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.

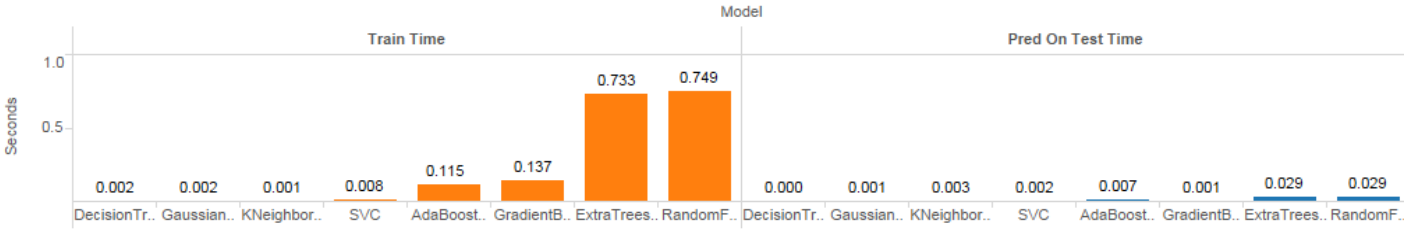
FYI, the x axis was set on a logarithmic scale from 0.6 to 1.3 to make visualization easier and the results are sorted by the higher test f1 score.

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

See the last section “Top 3 Algorithms Selection” for more details and explanation about the algorithms selected. Below is a table with their times and scores:

	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.

A KNeighborsClassifier works by keeping a “database” of the known values and its labels and when we request it to predict a new value, it compares the new value with the values on the database and infers its label based on the labels of its closest K neighbours.

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 ended up being less than the one I had with the default parameters.

	TrainingF1Score	TestingF1Score
0	1	0.810811

Top 3 Algorithms Selection

The main reason to choose the algorithms bellow was because they were the best performers from the list and also they'd be the best option to this problem according to the scikit-learn's Machine Learning Map

KNeighborsClassifier

K-NN is a 'Lazy Learner' classification algorithm. A lazy learner does not build any model beforehand; it waits for the unclassified data and then winds its way through the algorithm to make classification prediction. Lazy learners are, therefore, time consuming–each time a prediction is to be made as all the model building effort has to be performed again.

In k-nearest neighbour algorithm, the training data is first plotted on an n-dimensional space where 'n' is the number of data-attributes. Each point in 'n'-dimensional space is labelled with its class value. To discover classification of an unclassified data, the point is plotted on this n-dimensional space and this data point's distance from all other points is to be calculated and then only nearest k-points can be discovered for voting.

The number of neighbours that will vote for the class of the target point is set in the n_neighbors parameter and the weight of each vote (defined by the weights parameter) has two choices to be computed: 'uniform' and 'distance'. For the 'uniform' weight, each of the k neighbours has equal vote whatever its distance from the target point. If the weight is 'distance' then voting weightage or importance varies by inverse of distance; those points who are nearest to the target point have greater influence than those who are farther away.

The lazy learning can be seen both as a strength (fast training time) and as a weakness (slow prediction time). A clear disadvantage of lazy learners is that since they need to store all the instances, they take up a lot of memory to run. On the other hand they are known to work well for a small number of dimensions.

Support Vector Machines

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labelled training data, the algorithm outputs an optimal hyperplane which categorizes new examples. Since SVMs don't have a standardized way for dealing with multi-class problems it is fundamentally a binary classifier.

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, the method is likely to give poor performances.

- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

Random Forest Classifier

Random forests is a notion of the general technique of random decision forests that are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms.

Random Forest is an "evolution" from Decision trees (which are known for their habit of overfitting to their training set) by applying the general technique of bootstrap aggregating, or bagging, to tree learners where it repeatedly selects a random sample with replacement of the training set and fits trees to these samples.

Random forests add another step to the bagging approach: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called "feature bagging". The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the B trees, causing them to become correlated

Typically, for a classification problem with p features, √p (rounded down) features are used in each split. For regression problems the inventors recommend p/3(rounded down) with a minimum node size of 5 as the default.

Advantages:

- Does not expect linear features or even features that interact linearly
- Handles very well high dimensional spaces as well as large number of training examples.
- It produces a highly accurate classifier and learning is fast.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what variables are important in the classification.

Disadvantages:

- A large number of trees may make the algorithm slow for real-time prediction.

References:

- https://ashokharnal.wordpress.com/2015/01/21/k-nearest-neighbor-classification-using-python/
- http://www.improvedoutcomes.com/docs/WebSiteDocs/Clustering/Clustering_Parameters/Distance_Metrics_Overview.htm
- http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html
- https://www.quora.com/What-are-the-advantages-of-different-classification-algorithms
- http://jmlr.org/papers/volume15/delgado14a/delgado14a.pdf
- https://en.wikipedia.org/wiki/Random_forest
- http://ect.bell-labs.com/who/tkh/publications/papers/compare.pdf