Student intervention system

Saif Abid

1. **Classification vs Regression**

   This is a classification task. The output is discrete → in our case specifically, because the domain of the output is binary (student either *passed* or *not passed*), this is a binary classification problem.

2. **Explore the Data**

   a. Total number of students: 395
   b. Number of students who passed: 265
   c. Number of students who failed: 130
   d. Number of features: 30
   e. Graduation rate: 67.09% students graduated

3. **Preparing the data**
   *** This section's code snippets and execution can be found in the ipython notebook**

4. **Training and Evaluating models**

   a. Model 1: *Support vector machines*

      The first model I chose was a support vector machine. Their general use is for data sets with high number of features, and data sets which may not be linear in nature. Specific examples *may* include diseases classification problems, and also text classification problems - both of which demonstrate the potential nature of high number of features. The strengths of SVMs, as mentioned above are that they work well in high feature spaces and also in domains which relationships between features is nonlinear (thanks to the notion of kernels). Two major disadvantages of SVM are that firstly, on very large data sets, they may run slow → as Sebastian mentioned in his lecture for SVM, the train time can be $O(n^3)$. Secondly, because SVMs are after linear separating boundaries, if data is **very** noisy, they will not work very well.

      Knowing this information, and having explored the data (step 2), I decided an SVM would be a good fit mainly due to the number of features and size of the data set. The first thing I noticed was there

were many features in this dataset. The number of features however did not exceed the number of samples and as a result SVM seemed like a good choice. Secondly, knowing performance was a metric we are optimizing for, having a medium sized set of examples (~300 samples in the grand scheme of data seems to be small-medium sized), made SVMs make me feel as though their performance would not be very slow.

Model: Support Vector Machine	Training set size		
	100	200	300
Training time (sec)	0.003	0.006	0.010
Prediction time (sec)	0.001	0.001	0.003
F1 score for training set	0.9240	0.9050	0.9118
F1 score for test set	0.8734	0.9135	0.9125

Table A: Performance metrics of SVM

b. Model 2: *Decision Trees*

The second model I chose was a decision trees. Decision trees are very flexible in their uses. General applications of decision trees are on datasets where the data type of the features is very diverse (categorical, continuous, ranking), and also when learning rules need to be easily interpreted and visualized. The main advantages of decision trees, as mentioned above are that they are able to handle diverse feature data types without any normalization (since it buckets data based on the feature value and not actually what the value means relative to any other value the feature can take upon) and also can be visualized very easily. Lastly, one important advantage of decision trees is that they have a natural tendency to pick out the most *important* features early into their training phase (due to information gain). One of the biggest disadvantages to decision trees, is that by default, they are very easy to overfit to the training data. Secondly, they have very high variance - little changes to the training data can completely change the end resulting tree.

The main reason I decided to use decision trees is because after exploring the student data set, it's clear that the data types are very

diverse for the features. For example, Gender is a categorical (male/female) whereas the values for the freetime feature is more “ranking”-like in nature (having a level 2 implies you have less free time then if you had a level 3 of free time).

Model: Decision Trees	Training set size		
	100	200	300
Training time (sec)	0.002	0.003	0.004
Prediction time (sec)	0.000	0.000	0.001
F1 score for training set	1.0	1.0	1.0
F1 score for test set	0.8456	0.8951	0.8965

Table B: Performance metrics of Decision Trees

c. Model C: Bagging

The third model I chose was a ensemble method - specifically a bagging classifier. The bagging classifier is built from decision trees as its base classifier. Bagging seems to have typical applications where there may be some noise present in the data. This is because one of the advantages of bagging is just that - it’s robust to noise due to the fact that it averages (in bagging classifiers, this means it takes a majority vote) of the the classifiers it’s trained on. Also, another advantage of bagging classifiers is that they are less likely to overfit the data since it’s actually multiple classifiers learning subsets of the data. One disadvantage of bagging classifiers is that they tend to be slow learners. *How slow*, actually depends on a few factors, such as the number of classifiers it’s going to train on, and also the relative speed of training the base classifier itself - however in general bagging can be slower than other non-ensemble methods of classification. Another disadvantage of bagging classifiers is that they won’t give you much benefit if you train on a really small set of training data. This is because, bagging classifiers can train on subsets of the training data and if the entire set itself is very small, then each subset may be even smaller and training on very little data may not be enough for the bagging classifier to fully understand the relationships between different features and targets.

After exploring the student data set, I decided to use a bagging classifier for a few reasons. Firstly, because bagging classifiers are more robust to noise, it made sense to use them for this dataset since noise in datasets is always good to account for. Secondly, bagging classifiers as mentioned, are less likely to overfit because they are training multiple classification models.

Model: Bagging classifier	Training set size		
	100	200	300
Training time (sec)	0.027	0.036	0.046
Prediction time (sec)	0.004	0.003	0.004
F1 score for training set	0.9931	0.9895	0.9952
F1 score for test set	0.8571	0.9333	0.9262

Table C: Performance metrics of Bagging classifier

5. Choosing the best model

Looking at the experimental data, along with the advantages/disadvantage of the models I've chosen and lastly taking into account performance, time efficiency, resources and cost - I've decided to build the student intervention classifier using a bagging classifier with a base estimator of a decision tree. The bagging classifier has the best f1 score of the 3 models, that being 0.92 when trained on the full training set, and 0.933 when it was trained on 200 training samples. In terms of time efficiency, there are two metrics to consider here → train time and also prediction time. In terms of training time, bagging classifiers are significantly slower than SVMs and also single decision trees. However, when you look at the prediction time, their prediction time is very comparable to the other models. For example, in one test, I found prediction time for Bagging classifiers to be ~4ms while prediction time for svm was 1-3ms and decision trees were ~0-1ms. Comparably, it's true, that bagging classifiers are still slower than the rest - but overall these are all very fast prediction times. So, after taking both train and prediction times into consideration, the reason I still chose to go with a bagging classifier was that, given the problem statement is to design an intervention software - a slower train time yet a fast prediction time is feasible.

Furthermore, with regards to cost and limited resources, the bagging classifier still seems like a very feasible option. With regards to resources, the bagging

classifier → because it’s training multiple classifiers, can be slightly more bloated in terms of resource usage. However, that being said, with bagging classifiers there's a *n_jobs* parameter which can be used to control the level of parallelism to be used during prediction/training. How does this efficiency compare with SVM and regular decision trees? Well, SVMs will offer more memory efficiency because not all data points are needed to find the optimal decision boundary (only the *support vectors* kept around). Single decision trees have the possibility of becoming more memory intensive than SVMs if they grow with too much depth. However, given that certain parameters such as *max_depth* can be controlled, they too can be optimized to fit in limited resource systems (as with any parameter tuning though, there will be tradeoffs). Ultimately, I believe that because bagging classifiers are built off of decision trees, and also can be optimized to run with fewer resources (*n_jobs)*, they are a feasible option for both costs and resources (since using fewer resources can be directly correlated to lower costs).

All this being said, Bagging classifiers were chosen as being a optimal classifier for the problem at hand.

I would like to turn the focus of this proposal to give a brief summary as to how exactly the bagging classifier I have chosen works.

To start, we have information about 395 students. This information ranges from various aspects of the students life, for example *How much free time do they have*, *What is their gender* etc. We also know wheater or not each one of those 395 students *passed, or failed*. At this point, what we would like to do is determine all paths which lead to a student failing and all paths which lead to a student passing. Once we know this, then we can take a new students information such as their amount of free time, their gender etc, and see which path they fall on to predict if they will pass or not.

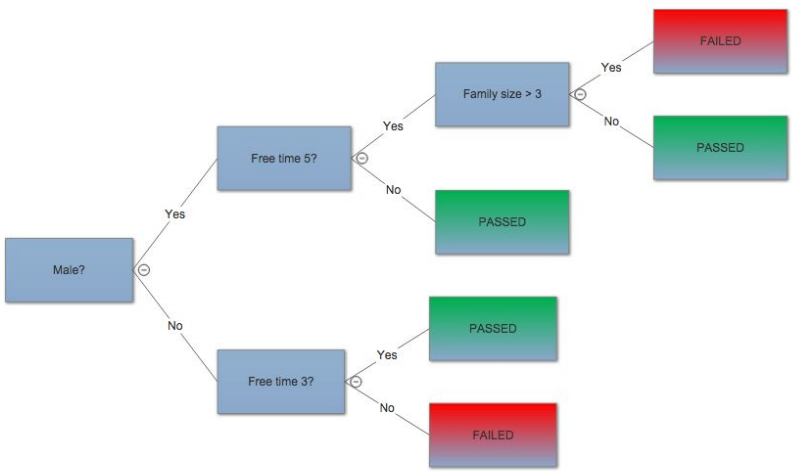
A path can be thought of as a series of yes/no questions. For example let's say we found that all students who had greater than 3 family members, were male and had a freetime rating of 4, ended up *failing*. Then, When a new student comes in, we can ask “Members of family greater than 3?” if yes , “Level of free time 4?” if yes , “Gender male” if yes , then this new student will possibly fail as well. As you can see, looking at the data, many of these “if-then” questions can be answered to create a path in which students fail or pass.

Now, if we just figure out all the paths for these 395 students and then use them to predict a new student's ability to fail or not, it might still not be very

accurate. This is because, there is no guarantee, that 395 students are a good representation of the whole student population but more importantly, since these questions were answered by students (or humans to be even more general), the chances of errors and false information is high. For example, let's say everybody who has 5 as their free time fails, regardless of the number of members in your family. However, let's say there's a rare case, single student who passed and has a free time of 5. Now, for the path to be constructed, it would no longer be enough to just “if the free time 5?”, you would also have to ask additional questions to predict if new students will fail or not, and the more specific you get with the questions the less likely you're representing the entire student population and more likely representing only the 395 students. This said, in reality - the rare student doesn’t represent students in general, however because of his data being used to construct the path, the entire path is becomes very specific and it becomes harder to understand the likelihood of a general new student passing or failing.

So, to account for niche cases, or for the fact that all data cannot be trusted, what we can do is first segment the initial 395 students into smaller random groups. Let's say you split up the 395 students into 5 groups.

Now you can construct if-then paths on each group. So each group would end up with paths which look different. A single group's paths may look like this (only an example):



What you will find is since you separated your 395 students into 5 groups, only 1 or two (let's assume 1 in our running example) of the groups will have

gotten the student who passed even though they had free time 5. Now when a new students data comes in, you can go down the paths of each group and see if the student fails or not majority of the time and report that as the final prediction. So, if a student comes in with a free time of 5 and is a male, 4 of the 5 groups will have said yes the student should fail, and 1 of the 5 groups (the group with the anomaly student) may report otherwise. Regardless of what this anomaly group gives as an output, it will be overlooked since the *majority* of the groups said the student should fail

Ultimately you can see that by making these simple if-then paths, and making multiple groups of them, you can get rid of the niche/super rare cases and predict based on a more general model - since in reality this this is exactly what you want, a system which models the general student, not one which models specific students and no one else.