Exploring CRISP-DM within the Context of Predicting Student Churn Rate

Ian McNair
Northeastern Illinois University
May, 2019

I. Abstract

Prediction is a powerful tool and essential to high performing businesses. In this study, the standard data mining process, CRISP-DM, was explored and used to predict which students would leave a local high school. The student database from this school was used, but many factors had to be left out because of the sensitive nature of the data. This data was the cleaned and preprocessed. Exploratory data analysis was performed and findings were reported to the school, which allowed administrators to get an overall view of students and factors associated with student attrition. Then three different classifiers (Naive Bayes, Artificial Neural Network, K-Nearest Neighbors) were developed, evaluated with 10-Fold Cross Validation, and used to identify 85 unique students from this school year who have the highest chance of leaving according to the models' prediction. Based on the data available, these models were sensitive to underperforming students and had trouble identifying outcomes of students who performed better in school. Even so, the AUC scores of each classifier performed well with the ANN scoring 0.9 on the training set and 0.792 on the 2019 data. The predictions of each model were then combined using a class majority voting method. 69 students left the school this year, with 30 of these being correctly identified by the voting method. The 39 missed were not identified by any of the models, which can be attributed to the type of data used. If better performance were desired, other factors like financial or scholarship information should be included.

II. Introduction

In 2012, a Target consumer and father received a coupon mailer that contained many advertisements for women going through pregnancies [1]. This mailer was not addressed to him, but to his daughter. This father contacted Target customer service in order to figure out why such advertisements would be addressed to his teenage daughter and demanded they be

stopped. Later, the father found out his daughter was indeed pregnant. How did Target know? It turns out, through tracking the buying patterns of other pregnant women, Target was able to predict which shoppers were expecting. This story sounds unsettling, but displays the power of a topic important to many industries, prediction. Businesses that can predict consumer trends can create more profit than those that cannot. A technique many of these businesses employ is called data mining. Data mining is the process of discovering interesting patterns and knowledge from large amounts of data [2]. Data mining is also commonly known as "knowledge discovery from data," which is more descriptive of what is actually done through it.

The general process of data mining involves data cleaning, integration, selection, transformation, mining, evaluation, and knowledge presentation. A formal and widely adopted methodology is known as CRISP-DM, which stands for the cross industry process for data mining. CRISP-DM goes through six different phases: Business Understanding, Data Understanding, Data Preparation, Modeling, Evaluation, and Deployment [3]. It is a cyclic process. After the completion of a phase, it may be determined that another phase needs to be updated, revised, or reworked in some way. This study uses the CRISP-DM methodology as a framework for guiding the data mining process from initial business question to final prediction.

The power of Targets prediction enabled them to send extremely directed mailers to consumers who had a need for their products. Unfortunately, making these data driven decisions can be difficult. Without the right team and digital infrastructure to mine the data, leaders of a company have to rely on intuition to guide their business. As more data is collected and stored, the need for making informed, data driven decisions will continue to increase. This knowledge discovery process is explored throughout this study within the context of student churn at a local high school.

III. Background and Data Understanding

For many schools, primary, secondary, and post-secondary alike, student churn is an important issue. The high school being studied wanted to address this issue. Some of the reasons why students leave can be inferred, like grades or finances, but the extent to which these contribute to student attrition is poorly understood. Other factors, like gender, ethnicity, or discipline could be predictive factors as well, but because data mining has not been done, the extent to which they contribute is unknown. The school in question allowed their student database to be data mined so that they could better understand their students and why they leave. Although the local high school will remain anonymous, it has a very diverse population, with students coming from very different walks of life culturally and

economically. This makes a variety of features interesting and worth studying in order to determine the most causal predictive factors.

Any real world data, if not well kept, can be incomplete, inaccurate, and inconsistent. Dealing with datasets such as these can be difficult and negatively impact the predictive model and its overall trustworthiness. If garbage data goes into a model, garbage predictions will come out. The original student database query contains over 100 features or columns of data per student. Most of the occupied space comes from the way student courses are tracked or from old features that are no longer utilized. From this large set of features, approximately twenty are kept for the initial data preprocessing phase. Most of the features seem complete, accurate, and consistent with what is expected from a school database. Their actual completeness, accuracy, and consistency will be discussed later in the exploratory data analysis phase. The features of this database are mostly nominal and give qualitative descriptions of the student, like ethnicity, gender, or program. The numerical features are grade point average, discipline, and distance from school. All of these numerical features are aggregated from other features..

IV. Data Preprocessing

Data preprocessing is an important step in CRISP-DM. It has great impact on the final outcome of any classifiers that train on it because it allows the data to be in a form that is actually consumable. It is also when the data is fixed, aggregated, or otherwise refined. If this stage is done poorly, classifiers may not even be able to train or final predictions could be very poor. The below goes through the issues faced when dealing with the student data during the preprocessing stage.

The student identification number was largely correct. Zip code was on average accurate, but had some strange values. Some zip codes were for areas very far from the school (over 1000 miles away) while others simply did not exist. These needed to be fixed. On average, most zip codes were changed to the mode value. Other zip codes that seem noticeably different from the mode where updated to zip codes that were similar to it. Using a random zip code for example, 55555, if it did not exist, would be updated to 55556 or 55554, depending on which one had a higher occurrence in the data set. Gender and elementary code were complete, accurate, and consistent. Unfortunately, the data set did not provide student names, so it was impossible to check if all the genders were connected to a corresponding gender specific name, so accuracy and consistency is assumed. Parish code and disability were similar in their reliability. Both had many missing values, but it was determined that this meant students did not attend a parish, did not have a disability, or it was not recorded. Unfortunately, the way disability has been recorded also changed at the school recently, meaning this feature had a high probability of being inaccurate and

unhelpful in the final models. Ethnicity needed to be adjusted because of recent legal changes in how it was recorded. Normally, a simple numerical code was used to determine whether a student was hispanic or not. Now, hispanic students are labeled as white with another feature altogether denoting their ethnic background. This feature was aggregated so that it could be represented by a single numeric digit instead of needing to reference two columns. The code for ethnicity ranged from 1-8, but starting at code 5, was largely unused. Codes 5-8 were combined into their own code to reduce the number of categories in this feature. Program was consistent but partially inaccurate. At this school, there should be only three main programs: Honors (H), College Prep (CP), and Learning Prep (LP), however there was another code, "G." This code was determined to be LP and was updated to be consistent across the program feature. Religion and parent marital status were largely left the same as they were complete, accurate, and consistent. There were no anomalies within these two features. Class, which determines which year in high school the student was in, had to be aggregated from two other columns. Originally, the data had a current year and a graduation year feature. These were subtracted from one another in order to obtain that student's class, which is a number from one to four. One represented a freshman, two a sophomore, and so on. Current year was left as is. Grade point average was also an aggregation of two other features, attempted points and quality points. Dividing these two numbers gave the overall GPA for the student. Discipline was not originally included in the data set and had to be merged from another database. Distance had to be predicted by using the zip code. This was done by converting the zip code to longitude and latitude values using the "uszipcode" python package [9]. Using the coordinates of the school, distances were measured using euclidean distance to calculate degrees away and converting that to miles. There is some error associated with this because a degree for longitude is much longer at the equator than at the poles. Since the location of zip codes around the school is very minimal in comparison to the size of the Earth, it is assumed each degree is 69 miles.

As previously mentioned the original data set was very wide, but was reduced by the end of the cleaning process from 20 uncleaned features to 16 cleaned or aggregated features. These features are identification number, zip code, gender, elementary school code, parish code, disability, ethnicity, school program, religion, parent marital status, class, current school year, gpa, discipline, distance and the target variable which is whether a student left the school.

V. Exploratory Data Analysis

Exploratory data analysis is used to analyze the data by creating tables, charts, and other visualizations to help address the

overall business goal of the data mining project. They also utilize descriptive statistics to analyze the data. By looking at various visualizations and metrics, it becomes more clear as to which factors may best help train the classifiers later. The below graphs look at all of the features in totality or over the course of the decade. They are explored in relation to the target variable. Some figures have been purposefully left out because they contain information that might jeopardize the anonymity of the school.

A. Student Class

On average, since 2008, the school has lost students at an average rate of 37 per year. The coefficient of determination is 0.95, which means the predicted trend matches the currently available data very well.

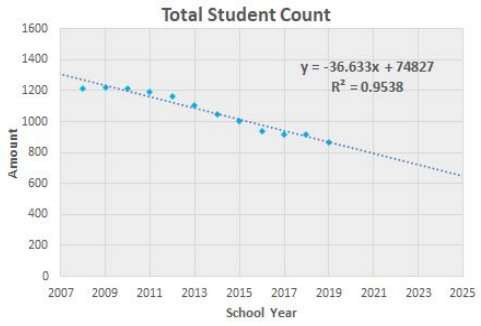

Figure 1: Average student loss per year

Based on current trends, the school will be down to approximately 600 students by 2025, which is half as much as was in 2008-2010. Other areas of interest are which classes lose the most students.

The highest turnover rates come from the freshmen and sophomore classes. The percent turnover for the freshmen and sophomore class is extremely sporadic, ranging from nearly 18% to as low as 3%, and averaging just under 10% in the last decade. However, it seems that if a student can make it to junior year, they are most likely going to remain at this school. There are some outliers, but on average this seems to be true.

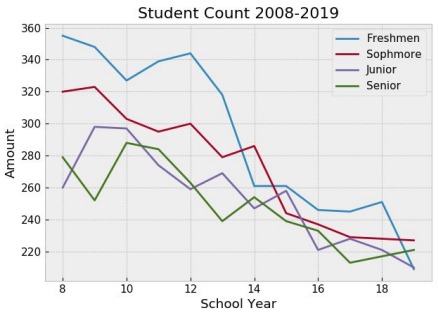

Figure 2: Trends based on student class

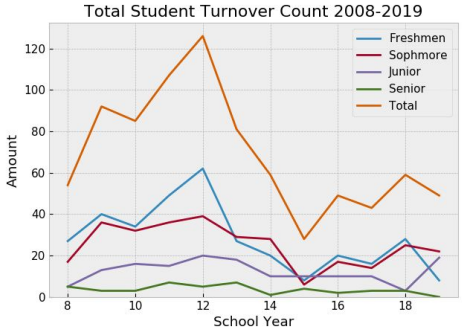

Figure 3: Students lost per year

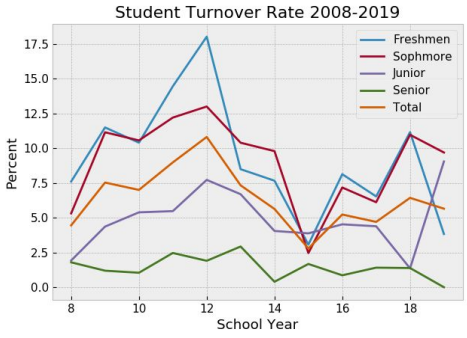

Figure 4: Percentage of students loss per year

B. Elementary School

The elementary schools figures were left out in order to maintain anonymity. There are 63 elementary schools that this school has received students from in the past that have resulted in 100% of students leaving. However, it should be noted that many of these schools provided very low numbers altogether. There are over 500 elementary schools, while this school has had over 4000 students total. The elementary feature is most likely too unique to be a candidate for a predictive factor.

C. Gender

The number of male students at this school has attributed to the most loss over the last decade while the number of female students lost per year has largely remained the same. It is alarming that the difference between the percent of male students leaving vs female students leaving is so large because the largest group has the largest percent leaving. This is a factor where more attention should be given by the school and may be a good indicator of student churn.

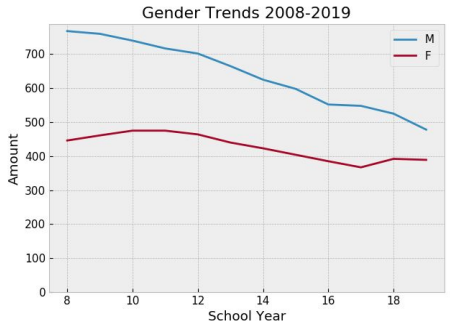

Figure 5: **Total amount of students based on gender**

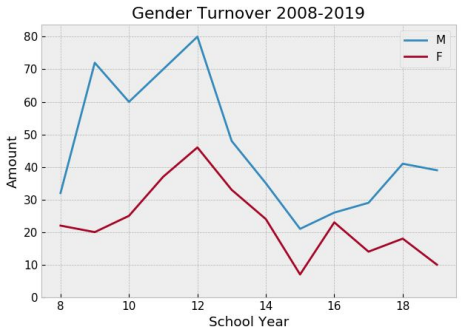

Figure 6: **Total lost each year based on gender**

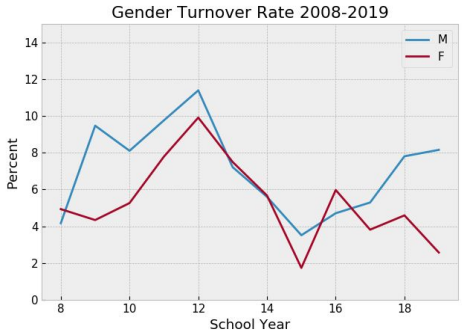

Figure 7: **Total percent lost based on gender**

D. Distance

Turnover rates increase as the distance increases which is to be expected. There is a dip near the 9 – 11+ mile mark, but it seems there is a general trend. Although the percent leaving from a far distance is large, the amount of students that actually live in the 11+ mile range is small compared to the number of students at this school. The difference between the high percentage, 23% and the low, 12% may not be a significant enough amount to aid in predicting.

Figure 8: **Percent of students that left based on distance from school**

E. Ethnicity

The ethnicity is left as a code in order to maintain anonymity. From these graphs, some of the more striking figures are seen from the ethnic code #2. Not only do they make up over a third of the student population, they also have the highest turnover rate consistently over the last ten years at 26%. Students in category #2 also leave in fairly high amounts (23%) whereas students in category 1 leave at a rate of 8%. Category 2 and 3 make up over two-thirds of the student body and over the last decade, this school has lost about 25% of them. Because of this large difference, this factor may make a good indicator of student churn.

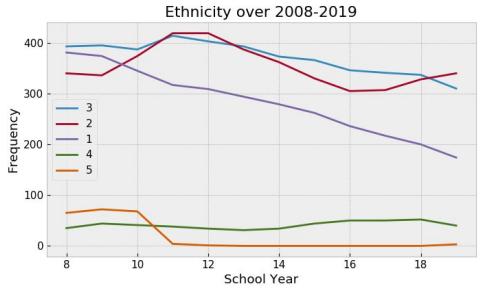

Figure 9: **Trend of students within each ethnic category**

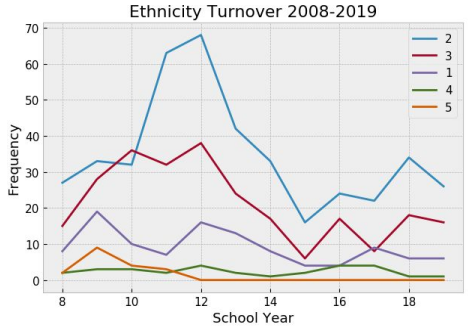

Figure 10: **Trend of student churn within each ethnic category**

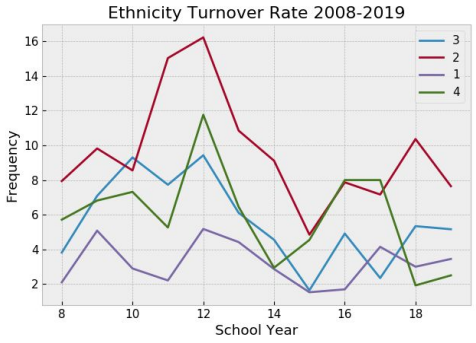

Figure 11: **Trend of student churn percentage within each ethnic category**

F. Program

Program is a bit difficult to analyze. Students just entering this school without taking the entrance exam are usually added into the college preparatory program. Although there is a large difference between the rates for Honors, College Prep, and Learning Prep, most students that leave are in the CP program. Program itself is supposed to be an indicator of student ability, but many students are potentially grouped incorrectly. LP turnover rates are much higher. It is also know that students in the LP program are moved up to the CP program at the beginning of their junior year, which hides even more information about these students. It seems that program

may not be a good predictive indicator because of the amount of information that is hidden by how the school handles placing students.

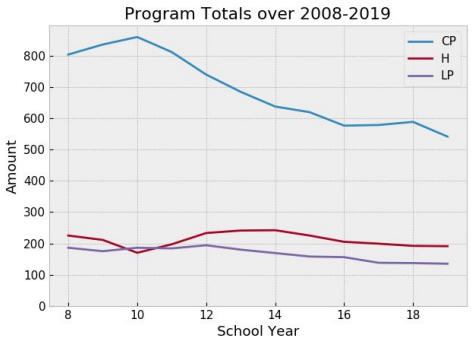

Figure 12: Trend of total students within each educational program

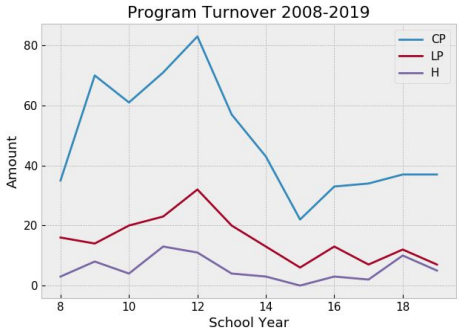

Figure 13: Trend of student churn within each educational program

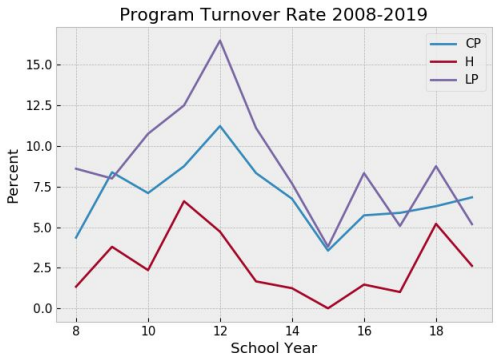

Figure 14: Trend of student churn percentage within each educational program

G. Grade Point Average

Students not performing well leave at a much higher rate than others. Students with an F overall leave at a rate close to 80% whereas those with an A+ leave at less than 5%. This makes student grades a very good predictor for differentiating between students who stay and students who leave.

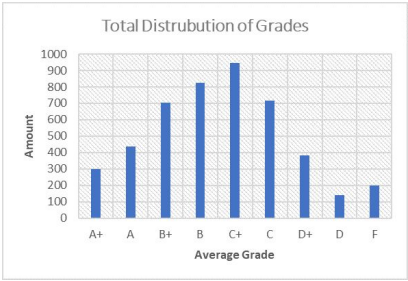

Figure 15: Distribution of all student grades

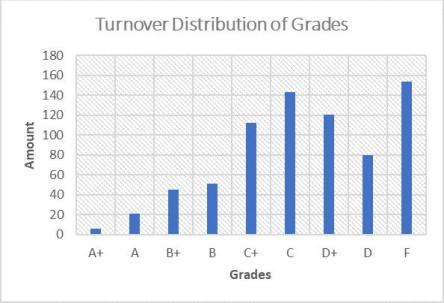

Figure 16: Distribution of grades for students who leave

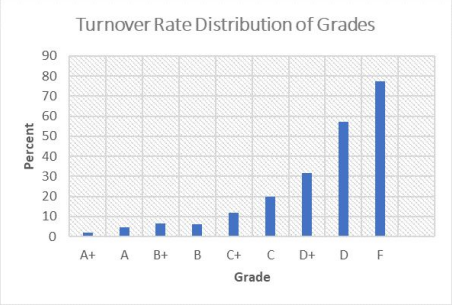

Figure 17: Distribution describing the grades of students who leave

H. Discipline

Students with greater than 25 detentions leave nearly 65 % of the time on average. Those without detentions leave 10% of the time. Unfortunately, as mentioned before, the variance for this category is extremely high. There are many students with very high discipline rates who do not leave while there are others without any who do leave. Without more information this feature may not be as predictive as it seems.

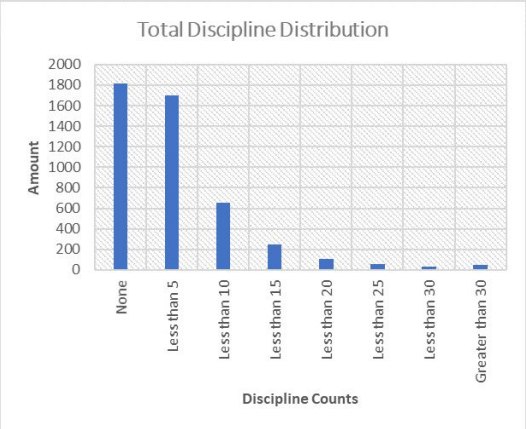

Figure 18: **Discipline distribution of all students**

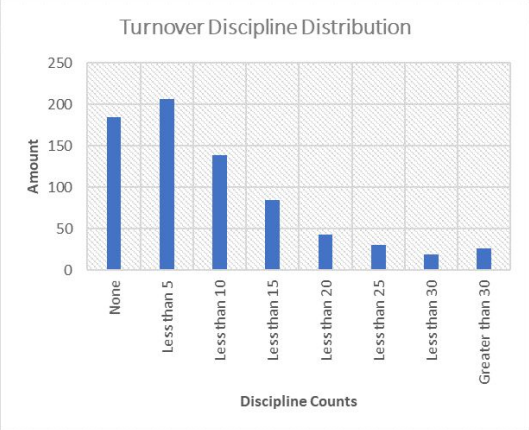

Figure 19: **Discipline distribution of students who leave**

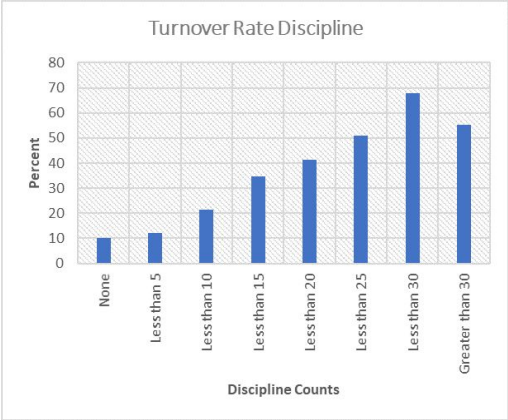

Figure 20: **Discipline distribution of students who leave**

I. Parental Marital Status

Parental Marital Status is a feature which describes the current relationship of a student's parents. Most students are of the 1st category, married, which has the second lowest turnover rate. The highest categories are 3 and 6, which have rates of 22% and 25% respectively. Although students mostly from the 1st category leave, their rate of leaving is much lower, at 12%. To some degree, doubts are raised about the validity of this number because divorce rates in the USA are 40-50% [4]. This school either breaks this mold greatly or the data taken is inaccurate. This means parental marital status may not be a good predictor because it is inaccurate.

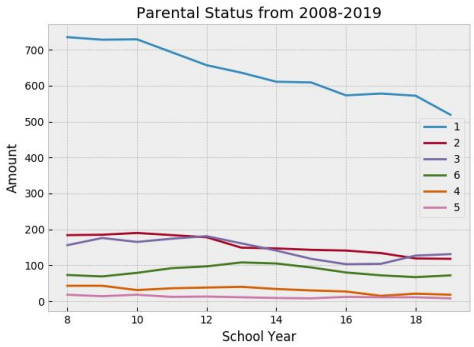

Figure 21: **Trend of parental marital status for all students**

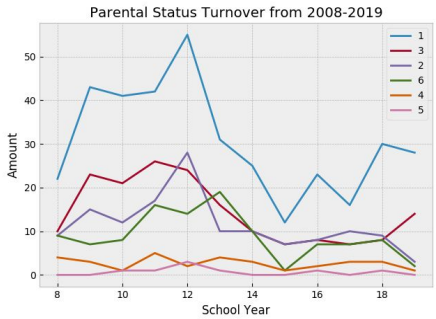

Figure 22: Trend of parental marital status for all students that leave

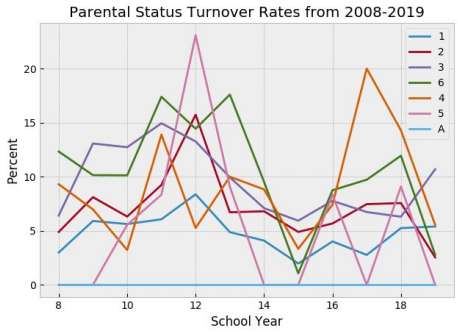

Figure 23: Trend of the percent of parental marital status for students that leave

J. Exploratory Data Summary

Not all of the features in the student database are useful. How disability was recorded and handled changed recently, and is no longer kept in this specific database. This makes the disability feature very incomplete and inconsistent across students. Other nominal features are unhelpful because they have a very high number of categories. For example, there have been roughly 4000 students unique students at this school over the last decade, and they come from 500 different elementary schools. A majority of these students come from ten schools while 60 elementary schools have only ever provided a single student over the last ten years. The Parish feature is also very similar. In both these cases, they make bad predictors because of the extremely high number of unique categories which affects the dimensionality of the train/test sets greatly. During the modelling phase, different variations of features will be trained on by the classifier in order to see what gives the best fit.

VI. Data Preparation for Modeling

A. Binning Continuous Data

The three classifiers used in this study are created by using only categorical or nominal features. Other than the changes already mentioned, the nominal features were left as is. Numerical features were discretized in order to make them categorical. In general, the discretizing of quantitative data is usually done because the data is noisy. Although some information may be lost in the discretizing process, it was found that discretizing the data provided better overall prediction in the final model outcomes compared to a combination of using both.

The three numerical features are grade point average (GPA), discipline count, and distance. A students grade was discretized based on common grading themes. A GPA of 4.5 and above became "A+", 4.0 to 4.5 became "A" and so on. Altogether there were 10 different categories for grade.

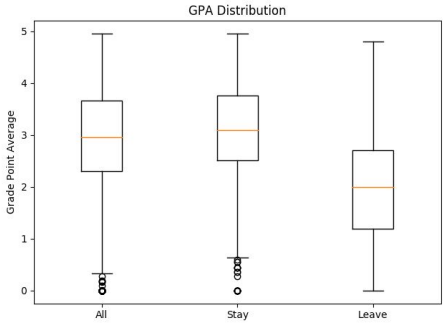

Figure 24: **Box Plot of student GPA**

Both discipline and distance were especially noisy and have many outliers. In general, a student with more detentions had a higher probability of leaving, but there were quite a few students with little to no detentions that left as well as those with 30+ detentions who did not leave.

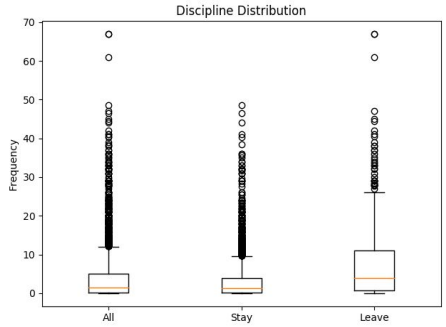

Figure 25: **Box plot of student discipline**

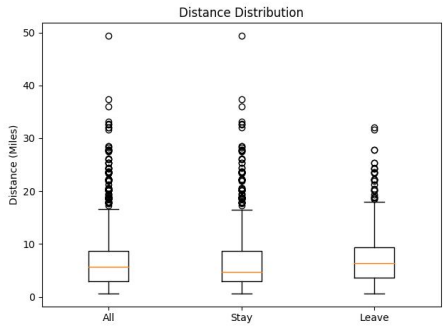

Figure 26: **Box plot of student distance from school**

Distance is based on a nominal feature, student zip code, which has no direct relationship with distance. It is nearly as noisy as discipline which makes both of these a good candidate for binning. Although GPA could be left as continuous, it is also binned in order to make the classifiers easier to implement.

B. Preparing Nominal Data for Quantitative Classifiers

Most of the data for this study is nominal. Because of this, One Hot Encoding was used to quantify the categorical data. One Hot Encoding is the process of taking every category from a feature and creating a new feature of that category which only has values of one or zero [6]. For example, suppose the data from table 1 needed to be encoded.

Table 1: **Flower example dataset**

Flower Set	
Color	Is Rose
Red	Yes
Blue	No
Red	Yes
White	Yes
Blue	No
Red	No

The color is the the only feature and the “is Rose” is the target variable. Each unique category of color will be made into is own feature. Then the original color feature is dropped.

Table 2: One Hot Encoded set

Flower Set			
Red	Blue	White	Is Rose
1	0	0	Yes
0	1	0	No
1	0	0	Yes
0	0	1	Yes
0	1	0	No
1	0	0	No

This process leads to something known as the dummy variable trap. If there are three unique categories for color, only two of these need to be encoded in order to provide the full amount of information for the classifier. It is important to make sure to leave out the dummy variable in regression analysis, however classification is not impacted by the extra feature. After dropping the dummy variable in the example, the final dataset looks like table 3.

Table 3: Dropping dummy feature

Flower Set		
Red	Blue	Is Rose
1	0	Yes
0	1	No
1	0	Yes
0	0	Yes
0	1	No
1	0	No

The tuple with 0 for both colors is considered to be the white color. The dummy variable can be any of the original categories.

Unfortunately, this makes the dimensionality very high for categorical datasets. If the elementary school code were used as a feature in the final model for example, its cardinality would be bigger than the original, uncleaned data. There is no way around this because the machine learning package used to implement most of the classifiers only accepts data with numerical values. Fortunately, the final cardinality of the dataset after utilizing this process is 14.

VII. Modelling

Three different classifiers were implemented to explore different frameworks utilized for predictive analytics. The Naive Bayes Classifier (NB) was built from the ground up to handle nominal variables. SciKit-Learn [5], a powerful package built for python and doing machine learning, was used to implement another type of Naive Bayes so that a comparison could be made. An Artificial Neural Network (ANN) was constructed using Keras and Tensorflow, both packages developed by Google. Finally the K-Nearest Neighbors Classifier (KNN) was built purely using SciKit-Learn.

The original question is a binary classification problem which groups student into either a leave or stay target class. Each of the models output probabilities which described which target category each tuple was predicted to belong to. In general, the threshold for each is 0.5, with 0.5 and above being classified as student who leaves and under 0.5 as students who will stay. The range of output

values is from 0 to 1. These thresholds can be adjusted, but this can potentially cause overfitting and needs to be done with care.

A. Naive Bayes Classifier

The Naive Bayes Classifier is a bayesian classifier, based on Bayes Theorem. These types of classifiers predict the probability a given tuple belongs to a certain class [8]. The Naive Bayes classifier is considered naive because it assumes all predictive features are independent. For example, consider a model that is trying to classify flowers as either a rose or not. Some features of the flowers being tested may be color, plant height, or number of leaves. If the plant height changes, this may affect the number of leaves, but probably will not affect the color of the plant. In this case, number of leaves may be dependent on plant height, but color is independent of plant height. In order to implement the Naive Bayes Classifier, Bayes Theorem must be understood:

$$P(H/X) = \frac{P(X/H)P(H)}{P(X)}$$

P(X) is the probability of some feature occuring while P(H) is the probability of the target variable occurring. These are known as the prior probabilities. Using the previous example, P(X) would give the probability of a plant that is blue, 10 cm tall, or has 4 leaves regardless of whether or not it is a rose. P(H) is the probability that any given flower is a rose regardless of what is known about any given plant. For a constant data set, this value should never change. P(X/H) is the posterior probability of the tuple conditioned on the hypothesis. Using the example, this relates knowing the plant is a rose to its actual attributes. Finally, P(H/X), gives the probability of some tuple being the target value depending on the other probabilities, which in the example case is whether or not it is a rose. Empirical studies have found that the Naive Bayes Classifier is very comparable to Neural Networks and Decision trees in some domains [8]. Although they have had success in prediction, the output probabilities are usually not completely trustworthy because of the independence assumption.

B. Artificial Neural Network

Artificial neural networks (ANN) are software implementations based loosely on biological neurons in the brain. As an organism begins to learn a new skill, certain neural pathways are strengthen over others based on how well that pathway allows for the correct output. For example, suppose a student is learning basic addition. As they learn how to add numbers together, certain patterns emerge for the student and are strengthened by a teacher acknowledging the correct summation. For incorrect summations, patterns that a student thought were emerging are minimized such that the student relies on them less and less as summation training goes on. ANNs work similarly. Given input and target information, ANNs will attempt to learn patterns that output the correct

target values. Learning is a bit of a misnomer however. The input values are combined with weights and biases which can be adjusted as training of the ANN occurs. The weights and biases which output the best target values are reinforced while those that output the worse target values are diminished.

A simple example of the ANN can be understood by figuring out the equation to the conversion of temperature from Celcius to Farenheit. In order to train the ANN to learn the relationship between these two measurements, a training set must be provided which gives known input values and target values. Suppose the table 4 was used for training the ANN:

Table 4: Temperature dataset

Celcius	Fahrenheit
0	32
5	41
-5	23
-40	-40
100	212

After training, the final model would be able to take other temperatures in Celsius and predict their value in Fahrenheit. A simple neural network with only one node and bias inside the hidden layer takes the form:

$$Input_{weighted} = x_1 w_1 + b$$

During the training stage, the neural net would attempt to adjust the weight and bias in order to get as close to the correct target value as possible. In other words, the neural net tries to recreate the temperature conversion equation:

$$T_F = 1.8T_C + 32$$

A model was trained for the purpose of this example. After training, the weight reinforced was 1.82 and the bias was 29.05, which is very close to the actual equation for temperature conversion. Unlike other models which can be tuned by adjusting parameters, the ANN tweaks itself during training. This makes the ANN applicable in many areas, but these models can take an extremely long time to train and validate. For applications where speed is a factor, ANNs are not the best candidate.

C. K- Nearest Neighbors Classifier

The K-Nearest Classifier (KNN) operates on a very simple premise. Given some tuple with unknown class, a KNN classifies that tuple based on the data that appears most often around it. For example, suppose a customer has a salary of $35,000 and is 20 years old. The target for the classification is whether or not the customer buys a computer. The KNN classifier will look at all of the available training data and classify the customer based on the training data. A new customer gets classified depending on how far away the other data points are, with a weight corresponding to that distance. Euclidean is

the typical distance used, but there are other ways to measure distance as well. These two factors are the main parameters in adjusting the performance of the K-Nearest Neighbor classifier. Figure 27 is a visualization of the KNN trained on salary and age.

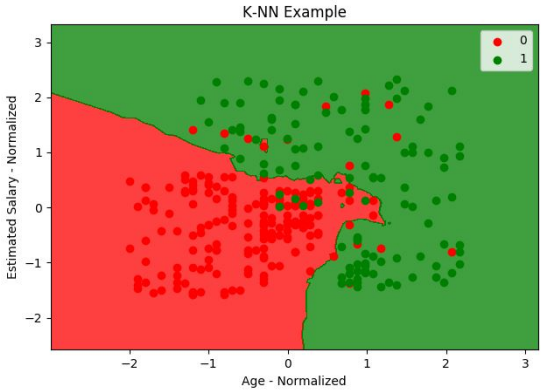

Figure 27: Utilizes customer data to predict the purchase of a computer

The green dots represent customers who actually bought the computer while the red dots represent those who did not. The colored areas correspond to what the training data will predict a new customer as based on salary and age. Dots in their same colored areas are correct predictions. A red dot in a green area represents a customer who was classified as buy but did not whereas a green dot in the red area represents the opposite. In this specific example, the background color was created using the 5 nearest neighbors and weighting their influence based on euclidean distance.

D. Classifier Implementation

In Python, the SciKit-Learn package contains implementations of the Naive Bayes and K-Nearest Neighbor Classifiers. The issue with these implementations is that they are purely quantitative. The data in the study is mostly nominal. A categorical Naive Bayes was developed in order to handle String or nominal inputs. This implementation follows the traditional methodology, but does not need to use dummy variables and can handle nominal data. Implementation is explained using the previous example of identifying a flower:

Table 5: Example flower set for NB

Flower Set		
Color	Leaves	Is Rose
Red	3	Yes
Blue	5	No
Red	4	Yes
White	4	Yes
Blue	4	No
Red	8	No

P(X) is the probability of the category of some feature. For now, consider the probability a flower in the set is red. P(Red) = 3/6 = 0.5 as seen in table 6.

Table 6: Probability of red flower in set

Flower Set		
Color	Leaves	Is Rose
Red	3	Yes
Blue	5	No
Red	4	Yes
White	4	Yes
Blue	4	No
Red	8	No

P(H) is the probability any given flower is a rose. There are 3 total roses in the set, so P(Rose) = 0.5

Table 7: Probability of rose in the set

Flower Set		
Color	Leaves	Is Rose
Red	3	Yes
Blue	5	No
Red	4	Yes
White	4	Yes
Blue	4	No
Red	8	No

P(X/H) is the probability that a red colored flower is a rose. P(Is red if flower is rose) = 2/3 = 0.67.

Table 8: Probability a rose is a red flower

Flower Set		
Color	Leaves	Is Rose
Red	3	Yes
Blue	5	No
Red	4	Yes
White	4	Yes
Blue	4	No
Red	8	No

Finally, using these values and Bayes Theorem:

$$P(H|X) = \frac{0.67 * 0.5}{0.5} = 0.67$$

This is then repeated for each unique category. The same process is repeated and stored for each category. P(Is rose if flower is blue) is 0, since there are no blue flowers that are roses in this set, and P(is rose if flower is white) is 0.33, because the total sum of the probabilities need to add up to 1.

Although all data in the study was discretized, there is another way to calculate probability for continuous variables. Technically, the feature "Number of Leaves", could also be treated as discrete for this example since a fraction of a leaf is impossible. A better way to calculate the probability of this feature however, is by assuming the data is continuous and normally distributed. The Gaussian is used to determine the probability density and takes the form:

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Where μ is the mean of the data, σ is the standard deviation, and x is the quantitative data value, which in this example is Number of Leaves. Similarly, the probability densities need to be calculated separately based on the target category. For the yes category, the mean of the data is 3.67, the standard deviation is 0.471, and x is the actual number of leaves. The probability density functions are plotted in figure 28.

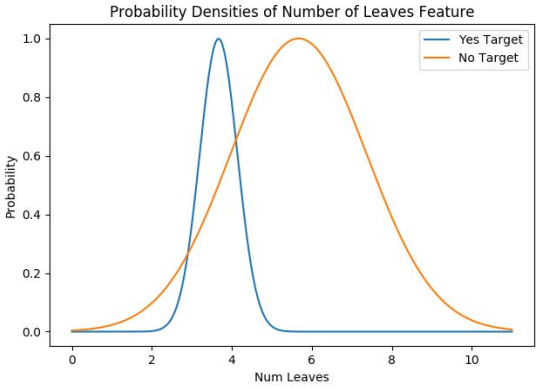

Figure 28: Probability densities of the yes and no targets for the rose example

This graph shows the probabilities corresponding to the target values based on the number of leaves a flower has. The posterior probability, P(X|H) which is P(x

number of leaves if flower is rose), is approximately 3.67 for flowers which are roses. Then, multiplying by the prior probability of the hypothesis and dividing by the prior probability of the feature, gives the posterior probability, P(H|X). Suppose a flower has 4 leaves, P(H|X) = 0.778 * 0.5 / 0.5 = 0.778.

To finally classify a tuple as belonging to the yes or no class, the probabilities are multiplied because of the class independence assumption. The product which gives the maximum between the Yes and No target categories determines which class the tuple will be classified as. In the actual implementation, the Gaussian method was not utilized in the final implementation because of the amount of noise present in the data. Using the Gaussian for the continuous variables decreased the overall prediction rate in the final model. They were discretized because of this. The developed Categorical Naive Bayes can be found in section B of the appendices.

The SciKit-Learn implementation of the Naive Bayes Classifier used to compare to the developed categorical one is very easy to implement and takes three lines of code, as seen below:

```
# Fitting classifier to the Training set
1    classifier = BernoulliNB()
2    classifier.fit(X_train, y_train.iloc[:,0])
# Predicting the Test set results
3    y_pred = classifier.predict(X_test)
```

As long as the data is already preprocessed, the actual implementation of the any classifier using SciKit-Learn fast and efficient. There are three styles of creating a Naive Bayes Classifier which are applicable to different types of data. For this study, the Bernoulli Naive Bayes was implemented. The Bernoulli Naive Bayes classifier takes features which only have values of one and zero. This implementation became the basis for the final NB model because it worked more seamlessly with the other functions of SciKit-Learn. Both the developed Categorical Naive Bayes and the Bernoulli Naive Bayes performed nearly identically as seen in table 9 and 10. The difference can be attributed to a threshold parameter added to the Categorical that lets the user tweak at which probability the model begins classifying a tuple as.

Table 9: Confusion matrix of the developed Categorical Naive Bayes Classifier.

Categorical		Actual	
		Leave	Stay
Predicted	Leave	110	54
	Stay	100	842

Table 10: Confusion matrix of the Bernoulli Naive Bayes Classifier implemented with SciKit-Learn

Bernoulli		Actual	
		Leave	Stay
Predicted	Leave	112	52
	Stay	106	836

The artificial neural network was implemented using Keras, which is built on top of Tensorflow. Although it is entirely possible to create an artificial neural network using only Tensorflow, Keras simplifies many of the processes making the creation of an ANN very simple. The below code section demonstrates the implementation of the ANN for the study:

```
# Setting up the Classifier
1       classifier = Sequential()
# Making the ANN
2       classifier.add(Dense(units = 15,
                             kernel_initializer = 'uniform',
                             activation = 'relu',
                             input_dim = len(X.columns)))
3       classifier.add(Dropout(rate = 0.1))
4       classifier.add(Dense(units = 10,
                             kernel_initializer = 'uniform',
                             activation = 'sigmoid'))
5       classifier.add(Dropout(rate = 0.1))
6       classifier.add(Dense(units = 1,
                             kernel_initializer = 'uniform',
                             activation = 'sigmoid'))
# Compiling ANN
7       classifier.compile(optimizer = 'adam',
                                 loss = 'binary_crossentropy',
                                 metrics = ['mae'] )
# Fitting the data to the classifier
8       classifier.fit(X_train,
                             y_train.iloc[:,0],
                             batch_size = 32,
                             epochs = 100)
# Predicting
9       y_pred = classifier.predict(X_test)
```

The Sequential class contains the methods needed to create a fully connected neural network. Line 2 -]]to Line 6 add the input, hidden, and output layers. Line 2, the input layer, takes in the initial features. The input_dim needs to match the cardinality of the features. In line 3 and 5, a dropout is added to prevent overfitting. Various activation functions are also used. Trial and error was used to determine which type of activation function worked best in creating the neural network. The final layer at line 6 has only one node or unit, because of the binary nature of the classification. Binary cross entropy needs to be used because of the same reason. The optimizer and metric were found using grid search (discussed in the model evaluation section) and gave the overall prediction rate. For batch size and epochs in line 8, grid search was also used to determine the best parameters. Finally, in line 9, similar to how SciKit-Learn operates, values are predicted using a predict method.

The K-Nearest Neighbors Classifier is very similar in implementation to the Naive Bayes Classifier. Although it has more parameter, like the number of neighbors and distance, the implementation takes the same amount of work.

```
# Fitting classifier to training set
1       classifier = KNeighborsClassifier(n_neighbors = 61,
                                          weights = 'distance',
                                          metric = 'minkowski',
                                          p = 1)
2       classifier.fit(X_train, y_train.iloc[:,0])
# Predicting the test set results
3       y_pred = classifier.predict(X_test)
```

In the implementation above, the n_neighbors parameter governs how many other tuples are used to classify a new tuple, the weights parameter causes data neighbors to affect the classification of the new tuple less as the distance between them increases, and the metric and p parameters make the distance calculation of the classifier based on Euclidean distance. Each of these parameters was tuned by utilizing grid search, which found

that this combination gave the best overall results.

IIX. Model Evaluation Techniques

In practice, a good classifier predicts future outcomes very well. Many times, these future outcomes cannot be determined, so other methods need to be used to evaluate the performance of a classifier. These methods need to be based on the available data with the assumption being that the data used is generalized enough to be reflective of the future data. For this study, K-Fold Cross Validation and Receiver Operating Characteristic curves were used to validate the classifiers performance. The area underneath the ROC curve (AUC) is used as a metric to measure how well the model discriminates data.

One issue in training any classifier is overfitting. When a classifier overfits data, it becomes great at predicting the training and test data, but is not good at predicting with new data. Having a model that cannot predict on new data is useless in industry because it provides no insights outside of a very narrow scope of data. When splitting the the training data into train and test sets, that certain distribution of data can give very good results. If the same data were split another way, the model may actually give very poor results. Ideally, no matter how the data is split, the accuracy of prediction should be approximately the same. K-Fold Cross Validation is a technique used to measure how well a model generalizes and predicts by utilizing different train/test splits on the same data. The "K" in K-Fold Cross Validation is the number of different test sets used to validate a models accuracy. To use this technique, the training data is divided into a train/test split. The model is trained and some metric is used to measure how well it does, like accuracy or recall. Then these steps are repeated, however the test data is changed to another section or fold within the original training set. The left over data is used as the training data, the model is retrained, and the same metric is recorded. The below image gives a description of this process.

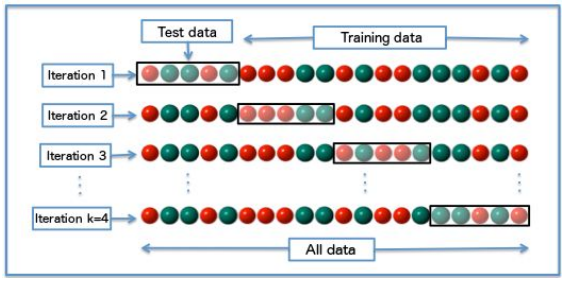

Figure 29: Example of K-Fold Cross Validation, created by and used with permission from Fabian Flöck [7] licensed by CC BY-SA 3.0

Ideally, across different folds, the metric used to gauge model performance will have a low standard deviation indicating the model will perform just as well on new data.

Tuning models can be very tedious and can oftentimes lead to overfitting, however it is essential to creating a classifier that fits the data well. SciKit-Learn has a method called grid search which allows the user to input different values for different parameters within a classifier and runs each

combination, validating the iteration with K-Fold Cross Validation. The output of this method are the parameters that give the best score while also not overfitting the data. Grid search was utilized for both the ANN and KNN to find best parameters. The NB classifier does not have other parameters, so grid search was not needed.

For this study, the Receiver Operating Characteristic (ROC) was used as an evaluation tool for understanding how well the model predicted the target variable. The Receiver Operating Characteristic is a graph of the true positive rate versus the false positive rate and "shows the tradeoff between sensitivity and specificity." [8] Figure 30 displays a sample ROC.

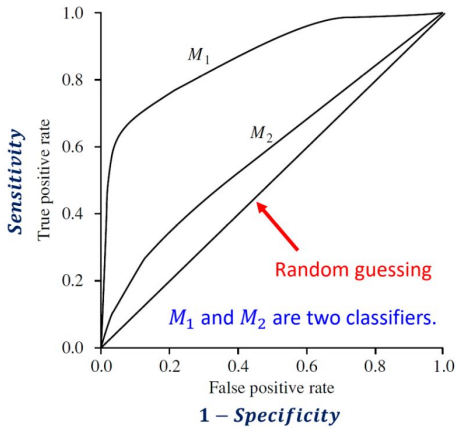

Figure 30: Example of two classifiers and their resulting ROC curves

M1 and M2 represent two trained classifiers. The line from 0,0 to 1,1 represents a classifier that is as good as randomly guessing. Models further from this line are better able to discriminate data. The graph in figure 31 displays the ideal classifier, which is one that predicts the target variable perfectly.

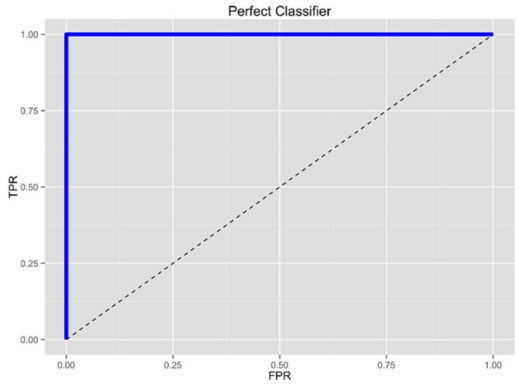

Figure 31: The perfect classifier with an AUC of 1.0

Along with the ROC curve, the area underneath it (AUC) is metric that measures classifier discrimination. Generally speaking a model with AUC 0.9 and higher is a great model, 0.8-0.9 is good, and so on [9]. AUC scores close to 0.5 are models that are no better than guessing and should be an indicator that the model needs to be fixed or the data is not fit for training a model on. An AUC close to zero indicates a model very good at making the opposite guess.

IX. Outcomes

A. Model Validation

The results of applying these evaluation methods are displayed in the following tables and figures. A train-test split of 75-25 was used for training and evaluating every model. 10-Fold Cross Validation was used to make sure the classifiers were not overfit. Instead of accuracy, recall was used in order to evaluate the model performance because having a false positive is less detrimental

than a false negative within the context of this study. A high recall is one with a minimized number of false negatives. The following tables show the results of 10-Fold Cross Validation for each model.

Table 11: Result of 10-Fold Cross Validation with NB Classifier

NB Classifier	Mean	0.646
	Standard Deviation	0.045
Model Recall across 10 Folds	0.618	0.709
	0.611	0.611
	0.611	0.593
	0.704	0.685
	0.704	0.618

Table 12: Result of 10-Fold Cross Validation with ANN

ANN Classifier	Mean	0.424
	Standard Deviation	0.0879
Model Recall across 10 Folds	0.333	0.426
	0.281	0.333
	0.412	0.404
	0.523	0.547
	0.545	0.431

Tabe 13: Result of 10-Fold Cross Validation with KNN Classifier

KNN Classifier	Mean	0.409
	Standard Deviation	0.065
Model Recall across 10 Folds	0.491	0.455
	0.418	0.444
	0.333	0.278
	0.444	0.37
	0.481	0.37

The mean in the above tables are the results of the cross validation and represents the average recall for each model. In general, the higher, the better, with 1.0 being a model that perfectly identifies all students leaving. However, this is also achievable if the model predicts that all students leave. It is important that there is an acceptable balance between false negatives and false positives. Both extremes are unhelpful to the end user of the predictions. The standard deviation of each model is also very low. Overall, the KNN model seems to be the most prone to potentially overfitting while the NB, having the lowest standard deviation, is a model that seems to be the least affected by overfitting.

B. Results

After training and testing each classifier, each model is then fed the newest 2019 data that contains the most updated features. It is expected that the AUC will score less because the school year has not ended. At the beginning of the year, the AUC of this graph would be near zero simply because students have not had a chance to decide to leave. The AUC for the 2019 data however is starting to trend near the 2008-2018 AUC after half the school year had ended. The green curve represents the 2008-2019 prediction rates and the blue curve represents the 2019 prediction rates. The confusion matrix, ROC curves and AUC score is provided in the following tables and figures.

Naive Bayes Classifier

Table 14: NB Confusion Matrix for 2019 student data

2019 NB Predictions		Actual	
		Leave	Stay
Predicted	Leave	24	45
	Stay	39	759

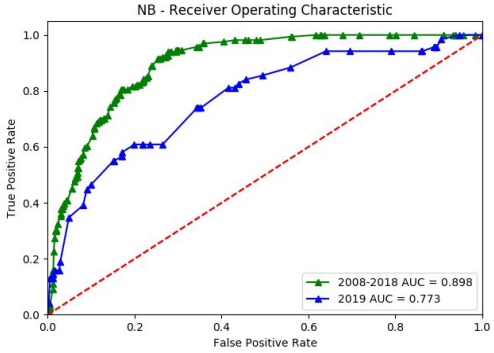

Figure 32: ROC and AUC of NB Classifier

Artificial Neural Network

Table 12: ANN Confusion Matrix for 2019 student data

2019 ANN Predictions		Actual	
		Leave	Stay
Predicted	Leave	29	40
	Stay	49	749

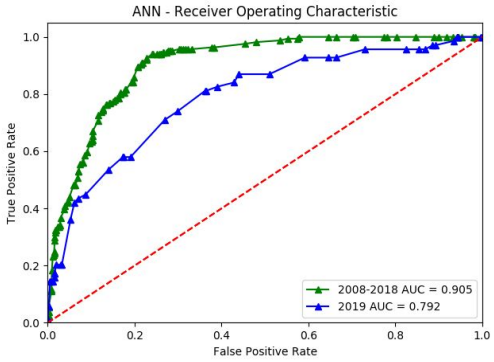

Figure 33: ROC and AUC of ANN

K-Nearest Neighbors Classifier

Table 13: KNN Confusion Matrix for 2019 student data

2019 KNN Predictions		Actual	
		Leave	Stay
Predicted	Leave	14	55
	Stay	20	778

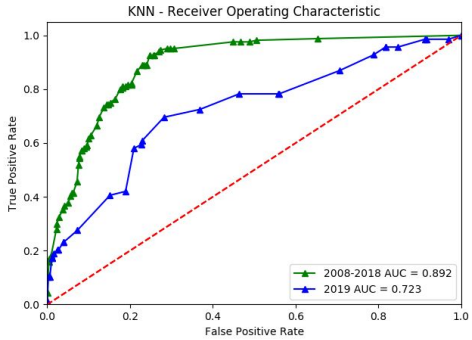

Figure 34: ROC and AUC of KNN Classifier

The ANN had the best scores overall. Although it tends to over predict, it correctly identifies the most students from the 2019 data who left. The KNN model predicts less well overall on the 2019 data. This most likely stems from the class imbalance found in the data set as KNN models are sensitive to this. Class imbalance means that the number of students who left this school is significantly less than those who stayed. The NB algorithm has very good prediction rates for how quick it is. 10-Fold Cross Validation on the ANN takes 5-10 minutes whereas the same task for the NB takes less than a minute. None of the models are perfect, so another technique is utilized to improve the overall prediction rates. This method is known as Class Majority Voting.

This method takes all of the students predicted by any classifier and combines them into a final table. Any students that were identified by all three models are deemed a high priority student, those with 2 votes are average priority, and those with one vote are classified as low priority. In total 85 unique students were predicted to leave by at least one model. 24 were predicted by at least two or more models. 12 students were classified as leaving by all three models. In 2019 so far, this school has lost 69 students. Of those, 30 students were predicted by the models correctly while 39 were not identified by any model. The final prediction table is provided in section A in the appendices.

If garbage data goes into training a model, garbage predictions come out. There could have been significant improvements if more data could have been accessed. The models that were developed were sensitive to students who performed poorly in school. Students however leave schools for many other reasons. Students in the past have used this school as a transition year to get into another, better performing school. Other students lose scholarships or cannot compete athletically because of grades, which causes them to transfer out. Other students struggle to find community, which can lead to a decrease in satisfaction and a student transferring or leaving this school. Important financial information, student after school involvement, and a students' family relationship to this school (e.g. do siblings attend, are parents alumni) are all factors which could greatly help prediction and exist, but could not be accessed for this study.

X. Conclusion

In this study, CRISP-DM was explored in order to predict student churn and identify which students were potentially leaving this year. This process went from taking raw data, preprocessing it, exploring and visualizing it, training models on it, and finally making real predictions that can impact that school. Although validated by 10-Fold Cross Validation, ROC curves and scored with the AUC metric, the models still missed 39 of the 69 students that left this year. This could potentially be fixed if other, more sensitive data were used. The features that seemed to discriminate the most were grade point average, class, academic program, and gender. Although not perfect, these models provide a way for this school to continually predict on student churn, but also can help the leaders of this school identify ways to address potential reasons for student attrition. Schools live and die by their student body. In any consumer focused business, predicting customer wants and needs enables larger profits and growth. Although looking at a school in this fashion may seem ingenuine, it is still very true. A school that knows and can provide for student needs will be able to not only retain them, but grow and better equip them for their future.

XI. References

1. Duhigg, C. (2012, February 16). How Companies Learn Your Secrets. Retrieved April 30, 2019, from https://www.nytimes.com/2012/02/19/magazine/shopping-habits.html?pagewanted=6&_r=1&hp&mtrref=undefined
2. Han, J. (2012). Data mining: Concepts and techniques, third edition (3rd ed.). Waltham, Mass.: Morgan Kaufmann Publishers.
3. What is the CRISP-DM methodology? (n.d.). Retrieved February 2, 2019, from https://www.sv-europe.com/crisp-dm-methodology/
4. Kazdin, A. E. (2000). Encyclopedia of psychology. Oxford: Univ. Press.
5. Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.
6. Harris, David and Harris, Sarah. Digital design and computer architecture (2nd ed.). San Francisco, Calif.: Morgan Kaufmann. p. 129. ISBN 978-0-12-394424-5.
7. Fabian Flöck [CC BY-SA 3.0 (https://creativecommons.org/licenses/by-sa/3.0)]
8. Tape, T. G. (n.d.). Interpreting Diagnostic Tests. Retrieved April 1, 2019, from http://gim.unmc.edu/dxtests/Default.htm
9. Uszipcode-project, MacHu-GWU, Sanhe
10. John D. Hunter. Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9, 90-95 (2007)
11. Wes McKinney. Data Structures for Statistical Computing in Python, Proceedings of the 9th Python in Science Conference, 51-56 (2010)

XII. Appendices

A. Majority Voting Outcomes

Student ID	Model Predictions			Voting Sum	Actual Outcome
	ANN	NB	KNN		
1	1	1		2	0
2	1	1		2	0
3	1	1		2	0
4	1	1		2	0
5	1	1		2	0
6	1	1		2	0
7			1	1	0
8	1	1		2	0
9	1	1		2	0
10	1	1		2	0
11	1	1		2	0
12	1	1		2	1
13	1	1		2	0
14	1	1		2	1
15	1	1		2	0
16	1	1	1	3	1
17	1	1	1	3	1
18			1	1	0
19			1	1	0
20	1			1	0
21	1			1	1
22	1	1	1	3	1
23			1	1	1
24			1	1	0
25	1			1	1
26	1	1		2	1

27	1	1		2	1
28	1	1		2	1
29	1	1		2	0
30	1	1		2	1
31	1	1	1	3	0
32	1	1		2	0
33	1	1		2	0
34	1	1		2	1
35			1	1	0
36	1	1		2	0
37	1	1		2	1
38	1	1		2	0
39	1	1	1	3	0
40	1	1		2	1
41	1	1		2	1
42	1	1		2	0
43	1			1	0
44			1	1	0
45	1			1	0
46	1			1	0
47			1	1	0
48	1			1	0
49	1			1	0
50	1			1	1
51			1	1	0
52	1	1	1	3	1
53	1	1	1	3	1
54	1	1		2	0
55	1	1	1	3	0
56	1	1		2	0
57	1	1	1	3	1

58	1	1		2	0
59	1	1	1	3	0
60	1	1	1	3	0
61	1	1	1	3	1
62	1	1	1	3	1
63	1	1		2	0
64	1	1	1	3	1
65	1	1		2	1
66	1			1	1
67	1	1		2	0
68	1	1		2	0
69	1	1		2	0
70	1	1	1	3	1
71	1	1	1	3	1
72	1	1	1	3	1
73	1	1		2	0
74	1	1	1	3	0
75	1	1	1	3	0
76	1	1		2	0
77	1	1	1	3	0
78	1	1	1	3	0
79	1	1	1	3	0
80	1	1	1	3	0
81	1	1	1	3	0
82	1			1	1
83	1	1	1	3	1
84	1			1	0
85	1			1	0

B. Python Implementation of Categorical Naive Bayes

```
import pandas as pd
class CategoricalNaiveBayes:
    def __init__(self, lap_corr = 0.0001, threshold = 1):
        """Initialization of Categorical Naive Bayes"""
        self.lap_corr = lap_corr
        self.list_of_probs = ''
        self.target_values = ''
        self.features = ''
        self.target_threshold = threshold
        self.threshold_category = 'True'

    def __get_feature_categories(self, dataset, features):
        """Grabs and stores all features from a dataset """
        cat_dict = {}
        for feature in features:
            u_categories = list(dataset[feature].value_counts().index)
            u_categories.sort()
            cat_dict[feature] = u_categories
        return cat_dict

    def __create_prob_df(self,features,feature_categories, target_values):
        """Creates the probability table that stores the final probabilities
        used for classification"""
        prob_dict = {}
        for feature in features:
            temp_df = pd.DataFrame(0, index = feature_categories[feature],
                                   columns = target_values)
            prob_dict[feature] = temp_df
        return prob_dict

    def __get_count_of_targets_from_unique_set(self,
                                               dataset,
                                               features,
                                               category,
                                               target_values):
        """Counts how many tuples are classified as target class"""
```

```
        big_dict = {}
        for feature in features:
            temp_dict = {}
            for j in range(len(category[feature])):
                temp = dataset[dataset[feature] == category[feature][j]]
                temp = temp.iloc[:, -1]
                temp = temp.sort_values()
                temp = temp.value_counts()
                if len(temp) < len(target_values):
                    temp = self.__add_missing_targets(temp, target_values)
                temp_dict[category[feature][j]] = temp
            big_dict[feature] = temp_dict
        return big_dict

    def __add_missing_targets(self, temp, target_values):
        """Used for when a target is not included in the training
        set and adds it to the unique count"""
        targets_hit = list(temp.index)
        add_targets = list(target_values)
        for target in targets_hit:
            add_targets.remove(target)
        for target in add_targets:
            temp[target] = 0
        return temp

    def __get_probabilities(self,feat_cat_target_counts,
                            target_categories,
                            features,
                            feature_categories):
        """Gets the probabilities of each category and stores them"""
        for feature in features:
            for category in feature_categories[feature]:
                for i in range(len(target_categories)):
                    self.list_of_probs[feature].loc[category, target_categories.index[i]] =
feat_cat_target_counts[feature][category][target_categories.index[i]] / target_categories[i] +
self.lap_corr

    def fit(self, dataset):
```

```
        """Method that trains the classifier"""
        self.features = list(dataset.iloc[:,:-1].columns)
        target_categories = dataset.iloc[:,-1].value_counts()
        target_categories = target_categories.sort_index()
        self.target_values = list(dataset.iloc[:,-1].value_counts().index)
        self.target_values.sort()
        feature_categories = self.__get_feature_categories(dataset, self.features)
        self.list_of_probs = self.__create_prob_df(self.features, feature_categories,
self.target_values)
        feat_cat_target_counts = self.__get_count_of_targets_from_unique_set(dataset,
self.features, feature_categories,self.target_values)
        self.__get_probabilities(feat_cat_target_counts, target_categories, self.features,
feature_categories)

    def __set_pred_dict(self):
        """Sets up the dictionary used for predicting classes"""
        target_guess = {}
        for i in range(len(self.target_values)):
            target_guess[self.target_values[i]] = 1
        return target_guess

    def __get_max_key(self, target_guess):
        """ Gets the max from a dictionary"""
        target_guess[self.threshold_category] = target_guess[self.threshold_category] *
self.target_threshold
        values=list(target_guess.values())
        keys=list(target_guess.keys())
        return keys[values.index(max(values))]

    def predict(self, X_test):
        """Method used to predict using the test set or new data"""
        y_pred = []
        for i in range(len(X_test)): # iterate through observations
            target_guess = self.__set_pred_dict() # Set
            for target in self.target_values: # iterate through targets
                for feature in self.features:
                    target_guess[target] *= self.list_of_probs[feature].loc[X_test.loc[i,feature], target]
            obs = self.__get_max_key(target_guess)
```

```
            y_pred.append(obs)
        return y_pred

'''End of Scope'''
```