. For supervised learning, since the data set is already labeled and includes inputs and correct outputs, the model is able to learn over time and estimate accuracy through the loss function until the error is minimized. This type of Machine Learning is also known to be task driven, such as for predicting the next value.

In **Unsupervised Learning**, the models are used to analyze unlabeled data sets searching for patterns to cluster similar data types which helps to solve association problems.~uns (2020) To group data based on their similarities or differences, unsupervised learning algorithm uses clustering, which is a technique that inputs raw and unclassified data and clusters data into groups based on common patterns. This type of Machine Learning is also known to be data driven since it searches for patterns to identify clusters.

In **Reinforcement Learning**, we have an environment with intelligent agents that try to maximize

the notion of cumulative reward. It is based on rewarding desired behaviors and punishing undesired ones. We can think of an example with an agent that is able to recognize and perceive its environment and learn through trial and error using its own experience.~Bhatt (2019) Although both Supervised Learning and Reinforcement Learning share the same idea of connecting inputs and outputs, the differ in a way that Supervised Learning uses the training data has the answer key already, but in Reinforcement Learning the agent makes a decision on how to perform based on its experience. Another comparison is the in Reinforcement Learning decisions are dependent, where an output depends on the state of the current input, such as when playing chess, but in Supervised Learning decisions are independent and made on the initial input, such as how object recognition works. The following figure illustrates the action-reward feedback loop in a typical Reinforcement Learning model.

The above mentioned Machine Learning type require training for their models to learn patterns perform the best. Usually, the data set is divided into two different parts, where one part is used for training itself, and the other part is used for testing. After the ML model has gone through training, we want to test our model on the data that our model has not seen before to compare the performance and make sure that it can still solve out problem even with some different data.

The Machine Learning model that will be used in this study is known as a supervised algorithm. Support Vector Machines (SVM) is an algorithm that analysis data for classification and regression analysis. It is also known to be a non-probabilistic binary linear classifier. It works by assigning data two one of two categories, and then a Support Vector Machine builds a model that assigns new example two a specific category. SVM is attaching unclassified data to points in space as to maximize the width of the gap between the two categories. Based on which side of the gap they fall, those data points are mapped in the corresponding space, which is supposed to be an already established category or class as we are using labeled data.

The models in this study will be executed with the help of Tidymodels package. It is a collection of packages for modeling and machine learning using tidyverse principles that share the underlying design and structure.

Feature Selection

Because of the large amounts of data, proceeding such data can be often inefficient. The feature selection in a process that is very important for high dimensional data sets.

During a feature selection process, an algorithm is selecting a subset of the original features so that the feature space is optimally reduced to the evaluation criteria; a feature selection method selects a subset of relevant features. Kohavi et al. Kohavi and John (1997) describes features to be either strongly relevant or weakly relevant, where in strongly relevant feature removal of s deteriorates the performance of the algorithm. A feature is called weakly relevant if removal of a subset of features containing s deteriorates the performance of the algorithm. In addition, if a feature is neither strongly nor weakly relevant, then it is irrelevant

Hyperparameter optimization

For us to define **hyperparameter optimization**, we first need to define what is a **parameter** and a **hyperparameter**.

- Model parameters are the set of configuration variables that are internal to the model and can be learned from the training data, and the value of those parameters is estimated from the input data. Model parameters specify how input data is transformed into the desired output.
- Model hyperparameters are the set of configuration variables that are external to the model, they have a value that is used to control the learning process in the model, and they cannot be directly trained from the input data. Model hyperparameters define the structure of the model.
- Hyperparameter optimization is the process of finding the most optimal hyperparameters for the learning in Machine Learning algorithm. To generalize different data patterns, a hyperparameters set could be used which also should be optimized so that the Machine Learning model can solve tasks with the highest efficiency.

Data Description

Data, in the context of Machine Learning is the most important unit which is used to predict and classify models. With a sufficient data set, Machine Learning algorithms can be trained based on the similar features or patterns in the data. Typically, the size of the data is an important factor that determines the accuracy of the ML algorithm. There are two types of the data - labeled, where there is already a classified label for each input, and unlabeled, where data set does not have specific labels but only input data.

```
## [1] "dsML"
```

```
## Rows: 1,982
## Columns: 12
## $ EGDesc               <fct> AmIndian, AmIndian, AmIndian, White, AmIndian, Wh~
## $ EntryUMCrs           <dbl> 8, 0, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ RetEvent1            <chr> "NonEvent", "NonEvent", "Event", "Event", "Event"~
## $ ACT_Flag             <chr> "ACTPresent", "ACTPresent", "ACTPresent", "ACTPre~
## $ COMP_ACT_SCORE       <dbl> 13, 16, 20, 29, 25, 31, 20, 0, 15, 19, 25, 22, 21~
## $ AcadProg2            <chr> "Phys", "Mgmt", "Compt Sci", "Math", "Psy", "Phys~
## $ ENTRY_REG_STA        <chr> "NAS", "NAS", "NAS", "NAS", "NAS", "NAS", "NAS", ~
## $ ENTRY_LOAD           <chr> "F", "F", "F", "F", "F", "F", "F", "F", "P", "F",~
## $ ENTRY_CR_LOAD        <dbl> 13.0, 12.0, 18.0, 18.0, 14.0, 13.0, 12.0, 12.0, 7~
## $ Entry_Cr_Flag        <chr> "EntryCrPositive", "EntryCrPositive", "EntryCrPos~
## $ ENTRY_TOT_CUMULATIVE <dbl> 63.00, 62.00, 12.00, 17.00, 144.50, 87.00, 55.00,~
## $ HomeLocCode2         <chr> "SD", "SD", "MorrisArea", "OtherUSA", "HennRams",~
```

```
##
##    Event NonEvent
##     1498      484
```

This data comes from UM central retention data reports. This data is already labeled and was prepared by the Professor Jon Anderson and other researchers, where they have grouped some variables. For example, the variable *AcadProg2* was created to group academic programs by department with the largest twenly programs left ungrouped. The *HomeLocCode2* variable is a categorical variable that includes Minnesota county codes into specific areas, such as Hennepin

Ramsey area, St. Cloud Area, Morris Area, etc. Students from states bordering Minnesota are labeled as their states, students from outside those specific states are labeled as OtherUSA, and international students are labeled as such.

Variable explanations:

- EGDesc: the student's ethnic group description
- RetEvent1: retention variable, either an "event" in which the student returned or a "nonevent" if the student did not, this will serve as the response variable for our analysis
- ACT_Flag: this is a flag variable which flags if the student does not have an ACT score
- AcadProg2: the student's academic program (major), divided as described above
- ENTRY_REG_STA: the student's standing upon entry, either from high school or a transfer student (NHS means from high school, NAS means transfer)
- ENTRY_LOAD: the student's credit load status, either full time (F) or part time (P)
- Entry_Cr_Flag: this is a flag variable which separates the students that entered with credits from those who did not
- HomeLocCode2: the student's home location, divided as described above
- EntryUMCrs: how many University of Minnesota credits the student has at entry
- COMP_ACT_SCORE: the student's composite ACT score, for students without an ACT score, this is 0
- ENTRY_CR_LOAD: the student's credit load after 2 weeks into the semester of entry
- ENTRY_TOT_CUMULATIVE: This is the student's total cumulative credits they are entering with.

Methodology

The Support Vector Machine (SVM) algorithm can be used for both classification and regression purposes, but we are going to focus on classification performance. Given that n is a number of features we have, the algorithm classifies data by plotting each data item as a point in n-dimensional space with the value of each feature being the value of a particular coordinate. The classification is done by finding the hyper-plane that differentiates the two classes the best. In other words, SVM classifier is a frontier that best segregates the two classes. In addition, for more complex problems that involve more than two classes, the SVM algorithm creates a mapping space to separate the input data in different classes in a linear or non-linear data by developing kernel functions that can transform the inputs to a higher dimensional space for linear separability, or in other words it converts not separable problem to separable problem based on the defined labels or outputs. The overall goal of a Support Vector Machine algorithm is to find a hyperplane which could separate the data accurately, or to be more specific, find the optimal separating hyperplane which maximizes the margin of the training data ~svm (2020).

Let's assume that we have a data that needs to be separated/classified by the algorithm. The given data that needs to be classified is represented as a unique point in space where each point is represented by some feature vector x where RˆD with D dimension.

$$x \in R^(D)$$

To illustrate the hyperplane that separates data to specific classes, please refer to the following

figure 1. In the figure below we can see data points represented in space where we have the size and weight of several people and the algorithm would have to separate data points, or people, by gender. This is a linearly separable data as in this case we can select two paralle hyperplanes that separate the two classes of data that achieve the maximum distance possible, and this method is also called **hard-margin**.

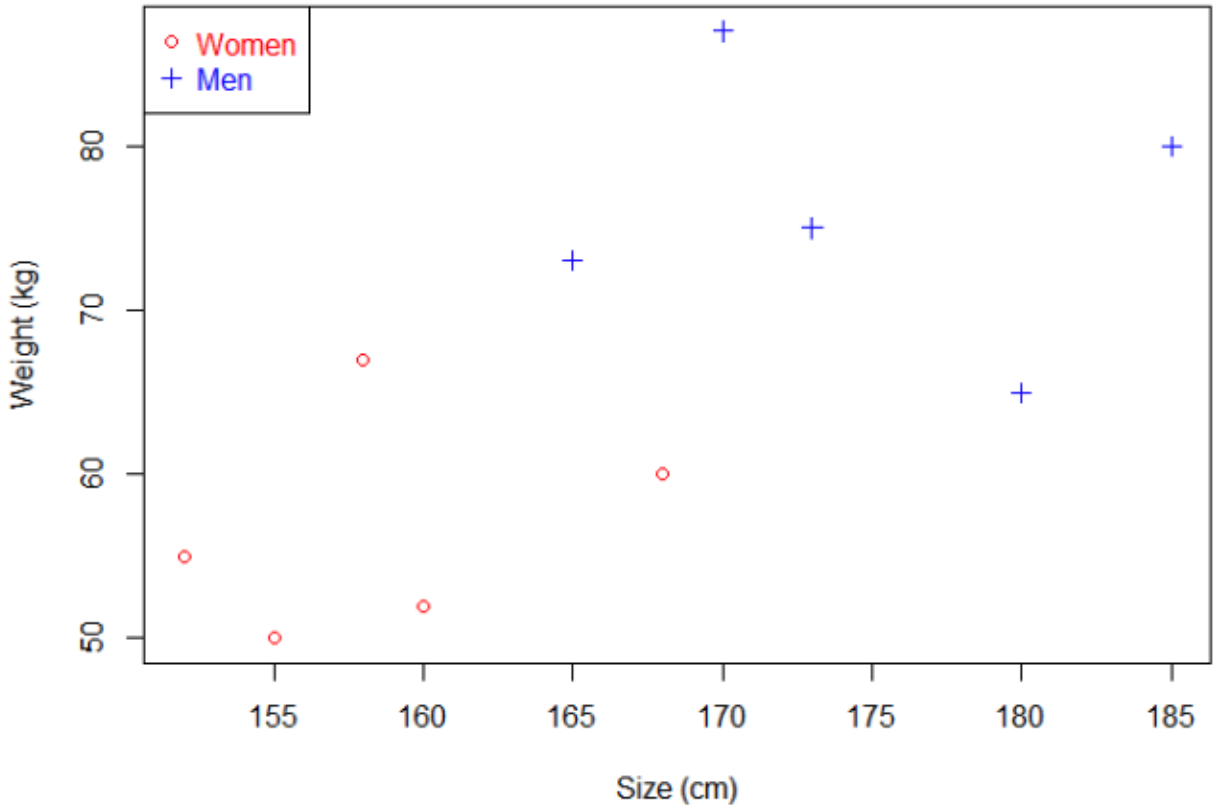

Figure 1: An example of data points in 2-D space

In this example we can assume that men, for example, are data points represented in blue, and women in red. To separate them, we could trace a line that would separate them equally, and such a lane is called a separating hyperplane. A Support Vector Machine can work in any number of dimensions. In only one dimension, for example, a hyperplane is called a point. In two dimensions, a line, and it is called a plane in three dimensions. In more than three dimensions such a line is called a hyperplane. As it was mentioned earlier, the purpose of the SVM is not just to find a hyperplane that would classify data into classes, but to find the most optimal, the best hyperplane of all possible options by maximizing the margins.

Looking at the figure 2 below, we can try choosing a green hyperplane. With the green hyperplane we see that it wrongly classified three women and thus it might not be the best option. Our goal is to choose a hyperplane that is as far from data points as possible from each category, such as a black hyperplane below. This hyperplane is optimally the best because it has the maximum distance, or margins, from the data points, and thus it is more likely to correctly classify the training data and more likely to perform better with unseen data.

To find the margin we can compute the distance between the hyperplane and the closest data point,

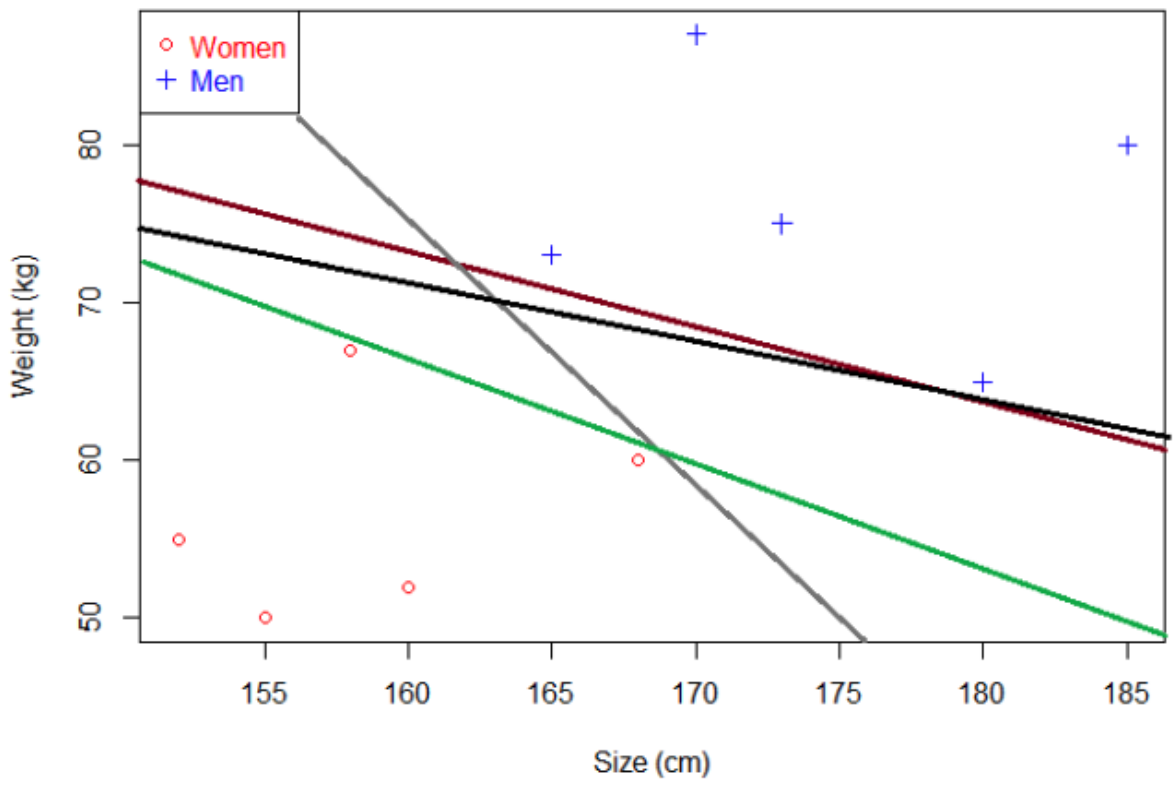

Figure 2: A variation of hyperplanes in space

and once we double the computed value we will get the margin. The optimal condition defined by the model is found by minimizing the Euclidean norm of the weight vector. The equation of the hyperplane is, in fact, the same as the equation of a line y=ax+b and follows as:

$$w^T x = 0$$

We are using this equation instead of the formula for a line because vector **w** will always be normal to the hyperplane, and it is easier to work in more than two dimensions with this notation. To find the margin, we need to find the maximum distance between the hyperlane and a data point. Let's assume the following figure 3 as an example:

In this example, the equation of the hyperplane is

$$x_2 = -2x_1$$

which is essentially the equation of the hyperplane if we see it from the perspective of vectors

$$\mathbf{w}\begin{pmatrix}2\\1\end{pmatrix}\mathbf{x}\begin{pmatrix}x_1\\x_2\end{pmatrix}$$

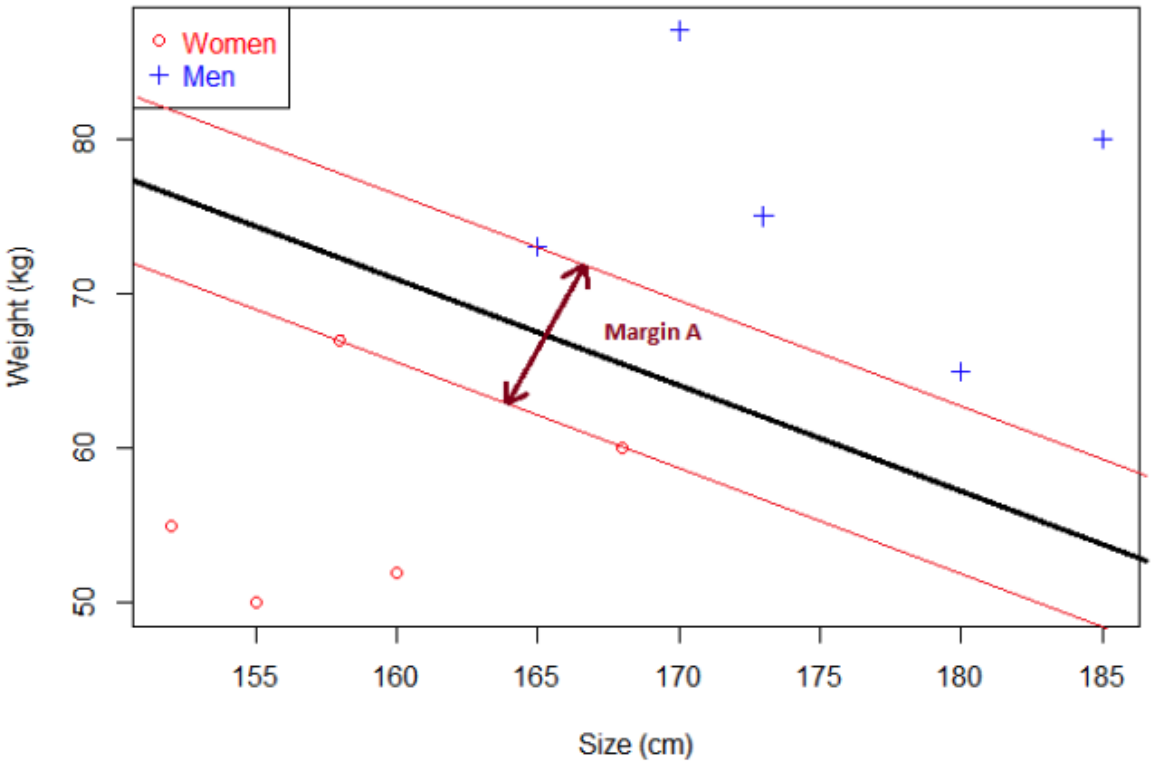

Figure 3: Representation of the margin

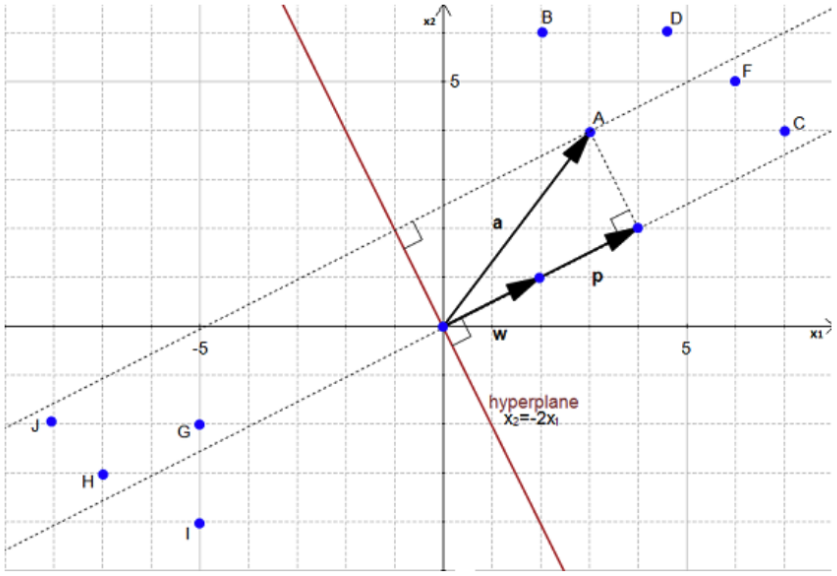

Figure 4: Distance between a hyperplane and data point

The next step would be to find the distance between point A(3,4) and the hyperplane. If we project A, assuming it is a vector, to the normal vector **w**, we will get the vector p, which is the distance

we are looking for. If the normal vector w is (2,1) and **a** is (3,4) then the distance would be the orthogonal projection of **a** onto **w**, where **u** is the direction of w:

$$\|w\| = \sqrt{2^2 + 1^2} = \sqrt{5}$$

$$p = (u * a)u \rightarrow p = \frac{10}{\sqrt{5}}u \rightarrow p = (4,2) \rightarrow \|p\| = 2\sqrt{5}$$

To compute the margin, all we have to do is double the distance between A and the hyperplabe that we just found, and the result would be:

$$margin = 2\,\|p\| = 4\sqrt{5}$$

But even though we were able to find a hyperplane that separated our data and even computed the margins, it does not mean that this is the optimal hyperplane, so our next goal would be to find a hyperplane that has the biggest margins. Below is an example of an optimal hyperplane, where we have three hyperplanes, two of which are on opposing polar sides on the plane svm (2015).

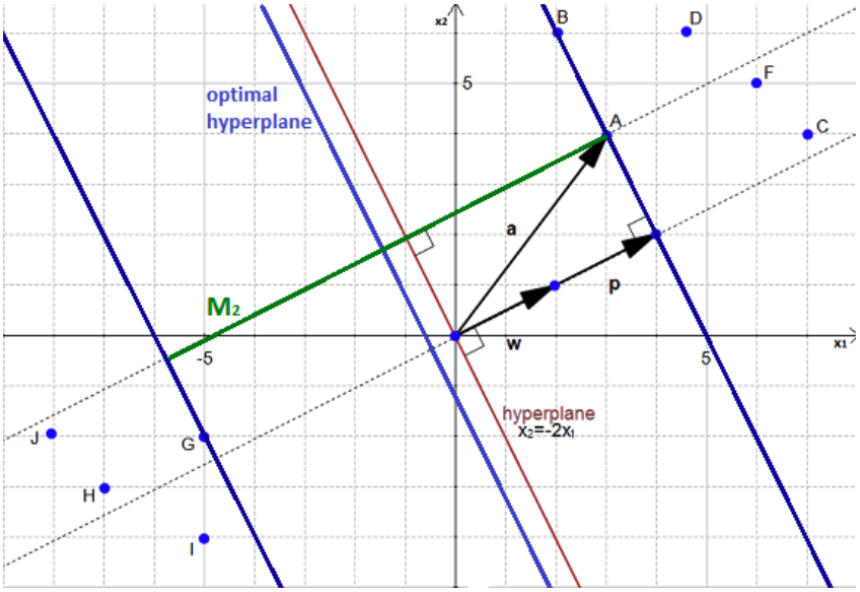

Figure 5: The optimal hyperplane

We have selected the furthest possible hyperplane to the left, crossing point G, and to the right, crossing the point A. From those extreme points, since we already know the margin M2, we can trace a line crossing M2 in the middle and thus will see the optimal hyperplane. As we can see, there is a clear correlation between margins and hyperplanes, as finding the biggest margin is the same as finding the optimal hyperplane. To find the biggest margin we need to select two hyperplanes that separate data with no points between them and maximize their distance. Let's imagine we have the following dataset:

$$D = [(\mathbf{x}_i, y_i)|\mathbf{x}_i \in \mathbb{R}, y_i \in (-1,1)]_{i=1}^{n}$$

Here, our data consists of n number of p-dimensional x_{i} vectors where each x_{i} is associated with a value y_{i} indicating if the element belongs to the class (+1) or not (-1). For simplicity, let's also assume that the data is linearly separable in some p-dimension as illustrated on the figure below:

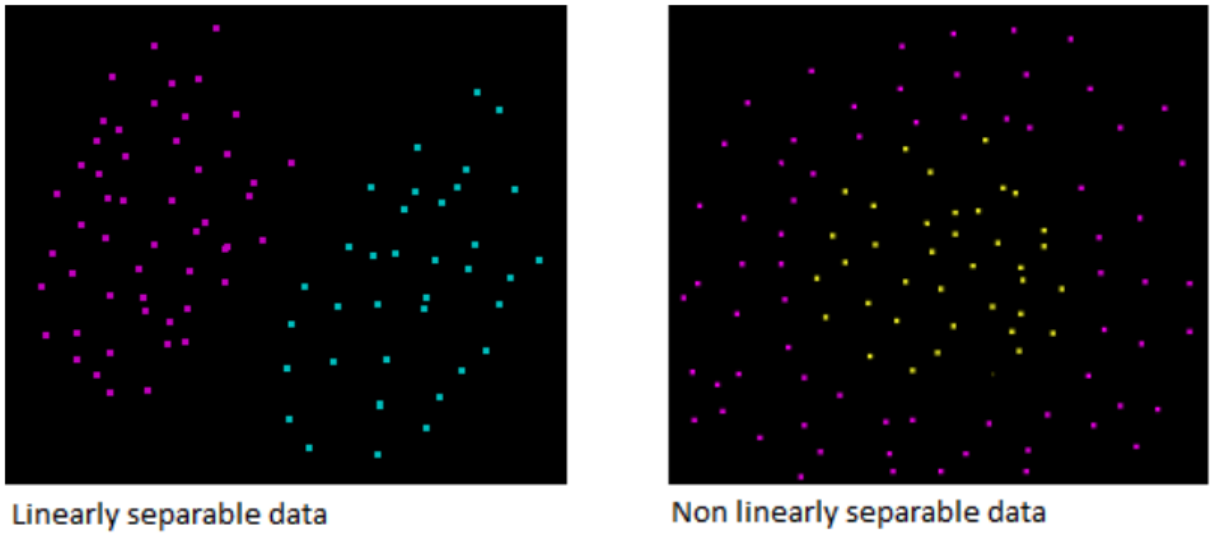

Figure 6: Linearly vs non-linearly separable data

If the data is linearly separable, we are coming back to the hyperplane equation that can be rewritten as

$$\mathbf{w}^T\mathbf{x} = 0 \rightarrow \mathbf{w} \cdot \mathbf{x} + b = 0$$

because given two 2-dimensional vectors w(-a, 1) and x(x,y), we could derive

$$\begin{gathered}\mathbf{w}' \cdot \mathbf{x}' = (-a) \times x + 1 \times y \\ \mathbf{w}' \cdot \mathbf{x}' + b = y - ax + b \\ \mathbf{w}' \cdot \mathbf{x}' + b = \mathbf{w} \cdot \mathbf{x} \\ \mathbf{w} \cdot \mathbf{x} + b = 0\end{gathered}$$

And having two hyperplanes H_1 and H_2 trying to separate data, we would have

$$\begin{gathered}\mathbf{w} \cdot \mathbf{x} + b = 1 \\ \mathbf{w} \cdot \mathbf{x} + b = -1\end{gathered}$$

So now the algorithm would try to select two hyperplanes on the different polar sides that satisfy the following criteria:

$$\mathbf{w} \cdot \mathbf{x_i} + b \geq 1 \text{ for } \mathbf{x_i} \text{ having the class } 1$$
$$\mathbf{w} \cdot \mathbf{x_i} + b \leq -1 \text{ for } \mathbf{x_i} \text{ having the class } -1$$

In our previous example, red points would have the class 1 and blue points would have the class -1. From the figure below we can observe the several hyperplane scenarious, colored in black, green, and orange. The orange selection of hyperplanes has the biggest margin, but we can also see that some data points are outside of the margin, which means that the algorithm has classified them with mistakes. The green margin does not have any classification errors, but it also doesn't mean that it is the most optimal choice of hyperplanes as the margin could probably be larger. Finally, if we take a look at the black selection of hyperplanes, we can see that it does not have any classification errors, and also has the largest possible margin that is on the edge of points H, A, and B.

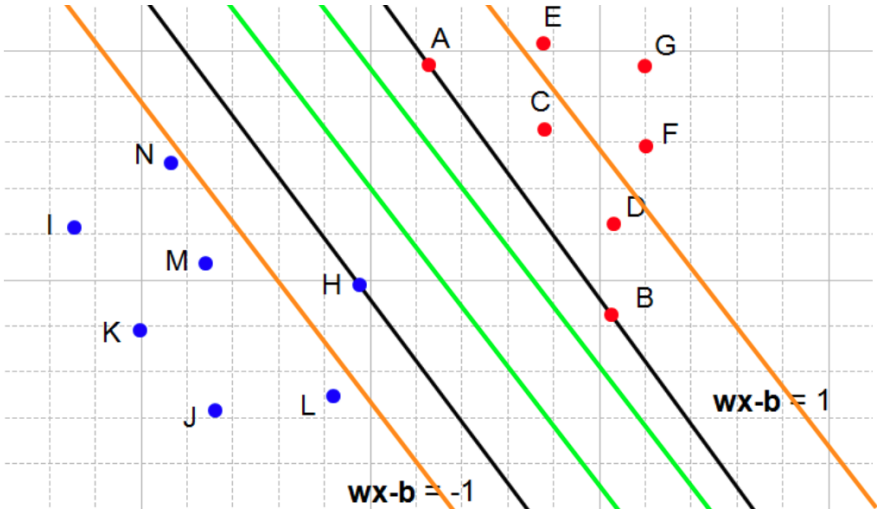

Figure 7: The most optimal hyperplane

In our examples, we were able to find the largest margin and the optimal hyperplane just by trial and error. If we would compute the distance between two hyperplanes mathematically, like the Support Vector Machine algorithm would, we need to set several rules.

Assuming that

$$\mathcal{H}_0 \text{ be the hyperplane with condition } \mathbf{w} \cdot \mathbf{x} + b = -1$$
$$\mathcal{H}_1 \text{ be the hyperplane with condition } \mathbf{w} \cdot \mathbf{x} + b = 1$$
$$\mathbf{x}_0 \text{ be the point in the hyperplane } \mathcal{H}_0$$
$$m \text{ is the margin}$$

If m is the distance between hyperplanes H_0 and H_1, then we need to find the value of m. We can try to find a point on another hyperplane if we transform m to a vector and try to add it to vector x_0. From our equation of a hyperplane we know that a vector perpendicular to H_1 is a vector w, and

another criteria that we need to satisfy is to find a vector that has a magnitude of m. If we assume u to be the unit vector of w, it must have the same direction as w and also is perpendicular to the hyperplane. Let's also assume that if we multiply **u** by m we can get vector **k** = m**u**

$$\mathbf{u} = \frac{\mathbf{w}}{\|\mathbf{w}\|}$$
$$\|\mathbf{u}\| = 1$$
$$\|\mathbf{k}\| = m$$
$$\|\mathbf{k}\| = m \rightarrow m\mathbf{u} \rightarrow m\frac{\mathbf{w}}{\|\mathbf{w}\|}$$

As a result we transformed a scalar m into a vector **k**, and now we can add it to x_0. We can define this addition as z_0 and from there derive a formula to compute m.

$$\mathbf{z}_0 = \mathbf{x}_0 + \mathbf{k} \rightarrow \mathbf{w} \cdot (\mathbf{x}_0 + m\frac{\mathbf{w}}{\|\mathbf{w}\|}) + b = 1$$
$$\mathbf{w} \cdot \mathbf{x}_0 + m\frac{\|\mathbf{w}\|^2}{\|\mathbf{w}\|} + b = 1$$
$$-1 = 1 - m\|\mathbf{w}\| \rightarrow m = \frac{2}{\|\mathbf{w}\|}$$

Now when we know how to compute the margin m, we know how to maximize the distance between two hyperplanes, and the variable that can change in the formula is the norm of **w**. This way, if we plug 1 into **w**, then m = 2, and if **w** is 2, then m is 1. If we keep going, we will see that the larger the norm is, the smaller the margin becomes. Thus, we can conclude that if our goal is to maximize the margin to find the optimal hyperplane, then we need to choose the hyperplane selection with the smallest norm **w**. The following figure taken from Wikipedia serves as a good representation of a vector that is crosses at a point where the margins are maximum. Samples on the margins here are called the support vectors wik (2022).

Non-linear Kernels

So far we have been discussing scenarios where the data is linearly separable, but we should also touch base on non-linear scenarios. To classify data that is non-linearly separable, the researchers are often using what-so-called the **kernel trick**. The algorithm in this case is similar, but the difference is that every dot product is replaced by a non-linear kernel function. So common kernels that were used in this paper include a regular **linear** kernel, a **tuned linear** kernel, a **polynomial kernel**, and a **radial kernel**.

Experiment Setup

For the experiment itself, four kernels in total were used to perform the analysis. To be specific, linear kernel without tuning, linear kernel with tuning, radial kernel with tuning, and polynomial

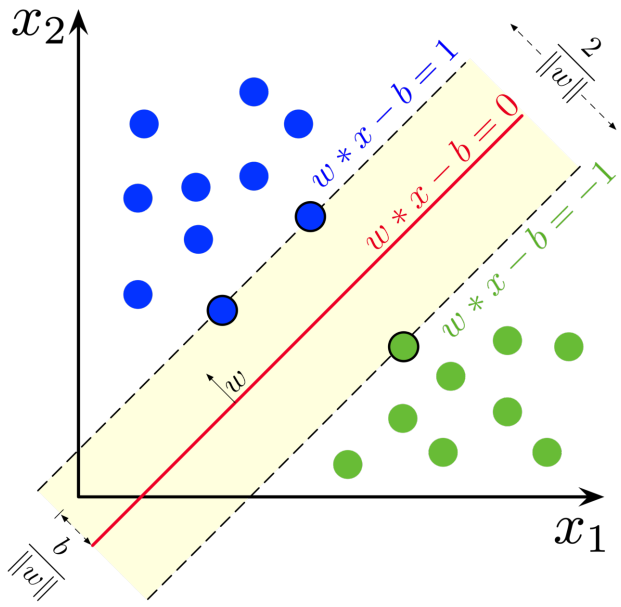

Figure 8: Definition of the maximum-margin hyperplane with samples from two classes

kernels were used to see the accuracy results. At the beginning of the experiment the entire dataset was split into two parts - training and testing data. The training data occupied 70% of all data, and the rest was left for testing. The procedure was done with help of createDataPartition function in CARET package. Empty values in the dataset were ommited.

The Support Vector Machine algorithm was tested with help of CARET package that involved several pre-defined SVM and other machine learning functions. FitContol method was prepared to define 10-fold cross validation criteria for the model. In our parameters for fitControl, we have specified the cross-validation to have 5 repetitions, and included twoClassSummary functions to have AUC pick up the best performance of the model.

For the first kernel, we have used regular parameters of the linear SVM model and trained it on training data. In CARET package, such a method is called *svmLinear*. To make sure variables are normalized and their scale is comparable, we have set such as option using *preProcess* function with center and scale specifications. After that we have predicted the results using testing data. The results were evaluated using a confusion matrix. Once again, the response variable is RetEvent1.

In many cases, tuning a linear SVM model is not necessary, but for educational purposes we have still tried that to see if the results are any better. Grid was expanded on the range from 0.0 to 2.5. In addition, tuneLength parameter was set to 10, which tells the algorithm to try different choice of default values for the main parameter, and then tuneGrid allows the algorithm to decide which value of the main parameter to implement in the model.

For radial kernel, the method in CARET is called *svmRadial* and we have tuneLength set to 5 using a single value of sigma based on sigest function. For polynomial kernel, the method was called *svmPoly* and tuneLength was set to 5 as well. For radial and polynomials kernels, a function bestTune was used to find the most optimal cost value for that specific kernel.

At the end we have also split our RetEvent1 response variable into two categories, event and non-event, and found a proportion of students that will return based on what we already have

in the data. For event, we have 1498 responses, and for non-event we have 484 responses, which results in 1982 total responses. The accuracy in this case would be defined as the number correctly predicted over the total number of responses, or 1498 divided by 1982.

Results

For evaluation the results, metrics such as True Positive (TP), False Negative (FN), False Positive (FP) and True Negative (TN) were used.

- **True Positive (TP)** - an outcome where the model correctly predicts the positive class
- **False Negative (FN)** - an outcome where the model incorrectly predicts the negative class
- **False Positive (FP)** - an outcome where the model incorrectly predicts the positive class
- **True Negative (TN)** - an outcome where the model correctly predicts the negative class

The most important parameter for evaluating the performance of a machine learning algorithm, including Support Vector Machines, is accuracy. Our kernels have presented a similar accuracy, where a linear SVM model with default parameters had accuracy of 76%, and linear kernel with tuning, radial kernel, and polynomial kernel had accuracy of 75%. After we know the performance of the SVM, we can now check what would be the proportion of returning student from what we already have in the dataset. After finding such a proportion based on already existing data, we also have 75% accuracy.

		Predicted	
		Positive	Negative
Actual	Positive	TP	FN
	Negative	FP	TN

Figure 9: Confusion matrix

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Conclusions

The experiment consisted of applying several types of Support Vector Machine models on UMM retention data. For each model, we have used different kernel either with tuning or without tuning. To better understand results, we had to go deeper into Support Vector Machine fundamentals to have an idea of how the algorithm classifies data. As a result, the best performance stays with a SVM model using a regular linear model without tuning that gives us 76% accuracy as the percentage of correct predictions for the test data. The fact that radial and polynomial kernels did not improve results, we can suggest that our UMM data in linearly separable, as those kernels are made to classify non-linearly separable data where we would expect an improvement in results. Although tuning the linear model didn't improve the results, it suggests that 76% could be the most

optimal, or very close to the optimal performance of the Support Vector Machine algorithm on our UMM retention data. A similar SVM study ~Van Belle *et al.* (2016) published by NCBI suggests similar accuracy results (73%) with their credit risk data using SVM.

Appendix

```
#Load CARET that contains the SVM
library(caret)
set.seed(1234)
#Split data 70% to 30%
train_index <- createDataPartition(y = retention$RetEvent1, p = .7, list = FALSE, times = 1)
# Use train_index of data to create train_data
train_data <- retention[train_index,]
# The rest goes to testing set
test_data <- retention[-train_index,]
# Setup for cross validation
fitControl <- trainControl(method="repeatedcv",   # 10-fold cross validation
                     repeats=5,          # do 5 repetitions of cv
                     summaryFunction=twoClassSummary,   # Use AUC to pick the best model
                     classProbs=TRUE)

#There are various options associated with SVM training; like changing kernel, gamma and C value
#Train model with the linear method with no tuning
svm_model_train <- train(RetEvent1 ~., data = train_data, method = "svmLinear", trControl = fitC
#Predict on test data using linear kernel
test_pred <- predict(svm_model_train, newdata = test_data)
test_pred
confusionMatrix(test_pred,as.factor(test_data$RetEvent1))

# Tuning linear kernel
grid <- expand.grid(C = c(0,0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2,5))
svm_Linear_Grid <- train(RetEvent1 ~., data = train_data, method = "svmLinear",
                     trControl=fitControl,
                     preProcess = c("center", "scale"),
                     tuneGrid = grid,
                     tuneLength = 10)
# Predict tuned linear kernel
test_pred_grid <- predict(svm_Linear_Grid, newdata = test_data)
test_pred_grid
confusionMatrix(test_pred_grid,as.factor(test_data$RetEvent1))

# Train and tune the SVM with Radial kernel
svm.tune <- train(RetEvent1 ~., data = train_data,
                    method = "svmRadial",   # Radial kernel
                    tuneLength = 5,                    # 5 values of the cost function
                    preProc = c("center","scale"),  # Center and scale data
                    metric="ROC",
                    trControl=fitControl)

svm.tune
```

```
restune<-as_tibble(svm.tune$results[which.min(svm.tune$results[,2]),])
restune

#Polynomial basis kernel
#Fit the model
svm4 <- train(RetEvent1 ~., data = train_data, method = "svmPoly", trControl = fitContol, prePro
# Print the best tuning parameter sigma and C that maximizes model accuracy
svm4best<-svm4$bestTune
#svm4best
#save the results for later
res4<-as_tibble(svm4best$results[which.min(svm4best$results[,2]),])
res4

#Compare the accuracy of linear default, linear tuned, radial kernel, and polynomial kernel
df<-tibble(Model=c('SVM Linear','SVM Linear w/ choice of cost','SVM Radial','SVM Poly'),Accuracy
df %>% arrange(Accuracy)
```

References

(2015). "SVM - Understanding the math : the optimal hyperplane." URL https://www.svm-tutorial.com/2015/06/svm-understanding-math-part-3/.

(2020). "Mathematics Behind SVM | Math Behind Support Vector Machine." URL https://www.analyticsvidhya.com/blog/2020/10/the-mathematics-behind-svm/.

(2020). "What is Unsupervised Learning?" URL https://www.ibm.com/cloud/learn/unsupervised-learning.

(2022). "Support-vector machine." Page Version ID: 1079167701, URL https://en.wikipedia.org/w/index.php?title=Support-vector_machine&oldid=1079167701.

Bhatt S (2019). "Reinforcement Learning 101." URL https://towardsdatascience.com/reinforcement-learning-101-e24b50e1d292.

IBM Cloud Education (2020). "What is Supervised Learning?" https://www.ibm.com/cloud/learn/supervised-learning, URL https://www.ibm.com/cloud/learn/supervised-learning.

Kohavi R, John GH (1997). "Wrappers for feature subset selection." *Artificial Intelligence*, **97**(1), 273–324. ISSN 0004-3702. doi:https://doi.org/10.1016/S0004-3702(97)00043-X. Relevance, URL https://www.sciencedirect.com/science/article/pii/S000437029700043X.

Van Belle V, Van Calster B, Van Huffel S, Suykens JAK, Lisboa P (2016). "Explaining Support Vector Machines: A Color Based Nomogram." *PLOS ONE*, **11**(10), e0164568. ISSN 1932-6203. doi:10.1371/journal.pone.0164568. URL https://dx.plos.org/10.1371/journal.pone.0164568.