Dementia

December 7, 2020

Contents

1 Data Sources **1**
- 1.1 Datasets Used . . . 1
- 1.2 Dementia Output . . . 2

2 Split Into numeric and textual **2**

3 Numeric Features **2**
- 3.1 Informant and High Percentage of Missing . . . 3
- 3.2 Nested Questions and High percentage of missing . . . 3
- 3.3 Treating Features with High Percentage of Missing . . . 4
- 3.4 Detecting Erroneous Values Inside Features . . . 5
- 3.5 Working with Legal Features . . . 5
- 3.6 Filtering Legal Features with Erroneous Values - Erroneous Code-Book . . . 5
- 3.7 Detecting Outliers . . . 6
- 3.8 Outlier Detection and Scaling . . . 6
- 3.9 Feature Cross . . . 7

4 Feature Selection **7**
- 4.1 Decision Tree Feature Importance . . . 7
  - 4.1.1 Select K Best Features . . . 7
  - 4.1.2 Deciding "K" . . . 7
  - 4.1.3 Selecting K Best Features . . . 7

5 Oversampling and Undersampling **7**

6 Chapter 6: Precision, Recall, and F-measure **8**

7 Chapter 7:ROC Curves and Precision-Recall Curves **8**
- 7.1 ROC Curve . . . 8
- 7.2 AUC . . . 8

8 Chapter 8: Probability Scoring Methods **8**
8.1 Probability Metrics 8
8.2 LogLoss Score 9

9 Cross Validation for Imbalanced Datasets **9**

10 Chapter 10 **9**

11 Chapter 12 **9**
11.1 Oversampling 9
11.2 Undersampling Techniques 10

12 Data Transforms (Hiyam) **10**
12.1 Scaling Numeric Data (chapter 17) 10
12.2 Scaling Data with Outliers (chapter 18) 10
12.3 How to Encode Categorical Data (chapter 19) 11
12.4 How to Make Distributions Look More Gaussian (chapter 20) . . 12

13 Data Pre-Pocessing **12**
13.1 Imputation in Ordinal/Categorical Data 12
13.2 Imputation Currently Done 12
13.3 Scaling 12
13.4 Encoding Data 13

14 Advanced ML Evaluation Techniques **13**
14.1 Analysis of Predictive Models 13
14.2 Evaluation Using Traditional Metrics 13
14.3 Problem with Standard Evaluation Metrics 14
14.4 Solution: Risk Estimates 14
14.5 Ensuring Quality Of Risk Estimates 15
14.5.1 From Models to Risk Estimates 15
14.5.2 Measuring the Goodness of Risk Scores 15
14.5.3 Comparative Evaluation of Risk Estimates 16
14.6 Interpreting Classifier Outputs - FP Growth 17
14.7 Technicalities 17
14.8 Characterizing Prediction Mistakes 18
14.9 Comparing Classifier Predictions 18

1 Data Sources

1.1 Datasets Used

The datasets used are from the following dementia surveys conducted in Lebanon:

- Bekaa Questionnaire: This dataset is the most recent survey conducted last year in the Bekaa. This dataset involved 219 elderly.

- full validation data: This data consists of 281 rows. This is the very first survey that we implemented for the sake of testing the 1066 Dementia SPSS algorithm. This survey was conducted on a 1-1 ratio with clinically diagnosed demented patients.
- dementia data 502 baseline updated: This is the second survey implemented which involved 502 elderly people. The selection of elderly to participate in this survey was random.

We then merged this dataset into one dataset consisting of a total of 1024 elderly.

1.2 Dementia Output

In order to label each row (elderly) as positive (has dementia) or negative (does not have dementia), we run the 1066 Dementia SPSS algorithm on our dataset. The result was a total of 1024 patients, 203 of whom have dementia according to the SPSS algorithm.

2 Split Into numeric and textual

We have split the data into two chunks, one containing numeric features and another containing categorical features. The Bekaa and full validation questionnaires involved textual questions as well. However, these textual questions are not used by the dementia 1066 SPSS algorithm in determining whether the elderly has dementia or not, so we dropped these textual features from the analysis.

3 Numeric Features

Our data is now a mix of numerical and categorical features. For the numeric features, we have obtained the following information:

- **data type:** Type of the numeric feature. Could possible be:
    1. **numeric** meaning this feature has continuous range of values
    2. **categorical** meaning this feature has only a finite set of values, each resembling a specific category
    3. **ordinal** meaning, this feature's values are categorical, but they can be compared between each other. Example would be: a job, were a specific job might be values more than the other. Example: manager vs assistant

  the specification of the **data type** of each feature was done manually by looking at the range of values in each, and also by looking at the choices workbook in the copy of dementia Excel questionnaire

- **Description:** Description of each feature
- **cat_options:** Short for "categorical options". *In case the feature is categorical*, what are the possible categories.
- **val_range:** Short for "values range". *In case the feature is numerical*, what are the range of values this feature takes
- **min:** *In case the feature is numeric*, what is the minimum value it has
- **max:** *In case the feature is numeric*, what is the maximum value it has
- **distribution:** link to the distribution plot of each feature on bitbucket. The distribution plot helps in detecting erroneous data
- **perc_missing:** Short for "percentage of missing values". This helps us ggregate features with high percentage of missing. In case the percentage of missing is very high, Imputation methods won't aid much
- **perc_erroneous:** Short for "percentage of erroneous values". Detecting erroneous values was done manually for each feature. Anomalies cover uni-intentional or intentional erroneous values. Example would be having "age" feature being 1966 instead of the actual age
- **erroneous:** What were the erroneous values, if any
- **cut_off:** The cut off used to decide whether a feature's value is erroneous or not. The cut off was discovered manually by us for each feature.
- **color code:** a color code was applied to each feature to determine whether it belongs to the informant or not.

3.1 Informant and High Percentage of Missing

Most of the features associated with very high percentage of missing were directed to informants and related to the living conditions of the family, the house, income, cor-residents of the elderly, etc... which are not used by the SPSS algorithm to label the patients as demented or not (since they are not about the patients) For this reason, we dropped the questions that were about the household, informants, and co-residents

* informant is usually the caregiver of the interviewed elderly person

3.2 Nested Questions and High percentage of missing

Since our data is survey in nature, some features are actually questions are related to the other. If a certain question is asked to the elderly, and only based upon the elderly's answer, the interviewer might ask the elderly another question or not. Let us give an example:

1. the interviewer asks if there has ever been a period when the elderly smoked cigarettes, cigars, a pipe, or water pipe nearly every day?
2. **Only if the answer to the question above was a Yes**
3. the interviewer asks the following questions:
   (a) What did the elderly smoke?
   (b) How old were you when you started using tobacco regularly? (if the above is true)
   (c) Do you still use tobacco regularly?

We also realized that many features have high percentage of missing because a parent question's answer did not cause a jump to that feature question.

3.3 Treating Features with High Percentage of Missing

We divided the features as follows:

1. **Informant** A feature belongs to this category if the question is asked to the informant
2. From Informants we have:
   (a) **parent** Questions that caused jumps to other questions
   (b) **child** child of parent question
3. **Non-Informant** We also have:
   (a) **parent**
   (b) **child**

We followed the following steps in order to eliminate features with high percentage of missing:

1. remove all informants and their children questions (so we removed all informants, and if one of them happens to be a parent question, we removed all its children questions)
2. It is important to note that all parent questions informants have their children questions also informant
3. Then we are left with **Legal features**
4. From legal features, remove all children

After doing the steps above, the number of numerical features decreased from 542 to 257, all of which have 0% missing (with the exception of 1 feature whcih has 50% missing)

3.4 Detecting Erroneous Values Inside Features

In our dataset, the categorical and ordinal values are encoded with numbers ranging from 1 to the length of the categories. therefore, since our categorical, ordinal, and numerical values are all numbers, we set cut-offs to determine erroneous values. For some of the features, the erroneous values are straight forward to detect, based on the description for the question. These are the following:

- age: some people enter the year, others enter the age. for this, if the value of both the numb as is or 2020 - the numb is more than 100, this means that the person is more than 100 years old which renders the age erroneous. For age, the cut-off was 100
- helphour: number of hours per week the elderly needed help, therefore bounded by max numb of hours in a week
- learn questions: the patient is supposed to repeat 3 words, bounded by the max numb of words: 3

For the rest of the categorical and ordinal values, we get the maximum number encoding for the question options from the options excel sheet based on which the interviewers selected these options. The cut-off is therefore the maximum number encoding a category. Were therefore label any value that is greater than the cut-off to be erroneous.

3.5 Working with Legal Features

As mentioned in section 3.3, the remaining ***"Legal"*** Features are the 257 out of originally 542. We will be using these 257 ***"Legal"*** features as input for data pre-processing and cleaning techniques.

3.6 Filtering Legal Features with Erroneous Values - Erroneous Code-Book

In section 3, we talked about how some of the 542 numeric features have erroneous values. However, as we will be working with only "Legal" features, its important to filter out which subset of the features with erroneous values are actually "Legal". Thats's why, we did the following:

1. From all the features containing erroneous, filter out the ones that are Legal
2. From the Legal ones:
   (a) get the feature name
   (b) get the feature's description
   (c) get the feature's percentage of erroneous values

(d) get the feature's actual erroneous values

(e) Display the feature's cut-off on values which decides which values are erroneous (crossing the cut-off) and which are non-erroneous (not crossing the cut-off)

3. When done from 2. above, sort the features by decreasing order of percentage of erroneous values

3.7 Detecting Outliers

We want to investifgate the existsence of outliers in the data. We have 3 different data types:

1. **Numeric**: very few columns
2. **Ordinal**: categorical values that obey a certain order of importance
3. **Categorical**

The outlier detection methods are known for numeric values, but it is **ordianl** and **categorical** values that raise the question of how are we going to detect outliers with such types of data ?
We will assume that ordinal values are numeric. We will prove the legit-ability of our assumption through an example:
Assume for instance, that we have a column that shows the socio economic status ("low income","middle income","high income"), education level ("high school","BS","MS","PhD"), income level ("less than 50K", "50K-100K", "over 100K"), satisfaction rating ("extremely dislike", "dislike", "neutral", "like", "extremely like"). These are not categorical but rather ordinal, and if we give, for example, the income level the following labels:

- less than 50K: 0
- 50K-100K: 1
- over 100K: 2

Then it is definitely the case that $2 > 1 > 0$. And if we have an individual who earns, $>$ 1000 K, we give this the label 3, and these individuals are definitely rare and can be considered as outliers

3.8 Outlier Detection and Scaling

We try different scaling methods before we detect outliers, and we apply the outlier detection only for columns *that are: numeric and/or ordinal*, and we extracted the percentage of outliers for each of the legal columns as well as the outlier values themselves (values considered outliers).
We realize that the outlier values, for the **ordinal** columns, happen to be the values *8 and 9* which is normal because the values are either 0,1,2 or 8,9.

3.9 Feature Cross

Our data mostly consists of categorical/ordinal columns, and we have to find a way to solve these non-linearities such that it fairs better for the model's understanding of the features when we begin with the modelling phase. We generate the list of all possible feature crosses.
Feature crossing, will come after we 'one-hot encode' our data.

4 Feature Selection

4.1 Decision Tree Feature Importance

We use decision trees to calculate feature importance implemented using scikit-learn's DecisionTreeRegressor.

After being fit, the model provides a feature_importances_ property that can be accessed to retrieve the relative importance scores for each input feature.

The results of the feature importance using the dummy imputed pooled data so far is then stored in the 'feature_importance.csv', sorted from highest (highest importance score) to lowest (lowest importance score).

4.1.1 Select K Best Features

4.1.2 Deciding "K"

In order to select K best features, we need to first decide the value of K. In order to do this, we use sklearn's RFECV, which uses step forward recursive feature elimination and cross validation to decide the number of features needed to give the highest out of sample accuracy.

4.1.3 Selecting K Best Features

After deciding on the value of K, which using the current imputation of the input data turned out to be 20, we perform step forward recursive feature elimination that ranks all the features as 0 or 1, with 1 meaning that a feature is one of the K best features. We store the names of the k-selected features (column names) into a csv file called "k_best_features.csv"

5 Oversampling and Undersampling

Based on the examples discussed in the books, we implemented 4 functions that accept a model and add it to a pipeline after adding the oversampling/undersampling layers. The following are the functions created:

1. random_sampling: consists of a random oversampling and a random undersampling layers

2. smote_random_sampling: adds a smote oversampler and a random undersampler

3. smote_tomek (combination approach): combines over and under sampling using SMOTE and Tomek links.

4. smote_enn: combines over and under sampling using SMOTE and Edited Nearest Neighbours

6 Chapter 6: Precision, Recall, and F-measure

The chapter summarizes the precision, recall, and f-measures of multiclass and binary classification for the case of balanced data.

7 Chapter 7:ROC Curves and Precision-Recall Curves

ROC Curves (receiver operating characteristic curve) and ROC AUC (area under the ROC curve) are used to decide if the model differentiates between the labels.

7.1 ROC Curve

The threshold is applied to the cut point in probability between the positive and negative classes. In order to decide on the best threshold, we evaluate the true positive and false positives for different threshold values, a curve can be constructed that stretches from the bottom left to top right and bows toward the top left. This curve is called the ROC curve. A classifier that has no discriminative power between positive and negative classes will form a diagonal line between (0,0) and (1,1). Models represented by points below this line have worse than no skill.

7.2 AUC

Area under the curve is calculated to give a single score for a classifier model across all threshold values. This is called the ROC area under curve or ROC AUC or sometimes ROCAUC. The score is a value between 0.0 and 1.0, with 1.0 indicating a perfect classifier.

8 Chapter 8: Probability Scoring Methods

8.1 Probability Metrics

On some problems, a crisp class label is not required, and instead a probability of class membership is preferred. The probability summarizes the likelihood (or

uncertainty) of an example belonging to each class label.

8.2 LogLoss Score

Logarithmic loss or log loss for short is a loss function known for training the logistic regression classification algorithm. The log loss function calculates the negative log likelihood for probability predictions made by the binary classification model. Most notably, this is logistic regression,but this function can be used by other models, such as neural networks, and is known by other names, such as cross-entropy.

9 Cross Validation for Imbalanced Datasets

The solution is to not split the data randomly when using k-fold cross-validation or a train-test split. Specifically, we can split a dataset randomly, although in such a way that maintains the same class distribution in each subset. This is called stratification or stratified sampling and the target variable (y), the class, is used to control the sampling process. This is available using sklearn stratified kfold.

10 Chapter 10

Summarizes oversampling and undersampling techniques which are available in chapters 12 and 13 in details.

11 Chapter 12

11.1 Oversampling

All techniques available through the python library: imblearn.over_sampling

- Random Oversampling: simplest oversampling method, involves randomly duplicating examples from the minority class in the training dataset
- SMOTE (Synthetic Minority Oversampling Technique): works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample as a point along that line
- Borderline-SMOTE: involves selecting those instances of the minority class that are misclassified, such as with a k-nearest neighbor classification model, and only generating synthetic samples that are difficult to classify. Borderline Oversampling is an extension to SMOTE that fits an SVM to the dataset and uses the decision boundary as defined by the support vectors as the basis for generating synthetic examples, again based

on the idea that the decision boundary is the area where more minority examples are required.

- Adaptive Synthetic Sampling (ADASYN): another extension to SMOTE that generates synthetic samples inversely proportional to the density of the examples in the minority class. It is designed to create synthetic examples in regions of the feature space where the density of minority examples is low, and fewer or none where the density is high.

11.2 Undersampling Techniques

- Random Undersampling

12 Data Transforms (Hiyam)

12.1 Scaling Numeric Data (chapter 17)

- Normalization (Min-Max Scaling)
- Standardization

12.2 Scaling Data with Outliers (chapter 18)

Many machine learning algorithms perform better when **numerical** input variables are scaled. Standardization is a popular scaling technique that substracts the mean from the values and divide by the stadard deviation, transforming the probability distribution from for an input variable to a Standard Gaussian (zero mean and unit variance).
Problem: Standardization can become skewed or biased when the input variable contains outliers.

Robust Scaling

When we are scaling the data, and if we have very large values relative to the other input variables, these large values can dominate or skew some machine learning algorithms. **The result is that the algorithms pay most of their attention on the large values and ignore the variables with smaller values**.
Outliers are values at the edge of the distribution that may have a low probability of occurence, yet are overrepresented for some reason. **Outliers can skew a probability distribution and make data scaling using standardization difficult as the calculated mean and standard deviation will be skewed by the presence of outliers.**
Approach: When standardizing input variables containing outliers, we can ignore outliers from the calculation of the mean and standard deviation, and use the calculated values to scale the data

This is called robust standardization. This can be achieved by calculating the median (50th percentile) and the 25th and 75th percentile. The values of each variable can then have their median subtracted and are divided by the inter quartile range (IQR) which is the difference between the 25th and 75th percentile.

$$value = \frac{value - median}{p_{75} - p_{25}} \tag{1}$$

The resulting value has a **zero mean and median and a standard deviation of 1**. Although not skewed by outliers and the outliers are still present with the same relative relationships to other values.

12.3 How to Encode Categorical Data (chapter 19)

Machine learning models require all input and output variables to be numeric. This means that if the data contains categorical data, we must encode it to numbers before we fit and evaluate our models.

Ordinal Encoding

Each unique category is assigned an integer value. Example: *red* is 1, *green* is 2, *blue* is 3.
For categorical variables, it imposes and **ordinal relationship** when no such relationship exists. This may cause problems and **one hot encoding** may be used instead.

One Hot Encoding

For categorical variable where no ordinal relationship exists, the ordinal encoding is not enough and may be misleading.
One Hot Encoding works by creating binary variables for each category. For Example: *red* will be [1, 0, 0], *blue* will be [0, 1, 0], and *green* will be [0, 0, 1].

Dummy Variable Encoding

The one hot encoding includes a binary representation for each category. This might cause redundancy.
if we know that [1, 0, 0] represents *blue*, and [0, 1, 0] represents *green*, we don't need another binary variable to represent *red*, instead we could use 0 values along, e.g. [0, 0]. This is called dummy variable encoding and always represents C categories with $C - 1$ binary variables.
Other being less redundant, it is required for some models.

12.4 How to Make Distributions Look More Gaussian (chapter 20)

Several Machine Learning Algorithms assume the numerical values have a Gaussian distribution. Our data may not have a Gaussian distribution and it might have a Gaussian-like distribution.

13 Data Pre-Pocessing

After we have filtered the legal features in our data, and we deleted the features that have > 40 % missing values, we are left out with 255 legal features. These features must be pre-processed before we do the modelling exercise, for the following reasons:

1. All the 255 legal features still have missing values
2. We have a combination of numeric, ordinal, and categorical features, whereby each might require a different kind of scaling

13.1 Imputation in Ordinal/Categorical Data

To be continued ... waiting to meet with Dr Khalil again. Some notes:

1. We said we might do KNN and impute by getting the values from the nearest neighbors.
2. It is common that public health practitioners usually disregard features with greater than 25% missing, and since we have features that have greater than 25% missing, we got the **theme** of each feature, and if the feature belongs to a particular theme that might not be of vital importance for detecting dementia, we will dis-regard the feature by itself.

13.2 Imputation Currently Done

Just for now, to get things going, we imputed all missing values by replacing with the majority

13.3 Scaling

We have said earlier that we have three data types in our features: ***numeric, ordinal, and categorical.***

- For ordinal and numeric data, we will be using the **Robust scaling** mechanism due to the presence of outliers
- For categorical data, we will not scale it as we will be applying **dummy variable encoding**

13.4 Encoding Data

- For ordinal features, we will be applying Ordinal Encoding, as it will maintain an order between feature values.
- For categorical features, we apply dummy variable encoding that decreases redundancy of one hot encoding

14 Advanced ML Evaluation Techniques

In this paper, authors seek to solve the problem of predicting students who are at risk of dropping out of highschool or not. They have a **binary classification problem** and they predict if the student graduates (0) or does not (1). School principals want to see if they can have enough budget to afford those - **at an early stage** - that are at risk of not graduating and help them with additional materials. Non-Machine learning practitioners do not value regular evaluation metrics like precision and recall because it means nothing to them, they want some evaluation techniques that could help understand the predictions more from their point of view

Disclaimer

- I have read only sections 5 & 6 from the paper as instructed by Dr. Fatima
- I have found a github repository for their work here: https://github.com/dssg/student-early-warning but their work is not complete at all, the only thing that they did - with regards to sections 5 & 6, is compute the ***risk***. The rest of the work is done solely by me.
- I used the data they provided in their github repository to complete my work, they also do not provide the complete data and there are differences between the column names mentioned in the paper and the columns in the dataset they provide. But I still worked with this incomplete version of the data

14.1 Analysis of Predictive Models

School districts are interested in identifying those students who are are at risk of not graduating high school on time so that plan their resource allocation ahead of time.
Goal: predict if a student is at **risk** of not graduating high school on time

14.2 Evaluation Using Traditional Metrics

- We have a binary outcome, use standard evaluation metrics such as *accuracy, precision, recall, f-measure, and AUC*

- Their data is **imbalanced**:
    1. **Class 0: 91.4%**
    2. **Class 1: 8.6 %**
- I must do a **stratified** split of classes between training and testing such that if the original data has X% 0s and Y% 1s, then the training as well as the testing must also have X% 0s and Y% 1s
- I have added **SMOTE** so that we can **oversample** our **training** data before predicting
- I used MinMax Scaling (just for now because my focus is on the implementation of the ideas in sections 5 & 6)
- In this paper they only use shallow models, this what I also do (just for now because my focus is on the implementation of the ideas in sections 5 & 6)
- In Figure 1 below, I show the ROC curves of various models

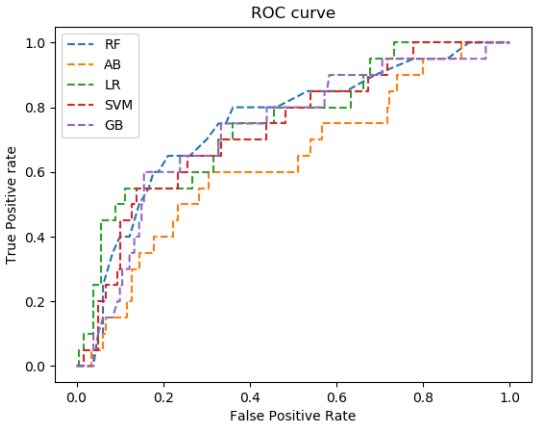

Figure 1: ROC Curves of various shallow models trained on the data provided

14.3 Problem with Standard Evaluation Metrics

1. Educators think about the performance of an algorithm in a slightly different fashion
2. the availability of resources varies with time. For example, a school might support 100 students in 2012, and 75 in 2013.
3. We want to build algorithms that can cater to these changing settings

14.4 Solution: Risk Estimates

- an algorithm, to cater to their needs, must provide them with a list of students ranked by some measure of *risk* such that, the students at the top of the list are at a higher risk of not graduating on time.
- Once educators have such a ranked list, they can choose the **top k** students from it and provide assistance to them
- **Challenge:** We only have binary records, but fortunately, all classification algorithms provide internally a probability of each class we have, we can use this probability to derive our **risk** measure

14.5 Ensuring Quality Of Risk Estimates

14.5.1 From Models to Risk Estimates

- The output of our predictions are binary: either 0 (will graduate on time) or 1 (will not graduate on time)
- Algorithms allow us to get the **probability** of each class. Here, we focus on the probability of the **positive** class (which shows the probability of not graduating on time). We use the probability of the predictions on the **final testing data**

14.5.2 Measuring the Goodness of Risk Scores

- We first get the probability of not graduating on time (class: 1) for each of the testing instances (students) we have. We call this ***risk***
- Rank the testing instances (students) in descending order of **risk** estimates. This way, students with higher probability of not graduating on time are at the top of the list.
- Group students into **bins (percentiles)** based on their risk scores. We can decide, for example, to choose 10 bins, and when we do, then the students who fall between the 10th and 20th percentile have very little risk scores, those between 20th and 30th have more risk scores than the latter
- My problem with this is as follows: we are doing the 3 bullets above for all the algorithms we have. The distribution of risk scores, for un-balanced classification problems, is usually very skewed and therefore, we can have the following 2 scenarios:
  - We might not be able to group them into 10 bins for example because each bin must have the same number of instances and that might not be possible in the un-balanced skewed distribution
  - A possible solution to the problem above is to decrease the number of bins.

- Even if we decrease the number of bins, we might not be able to attain the same number of bins for every algorithm we have. If you look at Figure 2 in their paper, you see that for all algorithms they were able to distribute risk scores into 10 bins, but in case one is not able, we cannot produce such a plot
- One solution is to distribute them into bins but **without guaranteeing the same number of instances in each bin**
- Even if we are able to produce such a plot, the 10 bins won't be unique (in terms of lower and upper limits) for each algorithm. Is that valid to work with ?

- For each bin, we produce the **mean empirical risk** which is the fraction of students, from that bin, who actually (as per ground truth) fail to graduate on time.
- This curve is called **mean empirical risk** curve
- An algorithm is considered to be producing good predictions if its empirical risk curve is **monotonically non-decreasing** (as the risk scores increase, we have more **correct** predictions about the positive class (more students who actually - as per ground truth - fail to graduate on time))

I present below the empirical risk curve of the shallow models I trained on - **with incorporating SMOTE** and MinMax Scaling

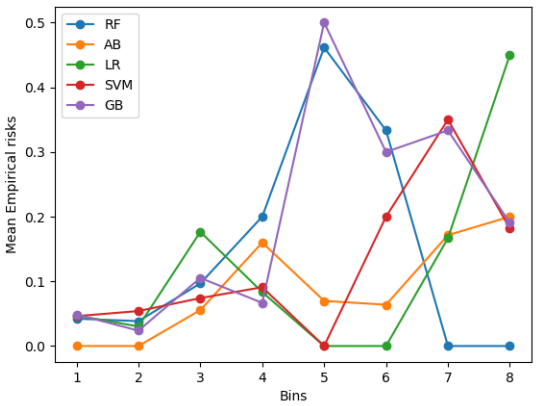

Figure 2: Mean Empirical Risks per Model

14.5.3 Comparative Evaluation of Risk Estimates

It is good to see the models performances on the **Top K** students **at risk**. The steps we do to attain this:

1. Rank Testing instances (students) by their **risk** scores
2. Define a list of **K**s
3. For each value **k** of **K**:
   (a) get the top **k** predicted values
   (b) get the ground truth of these **top k**
   (c) compute the precision/recall

We get the precision and recall at top K and we produce the following curves:

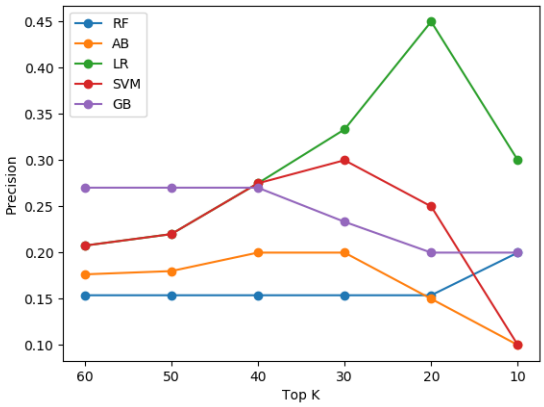

Figure 3: Precision at Top K

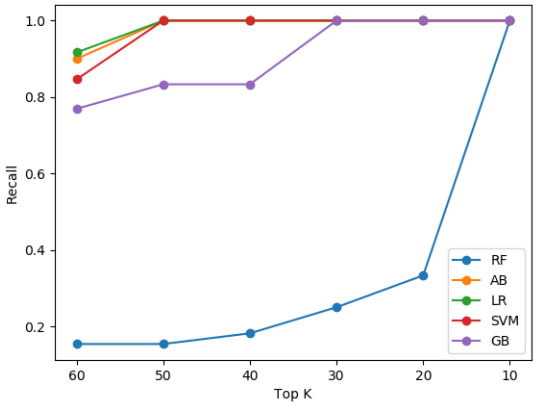

Figure 4: Recall at Top K

14.6 Interpreting Classifier Outputs - FP Growth

Frequent Patterns

To understand FP Growth algorithm, we need to first understand association rules.

Association Rules uncover the relationship between two or more attributes. It is mainly in the form of- If antecedent than consequent. For example, a supermarket sees that there are 200 customers on Friday evening. Out of the 200 customers, 100 bought chicken, and out of the 100 customers who bought chicken, 50 have bought Onions. Thus, the association rule would be- If customers buy chicken then buy onion too, with a support of 50/200 = 25% and a confidence of 50/100=50%.

References: https://www.mygreatlearning.com/blog/understanding-fp-growth-algorithm/

14.7 Technicalities

Usually, for identifying frequent patterns, we are represented with a **transactions** dataset, where each row represents **a set of items** bought by a customer. **However, we do not have transaction, we rather have multivariate dataset with a bunch of numerical columns. How can we do it?**

Disclaimer: The methodology below presents my own solution of the problem, I am not able to find such a technique anywhere on the internet.

In order to **present** values as items:

1. Consider Each column value, in a row, as being an **item** bought by the customer.
2. Our values are numeric, therefore, to avoid redundancy, I categorize the values as follows:
    - value <= 25th percentile
    - 25th percentile < value <= 75th percentile
    - value > 75th percentile
3. Do the categorization above, **per column**, for all values in the dataset
4. Achieve the so called "Item Dataset" where each row has one item (from the categories generated above) per column
5. Apply the FP-growth Technique on the generated "Item Dataset"
6. Extract the most frequent patterns

The reason I did the aforementioned methodology above is because, in the **paper we are referencing (montogemery)**, they have frequent patterns like (GPA > 2.0) and (Absence rate <= 0.1), therefore I deduced that these can be taken by doing 'quartiles' of the distribution of values we have per column.

14.8 Characterizing Prediction Mistakes

1. Get all frequent patterns using the methodology I created above that incorporates FP-Growth
2. Rank predictions based on risk score probabilistic estimates
3. Create a new field called *mistake* which is 1 if the prediction does not match ground truth and 0 otherwise
4. **For each frequent pattern** identify the **probability of mistake** by computing the number of mistakes done per frequent pattern
5. Do all of the above bullets **for each model. Therefore, we end up with a probability of mistake for each frequent pattern, per model**

14.9 Comparing Classifier Predictions

When we present educators with a suite of algorithms, they are keen on understanding the differences between *rank orderings* produced by each of these algorithms.
We will be using **Jaccard Similarity** for measuring similarity between predictions, **also at Top K**:

Let's say we have 200 testing instances (students). One model might put the highest risk score for the 180th testing instance, another model might put the highest risk score for the 190th testing instance. Therefore, it is important to note that in this exercise, we find which testing instances were chosen to be in the top k between model 1 and model 2, and we compute, using the jaccard similarity, the intersection of these testing instances (over their union). Good models must have very high intersections if they are classifying instances properly.

1. Get all possible **combinations** of model pairs (we are using a suite of models)
2. For each model pair, and for each **k** in **K**:
   (a) get the testing instances at top **k** from model 1 of the model pair
   (b) get the testing instances at top **k** from model 2 of the model pair
   (c) Compute jaccard similarity as the intersection of the two testing instances from each model over their union

After doing this, we get the following results:

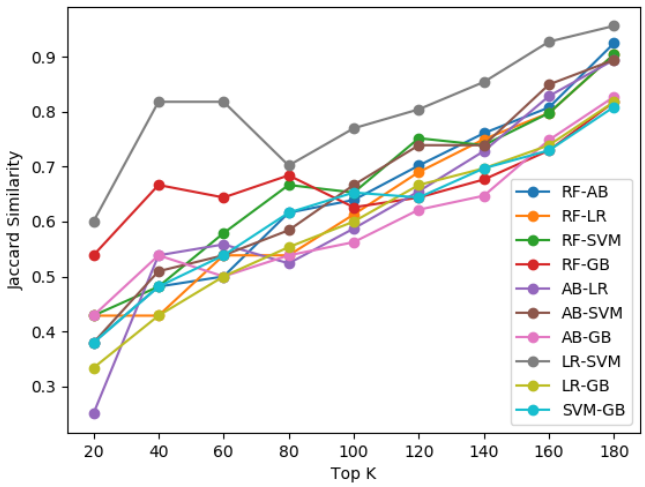

Figure 5: Jaccard Similarity of students at risk for various algorithms