IDENTIFYING UNDERGRADUATE STUDENTS AT RISK OF DROPPING OUT OF MATHEMATICS

RITVIK KHARKAR AND JESSICA TRAN

1. Introduction

.. todo ...

2. Machine Learning Pipeline

In this section of the project, we will be using various machine learning models to predict whether or not a student will drop out of their mathematics major within a specified time frame. While the success of our project is directly linked to the quality of our predictions, there are several steps, starting with a raw csv data file, which lead up to generating our predictions.

Indeed, we have chosen to structure our project in the format of a 'pipeline', a directed acyclic graph. Each node in the graph represents a data object such as a csv file, a directory of Python pickle files, a success file, etc. Each directed edge in the graph represents the task (an underlying Python script). The source of the edge represents the necessary input into the task and its destination represents the output of this task, if run to completion.

A visual representation of our current pipeline for this project, each of whose components will be discussed, is shown below.

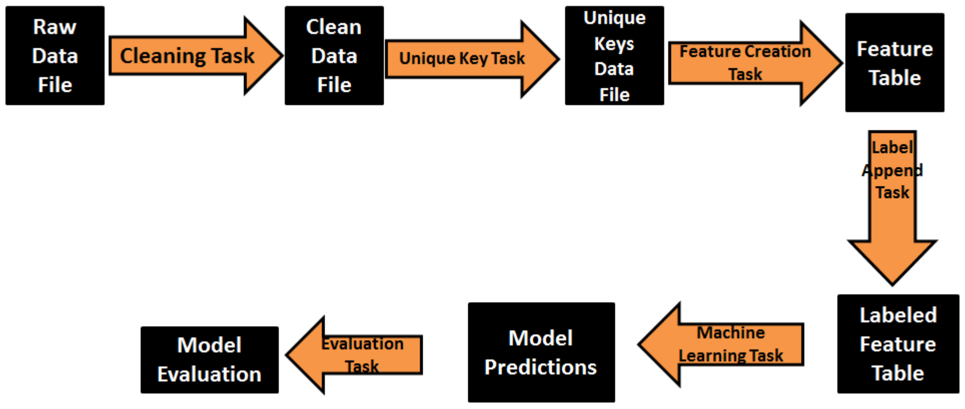

Figure 1. Pipeline for this project

In the pipeline as it is now, our input is a raw data file, provided by the department and out output is a table where each row is a model that was run on the data as well as information about the quality of that model.

We will now discuss each piece of this pipeline in detail.

2.1. Cleaning Raw Data.

The first step in our pipeline is cleaning the raw data provided by the department. We will discuss three classes of cleaning that we employ: homogeneous column removal, type conversion, and imputation of missing values.

First, any columns in the raw data who have a homogeneous value through the whole dataset are removed. They are removed because any column who shows no variation between individual points in the dataset

Date: November 20, 2016.

cannot be used to predict anything about the dataset. But also note that it is important to check for this homogeneity with each additional batch of new data in case new rows have different values for this columns.

Second, we convert the data types of certain columns into a type that is more easily manipulated later in the project. As a concrete example, the raw data treats 'Term' as a string, for example '14S' for spring quarter of 2014. Since we are often later concerned with subtracting terms in order to get measures of time between certain events, we would much rather have our term be a floating point number. Thus, in this cleaning script we decide on a convention to translate each string term into a floating point term. In particular, we change 14W to 14.0, 14S to 14.25, 14U to 14.5 and 14F to 14.75, and similary for other years. We choose this convension so that subtracting one term from another yields the number of years in between. For example, 15.75 - 14.25 = 1.5 and indeed there are 1.5 years, of 6 quarters between 14S and 15F.

Last, there are some (and sometimes many) missing values in our dataset for various columns. We have some options here. We can choose to ignore the entire column or we can choose to impute the missing values using the present values. For example, if our grade column contains 50 missing values out of the 220,000+ entries in this column, we might choose to take the median of the existing values and use this as the grade value of each missing value. We may, to be more accurate, use as the missing value, the same grade the student recieved in all other courses that quarter, etc.

Our output after this step is a clean dataset with no new columns compared to the raw dataset but with possible type differences between columns, some columns removed, and extra imputed values.

2.2. **Attaching Unique Keys.** In our project, we found that there was no native unique key to identify each row in our dataset. As an example of a unique key, a dataset about cars might use the VIN number of the car as the unique identifer because no two cars have the same VIN number. We are thus in a sense protected since no matter what new cars enter our database, we will never have a new row with the same VIN number as an existing entry.

In our project, we cannot simply use the (hashed) student ID number as our unique identifier because the same student can, and often does, appear multiple times in our dataset. This is because our dataset contains one entry for each instance of a particular student taking a particular course in a particular quarter. Thus, it is not even enough to treat each double of (student ID, term) as our unique identifier since a student can take multiple courses in a given term. So, we choose any given triple of (student, course, term) as our unique identifer since no student can take two copies of the same course in any given quarter (and indeed we validate this fact beforehand!).

Thus, we start with the clean data table and attach a column with the (hashed) triple of (student ID, term, course) to each row.

2.3. **Generating Features.** This is arguably the most important phase of the project. Indeed, in machine learning, the discipline called "feature engineering" is highly emphasized because standard machine learning models, operating on sub-par features, will not produce any remarkable results.

We should perhaps start by defining what a "feature" means for our project. We define a feature as any potential metric that will help us predict our label (which will be discussed below). In our case, this means any potential metric which may help us determine whether or not a student will drop our of their mathematics major in the next quarter, year, etc. So, current GPA would likely be an important predictor for this project but average rainfall in the last week might not be.

Our aim is thus to think of, efficiently code, and employ various features to predict student dropout. There are some important factors to consider though, discussed below.

2.3.1. *Object to Predict On.* Even though each row in our current dataset uses the triple (student ID, term, course) as the unique identifier, we acntually do not want to make a prediction for each row in this dataset. The reason is that we are essentially making a binary (0 or 1) prediction about whether or not a given student will drop out of their major in the next quarter, year, etc. Thus if there are two rows in our dataset, one corresponding to the triple (12345, 14F, 115A) and one to the triple (12345, 14F, 131A), it makes little sense to make a distinction between whether student 12345 will drop out in a year after Fall 2014 and her current course is Math 115A or Math 131A. Since course, given student and term, is fixed in time, we want to dimensionally reduce our three dimensional (student, term, course) dataset into two dimensions (student, term).

This means that our resulting feature table will have many fewer rows than our current working dataset. It also means that we will need to project features related to our omitted axis (course) down into our new two dimensional space. If this seems rather abstract, let us discuss a concrete example. Let us assume that we want to include course grade as a feature in our feature table. It no longer makes sense to assign a single course grade to a double of (student ID, term) since a student in a given term will have potentially taken multiple courses. Thus, given a student and term, we can take an average of grades received by that student in all courses prior to the given term as our feature. We can also possible take the max, min, median, or any other function of these grades.

2.3.2. *Timeframe of Features.* We also need to keep in mind how our final machine learning models will be used. Assume (optimistically) that we have a very impressive machine learning model to predict whether or not a student will drop out of their major in the next year based on information about that student in their current term. The key note is that we are only allowed to use information *known* in the current term. Thus, we *cannot* use information about grades received in the current term because this even has not happened yet from the point of view of our prediction. Indeed, imagine if we were able to use future data to predict a current event. We could just find out whether the student graduates from their major or not and make a 100% accurate prediction!

Thus, given a double of (student, term) we restrict ourselves to only use data from before the current term.

2.3.3. *Binarizing Categorical Features.* Our features generally fall into two categories, continuous and categorical. A continuous feature is always numerical and can be thought of as coming from a continuous probability distribution. An example for our data is a student's running GPA in any given term. On the other hand, a categorical feature comes from a discrete probability distribution. Note that it can still be a number. For example, in our data, term is represented as a floating point number but it does not make sense to talk about a term between 14.25 and 14.5 since they represent Spring 2014 and Summer 2014, and there is no term in between. A more obvious example is student gender which, in our dataset is either M for male or F for female.

We wish to use these categorical examples in our machine learning models but cannot just enter 'M' or 'F' into our model. Instead, we need to split up each categorical variable into multiple indicator variables, sometimes called dummy variables. For example, suppose there are only three student majors (untrue): Applied Math, Pure Math, and Financial Math. We will split up this "major" column into two indicator variables columns: isApplied and isPure. Each will take the value 1 if the student is of that major and 0 if she is not. We do not include the variable isFinancial because we want to avoid features which are perfectly correlated with each other. To see that isPure, isApplied, and isFinancial would be perfectly correlated, note that since each student in the dataset must have a major and only one major, and they must be one of these three majors, isPure + isApplied + isFinancial = 1, a perfect correlation. Thus, we only need (and should only use) two of these three variables, but it does not matter mathematically which two we use.

While the theory behind splitting categorical variables into multiple indicator variables is clear, in practice, we can run into issues with memory. As an example, suppose we consider again student major, but now take all the different majors (188 in our current dataset). This means that after our binarization, we will have 186 more columns in our dataset (187 new columns from indicator variables, and we remove the original major column). While not an issue for some databases, the addition of many new columns can cause cause memory errors when using the csv format.

One solution is to use a more suitable database, such as PostgreSQL (which we may do), or think of ways to store the same information using less data. One scheme is to store the indicator variables implicitly in a bitstring or multiple bitstrings and store these bitstrings as integers.

For example, we use 187 bits (0s and 1s) to store the information about student major. We will need to store an external map about which bit index represents which major. Assume that index 100 stands for the Psychology major. Then if we have the 187 bit number 00 ... 010 ... 00 where the 1 is at index 100, we know this means the student is a Psychology major in the current quarter. When converted to decimal format, this bit string represents 2^{100} or 1267650600228229401496703205376. Assuming the size of an integer is 4 bytes or 32 bits, and using the fact that $5 < 187/32 < 6$, we will need 6 columns to store our information about major. This is much better than our original 187 columns. We can have even less columns if we use bigger data types such as double or long double.

2.4. **Attaching Labels.** We have talked briefly about our labels but we define explicitly what this term means for us. We use the term 'label' to reference any attribute about students in a given term we are trying to predict related to dropping out of their major.

Usually, the only axis along we vary our labels is the length of time in the future we are trying to predict. For example, consider two labels drops$_out_in_next_quarteranddrops_out_in_next_year, whichbotheithertakea0or1valuerepresenting$

Our choice of label will change depending on the feasibility of applying our predictions in real time.

This step starts with the feature table and produces the feature table with an extra column, the label.

2.5. **Predicting Student Outcomes.** This is the step which performs the actual prediction given all our work in the previous steps. There are several sub-steps in this step, each of which is described below.

2.5.1. *Train-Test Split.* In order to train our models with data and then test how accurate they are, we will need to use two separate datasets. To see the need for this, imaging that we train our model on some dataset and then test it on this same dataset. This is how we run into the statistical problem called "overfitting". Essentially this means that our model is not actually very strong in general (i.e. on new data) but has been finely tuned to the randomness in our training data. This randomness, which is not the same in new data, will cause our model to perform poorly on this new data. But, if we use this model to make predictions about our training data, it will perform well.

So, we use a training set to train the model and a testing set to test the accuracy measures of the model. We decide this train-test split using time. Our dataset spans 2000 to 2015 so might at first decide, for example, to use the 2000-2010 data for training and 2011-2015 data for testing. This, however, might lead to issues because of the meaning of our labels for students in recent students. Actually, our label makes little sense for currently enrolled students because we do not know if they drop out in the next quarter or year. Thus, we first need to reduce our feature table by chopping off the last x quarters, where x is determined by the prediction interval of our label. With our slightly reduced dataset, we can build our train-test split.

2.5.2. *Choosing our Models.* For this project,we are using the Python coding language, which has a very useful build in modeling library called scikitlearn. This library contains code for various machine learning models such as linear regression, logistic regression, SVM, decision trees, random forests, etc. We will, for this portion of the project use built-in scikitlearn models due to their tried-and-tested reliability and code efficiency.

2.5.3. *Grid Search.* We will choose which models to run by performing a grid search on our model space, defined below. Our model space is a cross product of model types with their respective parameters. For example, assume we are considering the Logistic Regression model, and two of its parameters, C and penalty. Assume furthermore we are interested in two values for C, 0.01 and 0.1, and two values of penalty, l1 and l2. We thus run four logistic regressions, one for each combination of C and penalty. In general, if we run a model with n parameters, with a_1 possibilities for the first parameter, ..., a_n possibilities for the final parameter, we will run a total of $a_1 \times ... \times a_n$ models of this model. We do this for all the different types of models we consider. For each model we run, we will store a pickle file, a serialized Python object. This pickle file will contains a dataframe of each (student, term) double in our feature table, along with the score that model assigns for whether this student will drop out in the next specified time period, and finally a 0 or 1 indicating the true label for this student. This is all the information we need in order to carry out the ensuing evaluation step.

Note that the "scores" that the model outputs are not necessarily 0 or 1 values but rather floating point numbers. The higher these numbers, the more likely the student will drop out in the next specified time interval.

2.5.4. *Future Direction: Feature Importances.* It is evident that not all features have the same "importance" in a prediction problem. For example, to reference a previous example, student grade features are probably more important in predicting dropout than is average rainfall in the past week. Thus, in the future, when we find out which models perform best, we will also store feature importances, which are defined and calculated differently based on the particular model.

By storing feature importances, we will be able to add more features similar to the most predictive features to create even stronger models.

2.6. **Evaluating Models.** The final part of our pipeline is evaluating the predictions of all our models. In order to do this, we will first need to decide, based on the previously mentioned model scores, which scores to map to which prediction, 1 (will dropout in specified timeframe), 0 (will not dropout in specified timeframe). We use a threshold called $k \in [0, 100]$ in order to make this decision. The system works as follows. We sort the scores for any given model in decreasing order and assign the top $k\%$ to get a score of 1 and the bottom $(100 - k)\%$ to get a score of 0.

Now that we have binary predictions for all items in our feature table, we need to find out metrics for how many of these predictions we got right, how many we got wrong, as well as more nuanced metrics about which ones we got wrong.

At the most basic level, we can build a cost matrix, a 2×2 matrix which enumerates the penalty or reward we assign for each combination of true/false prediction with true/false reality. In more concrete terms, we can have either a true positive (this student actually drops out in the next time period and we predict this correctly), a true negative (this student does not drop out in the next time period and we predict this correctly), a false positive (this student does not drop out in the next time period but we predict that they do), or a false negative (this student actually does drop out in the next time period, but we predict they will not). Each correct outcome has a reward and each incorrect outcome has a penalty. For example, we might assign: true positive reward = 1, true negative reward = 1, false positive penalty = -1, false negative penalty = -5, if we believe not catching a student that will drop out is a major offense. Then, we add up the total score (rewards and penalties) for our model and divide by the number of items to get our average score for this model

We can also use the known measure of precision and recall to gauge the quality of our models.

Precision is the total number of times we correctly predicted a student would drop out divided by the total number of times we predicted a student would drop out. This is a measure of how good we are at making correct predictions out of those we predict will drop out.

Recall is the total number of times we correctly predicted a student would drop out divided by the total number of students who actually drop out. This is a measure of how many of the eventual dropouts we are catching.

Note at if $k = 100$, precision is simply the base rate of dropouts in our dataset and recall is simply 1, since we catch everyone who will drop out by virtue of always saying someone will drop out.

These three measures, weighted score, precision, and recall are the measure we use for now in evaluating our models but other measure may be added in the future.

After picking the best model(s) under our three metrics, we can find commonalities between these models such as type of classifier, commonality in parameters, certain features used, etc, and seek to improve upon these areas even further.

2.7. **Future Steps.**

2.7.1. *Error Analysis.* One missing piece of the pipeline is error analysis. That is, we have measures of how accurate we are, even weighted by the type of correct/wrong prediction, we we currently have nothing built in regarding characteristics of objects we are predicting wrong. For example, for each model questions to ask might include

- Are we predicting equally incorrectly across gender and ethnicity?
- Are we predicting equally incorrectly across major groups?
- Are we predicting equally incorrectly across GPA groups?
- etc.

In essence, we would like to know where we are making most of our errors and then why we are making many errors there and not in other places.

2.7.2. *Add SAT/AP/High School Features.* We are currently waiting on the math department to provide some additional data on SAT scores, AP exam scores, and High Schools that students attended to better predict their dropout rates.

2.7.3. *Code Review.* We would like to comment the code better, create better documentation, and speed up some computationally slow features if possible.

3. Similarity Metric Clustering

... Jessica's Notes ...