Project Description

As education has grown to rely more and more on technology, more and more data is available for examination and prediction. Logs of student activities, grades, interactions with teachers and fellow students, and more are now captured through learning management systems like Canvas and Edmodo and available in real time. This is especially true for online classrooms, which are becoming more and more popular even at the middle and high school levels.

Within all levels of education, there exists a push to help increase the likelihood of student success without watering down the education or engaging in behaviours that raise the likelihood of passing metrics without improving the actual underlying learning. Graduation rates are often the criteria of choice for this, and educators and administrators are after new ways to predict success and failure early enough to stage effective interventions, as well as to identify the effectiveness of different interventions.

Toward that end, your goal as a software engineer hired by the local school district is to model the factors that predict how likely a student is to pass their high school final exam. The school district has a goal to reach a 95% graduation rate by the end of the decade by identifying students who need intervention before they drop out of school. You being a clever engineer decide to implement a student intervention system using concepts you learned from supervised machine learning. Instead of buying expensive servers or implementing new data models from the ground up, you reach out to a 3rd party company who can provide you the necessary software libraries and servers to run your software.

However, with limited resources and budgets, the board of supervisors wants you to find the most effective model with the least amount of computation costs (you pay the company by the memory and CPU time you use on their servers). In order to build the intervention software, you first will need to analyse the dataset on students' performance. Your goal is to choose and develop a model that will predict the likelihood that a given student will pass, thus helping diagnose whether or not an intervention is necessary. Your model must be developed based on a subset of the data that we provide to you, and it will be tested against a subset of the data that is kept hidden from the learning algorithm, in order to test the model's effectiveness on data outside the training set.

To see detail regarding data exploration and data preparation, please check the attached Jupyter Notebook. Here is a small summary of the data:

Total number of students	395
Number of students who passed	265
Number of students who failed	130
Graduation rate of the class (%)	31
Number of features	67%

Model Evaluation:

The table below shows information about 8 different models tested where you can see the F1 score achieved on the training set and on the test set, using different train sizes (100,200 and 300 observations). The test size is always constant.

So, for example, on the SVC model, 0.8354 on the "Train F1 Score" represents the score when the model was trained with 100 observations and tested on the same 100 observations and the 0.8025 represents the score when the model was trained with 100 observations and tested on the full test set. On the last column, 0.8052, we have the score when the model was trained with 300 observations and tested on the full test set.

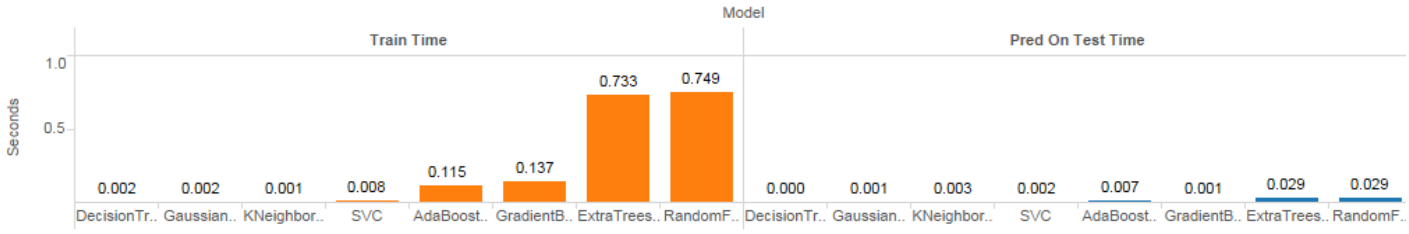
FYI, the x axis was set on a logarithmic scale from 0.6 to 1.3 to make visualization easier and the results are sorted by the higher test f1 score.

We can see a few very interesting behaviours on this table:

- Ensemble models (Random Forests, Extra Trees and Gradient Boosting – and even decision trees) overfit very quickly when tested on the same dataset they were trained on);
    - We see the same behaviour on AdaBoost but as we add more data to the training phase, the overfit seems to go down;
- On Gaussian Naïve Bayes the test Score increases significantly as we add more data to the raining set;
- Support Vector Machines doesn't seem to benefit from more training data

Training\Testing Time:

As training and\or testing will increase linearly based on the amount of data, I'm reporting the full training set on the table below:



As we can see ensemble methods are the ones who take longer to train (especially due to the high number of estimators) and test.

Instance Based models (lazy learners) like KNN are usually faster to train and slower to evaluate (and occupy more space), but given the small size of the data set, we can't observe that behaviour.

Selecting 3 best models

	Model	TrainingSize	TrainTime	PredOnTrainTime	PredOnTestTime	TrainingF1Score	TestingF1Score
2	KNeighborsClassifier	300	0.002	0.008	0.002	0.853933	0.813793
1	KNeighborsClassifier	200	0.002	0.003	0.002	0.809689	0.785714
0	KNeighborsClassifier	100	0.001	0.002	0.001	0.825175	0.758621

	Model	TrainingSize	TrainTime	PredOnTrainTime	PredOnTestTime	TrainingF1Score	TestingF1Score
4	SVC	200	0.004	0.003	0.001	0.843137	0.810458
5	SVC	300	0.008	0.005	0.002	0.866379	0.805195
3	SVC	100	0.002	0.001	0.001	0.835443	0.802548

	Model	TrainingSize	TrainTime	PredOnTrainTime	PredOnTestTime	TrainingF1Score	TestingF1Score
8	RandomForestClassifier	300	0.735	0.038	0.030	1	0.797297
7	RandomForestClassifier	200	0.715	0.032	0.029	1	0.789116
6	RandomForestClassifier	100	0.680	0.027	0.027	1	0.783784

Choosing the Best Model

Given the data, the best model would be a KNeighborsClassifier using a large training set because it has the highest evaluation score and one of the smallest training and testing times.

If supporting the large training size is not an option, an alternative would be to use a Support Vector Machines model with the small training set, which takes the same amount of time to train* and test and whose score suffers very little from the less training data. * Train an SVC on 100 observations takes the same amount of time than training a KNN on 300 observations

Fine-tune the model using the Gridsearch didn't produce a meaningful result. I was able to overfit the model (see training score of 1) but the Test score remained exactly the same (which indicates the default parameters are good):

	Model	TrainingSize	TrainTime	PredOnTrainTime	PredOnTestTime	TrainingF1Score	TestingF1Score
0	KNeighborsClassifier	300	0.001	0.008	0.003	1	0.813793