Build a Student Intervention System

§1. Classification vs Regression

In this project we wish to identify students who are at risk of failing and might require early intervention. This is a binary classification problem, that is, either it is predicted that a student will fail, and thus requires intervention, or not.

§2. Exploring the Data

We are provided with data for 395 students, of those 265 students passed and the remaining 130 students failed giving a graduation rate of 67.09%. The dataset contains 30 features for each student, in addition to whether they passed or failed. These features are described in more detail in Appendix section A1.

We make the following observations about our dataset:

- Our output data (label) is binary (either the student passed or failed)
- Our input data (features) are a mixture of binary, categorical (non-numeric and ordinal) and numerical. The categorical data has up to five classes
- The graduation rate is 67% (not 50%) - the data is 'imbalanced'
- The dataset is very small:
    - There are approximately 13 samples per feature
    - If we compare the number of samples per feature conditioned on the output, the samples per feature drops as low as 4 (in the case where the student fails)

§3. Preparing the Data

§3.1 Choosing the positive labelled output

As part of the data pre-processing we convert our non-numeric output to binary. It is important that the positive label in our output represents the outcomes we are searching for. This is because we are using F_1 score as the performance metric for our classifier and, of the correctly classified data, this only explicitly takes into account true positive classifications (not true negative). In our case we are interested in identifying which students require intervention to prevent failure, so rather than having our output as whether the student passed or not we change it to whether intervention was required or not. With this in mind our output is considered positive if the student failed and negative if they passed.

§3.2 Removing redundant data columns

As part of pre-processing our data, we convert non-numeric data columns to numeric. As an example take the data column 'sex' which contains one of two strings, either 'M' or 'F'. This is converted to two mutually exclusive binary columns 'sex_M' and 'sex_F' one of which contains 1 and the other a 0.

Pre-processing our data in this way takes our feature set size, which is already large given the small sample size, from 30 to 48. The method used in the conversion turns each categorical data feature into n mutually exclusive binary features, where n is the number of possible classes in that feature. We note that the sum of the resulting n binary features is always 1 for any given example, i.e. there is linear dependence between the n features. Consequently, any one of the n features can be removed without loss of data. In this way we are able to reduce the number of features from 48 to 39.

For each non-numeric feature which was converted to numeric, we must choose which of the resulting binary columns to remove. Ideally, we would like to remove the column which helps us the least in predicting the output.

We note that calculating the average of any binary feature column gives the proportion of examples with that feature or alternatively, the probability that an example exhibits that feature (assuming our sample to be representative of the population). For example, the average of the column 'sex_F' tells us the proportion of the students which are female or alternatively the probability that a student is female.

To decide which of the columns to remove we separate our data into students who passed and those who failed. We then calculate the proportion of students with that feature in each case (the probability a student has that feature given that they passed/failed) and finally, compute the absolute value of the difference. Features where the difference is larger are more discriminating.

For features which originally had two possible classes, (for example, the column named 'sex' which contained one of two strings, either 'M' or 'F') both of the resulting binary features are equally discriminating so we can remove either. For features which originally had more than two possible classes, we remove the least discriminating binary feature.

§3.3 Splitting data into training and test sets

As noted in section 2, we have an imbalanced dataset - approximately two thirds of the students in the data pass and the rest fail. To reduce statistical noise in our performance metrics we use `StratifiedShuffleSplit` (instead of just `train_test_split`) to separate our data into training and test sets. This ensures the output distribution is consistent across training and test sets.

§4. Training and Evaluating Models

§4.0 Choosing models

We choose three models from scikit-learn which are suitable for classification problems where the training sample size is 'small' (<100K is the ball park given by sci-kit learn[1], clearly our sample size is much smaller than this). Other models (stochastic gradient descent for example) require a larger dataset to train on. Our three models also all take very different approaches to solving the same problem.

We note that for this problem, both prediction performance and computational efficiency are important – we want to "find the most effective model with the least amount of computation costs". There are trade offs in the computational cost of training and querying when comparing instance based and parametric learning methods. Parametric models have an explicit training step where the parameters are calibrated to the data. Instance based models only store the data and have no explicit training step. This means that parametric model training times increase with the training set size, whereas for instance based models, training time remains approximately constant. In making predictions, parametric models only evaluate a formula whereas instance based models must iterate through the entire dataset to find the neighbouring points. Consequently, for parametric models, prediction time remains approximately constant as the size of the training data increases, however instance based models typically use more memory in making predictions and prediction time increases like log n (where n represents the number of data points).

For this reason, we include one of each model type, a Support Vector Machine and K Nearest Neighbours. A Support Vector Machine is a parametric model and K Nearest Neighbours is an instance based model.

[1] http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

These models provide us with a feel for what we can expect, computation time wise, at the upper and lower ends for training and querying.

§4.1 Baseline performance metrics

As discussed in section 3, we are interested in identifying students who will likely fail and thus require intervention. The primary metric we use to compare the performance of different learning algorithms is the F_1 score. For a given accuracy score, we prefer false positive over false negative errors, i.e. for a fixed accuracy, we would prefer a higher recall than precision.

In order to compute some baselines for our performance metrics, let the pass rate for students be p. Consider a model that makes predictions randomly under a distribution where q of the time we predict a student passes and $1-q$ of the time we predict they fail. The confusion matrix for such a model is given in Table 1.

		Actual Class	
		1	0
Predicted Class	1	$(1-q)(1-p)$ True Positives	$(1-q)p$ False Positives
	0	$q(1-p)$ False Negatives	qp True Negatives

Table 1: Confusion matrix for random guessing strategy which predicts pass q of the time where the pass rate is p

For our problem we have that p = 0.6709. We note in the case $q=0$, when we always predict the student fails and intervene, the accuracy is 0.3291 and the F_1 score is 0.5. Using this method, we correctly identify all the students which require intervention. In the case $q=1$, when we always predict they pass and intervention is never required, the F_1 score is zero[2] even though accuracy is 0.6709 and we make more than twice as many correct predictions. Indeed, for our purposes, there's little point in only correctly identifying those students who don't require intervention. Finally, we note that precision = recall = 0.3291 (we have false positive and false negative errors in equal amounts) when $q=p$.

Performance metric baselines are summarised in Table 2. We note the best F_1 score we are able to achieve with our guessing strategy is 0.5 but the accuracy is only 0.3291, the best accuracy we are able to achieve is 0.6709 but in this case the F_1 score drops to zero.

		$p = 265/395 = 0.6709$		
		$q=0$	$q=1$	$q=p$
Accuracy	$1-p-q+2pq$	0.3291	0.6709	0.5
Precision	$1-p$	0.3291	0.3291	0.3291
Recall	$1-q$	1	0	0.3291
F_1 Score	$2(1-p)(1-q)/(2-p-q)$	0.5	0	0.3291

Table 2: Performance metric baselines

[2] If we always predict pass, we have zero positive predictions, so strictly speaking precision, and hence F_1 score, are undefined, however we can instead consider the limiting case where we *almost* always predict pass e.g. q = 0.999. In this case precision (= $1-p$) is independent of q and the larger the value of q the closer to zero the F_1 score.

§4.2 Support Vector Machine

§4.2.1 General applications, strengths and weaknesses

Support Vector Machines or SVMs are a set of parametric supervised learning methods for both classification and regression. In the case of the former, SVMs are binary linear classifiers. A crucial part of the SVM output is a surface. In classification this surface is the decision boundary which divides the classes. In regression the surface itself is used to predict the output for unseen inputs.

+ SVMs can be extended to solve non-linear problems. A kernel trick can be used to map inputs which are not linearly separable to higher dimensions to make them so. The kernel function is restricted only to those which are positive semi definite so the method is versatile.
+ SVMs can be effective in high dimensional spaces even those where the 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 memory efficient.
+ SVM avoids over-fitting by construction – the model maximizes the margin between the different classes.

- Multiclass classification problems must be solved by applying the classifier repeatedly for each output class, one versus the rest, so the computational cost increases linearly with the number of classes.
- Does not perform well for very large datasets or datasets with a lot of noise
- If the number of features is much greater than the number of samples, the method is likely to give poor performance.
- SVMs do not directly provide probability estimates.

§4.2.2 Justification of model choice

Our problem is one of binary classification so well suited to SVM which is also a relatively robust model so serves as a good first port of call. It is computationally expensive to train but cheap for making predictions when compared with an instance based classifier such as K nearest neighbours. Since both computational efficiency, memory and prediction performance are all important in our problem SVM provides us with an understanding of the the lower end of computational cost for queries (and higher end for training).

§4.2.3 Results

Tables 3 contains the results using a Linear Support Vector Machine, straight ‘out of the box’. Accuracy is over 71% on our test sets for all three training sizes and F_1 score is between 43% and 53%. We notice that the performance metrics don’t show steady increase or decrease with training set size which indicates the presence od noise. Precision tends to be greater than recall for our trained models. As initially discussed, training is computationally expensive compared to prediction for an SVM.

SVM		Training set size		
		100	200	300
Time (seconds)	Training	0.016082	0.016467	0.046576
	Prediction	0.000538	0.000678	0.001033
F_1 score	Training set	0.735294	0.654206	0.533333
	Test set	0.526316	0.677419	0.434783
Precision	Training set	0.833333	0.76087	0.784314
	Test set	0.576923	0.677419	0.666667
Recall	Training set	0.657895	0.57377	0.40404
	Test set	0.483871	0.677419	0.322581
Accuracy	Training set	0.82	0.815	0.766667
	Test set	0.715789	0.789474	0.726316

Table 3: Support Vector Machine results - SVC (kernel=Linear)

§4.3 K Nearest Neighbours

§4.3.1 General applications, strengths and weaknesses

K Nearest Neighbours (KNN) is a non-parametric or instance based learning algorithm used in both classification and regression problems. In addition, KNN also forms the foundation for some unsupervised learning methods such as clustering. As the name suggests, the k nearest neighbouring points in the input data are used to predict the output. The distance can be any metric function. In classification, the output is determined by majority vote, while in regression, the output is the mean. In both cases the contribution from the neighbours may be weighted (for example, by the distance from the input).

+ Simple algorithm
+ Useful in problems where a functional relationship between inputs and outputs is not clear
+ Successful in classification problems where the decision boundary is very irregular
+ Learning is computationally cheap compared to parametric methods such as SVM since all we do is store the data. For cases where training data is being gathered/updated continuously this can be advantageous.

- Queries are computationally expensive compared to parametric methods, all the data must be stored and we iterate through it to make predictions
- For high dimension problems the method is ineffective as we end up finding that all points are similarly distant.

§4.3.2 Justification of model choice

KNN is computationally cheap to train but expensive for making predictions when compared with parametric classifiers like SVM. Since both computational efficiency, memory and prediction performance are all important in our problem, KNN provides us with an understanding of the the lower end of computational cost for training (and upper end for queries).

§4.3.3 Results

Tables 4 show the results for K Nearest Neighbours for K = 7 respectively used straight 'out of the box'. We note that the F_1 score consistently increases with training set size as does accuracy. Changes in Accuracy and F_1 score are monotonic. F_1 score and accuracy are 74% and 51% respectively for the largest training set. Precision is generally larger than recall. As initially discussed, prediction is computationally expensive compared to training for KNN.

7NN		Training set size		
		100	200	300
Time (seconds)	Training	0.000581	0.000667	0.001282
	Prediction	0.00122	0.001715	0.003025
F_1 score	Training set	0.508475	0.54	0.558442
	Test set	0.415094	0.470588	0.509804
Precision	Training set	0.714286	0.692308	0.781818
	Test set	0.5	0.6	0.65
Recall	Training set	0.394737	0.442623	0.434343
	Test set	0.354839	0.387097	0.419355
Accuracy	Training set	0.71	0.77	0.773333
	Test set	0.673684	0.715789	0.736842

Table 4: K Nearest Neighbours results – K = 7

§4.4 Bernoulli Naive Bayes

§4.4.1 General applications, strengths and weaknesses

Naive Bayes is a supervised classifier. The method takes a probabilistic approach based on Bayes Theorem with the 'naive' assumption of independence between the input features (though the method has been used successfully in cases where the features are not independent, the most well known of which is spam filtering). Different Naive Bayes classifiers differ in the assumption they make about the distribution of the input features conditioned on the output. Bernoulli Naive Bayes assumes the input features to be binary.

+ Simple model which takes a probabilistic approach and does not have many tuning parameters
+ Easy to understand
+ Fast compared to more sophisticated classifiers
+ Effective even where there exists linear dependence between features
+ Provides probability estimates

- Probability estimates are not accurate
- Generic implementations require all input data to be of the same type

§4.4.2 Justification of model choice

We found that using `StratifiedShuffleSplit` (instead of just `train_test_split`) reduced the noise in our results significantly and gave more consistent improvement with training size. The fact that we are able to reduce the noise through stratification implies that it is distributed proportionately among the output classes. This, among other things, leads us to our final model choice.

After pre-processing our data, we have 39 input features, 26 of which are binary (like our output). If all of our data were binary, Bernoulli Naive Bayes would be an obvious choice for a classifier. In this case, our problem would be much like the problem of spam filtering in emails for which we know the method to be both computationally efficient and successful in classifying. If all our features are binary, we would have a simple way of comparing how effective each one is in helping us predict the output and would be able to determine which features are worth keeping and which ones introduce noise and additional computation and could be removed.

Of the 13 non-binary features, 11 are categorical ordinal with 4/5 classes and the remaining two, age (15-22) and absences (0-93) are numerical with integer values. With this in mind we consider how we might go about converting the remaining 13 features to binary.

In section 3.2 we describe a method for measuring how discriminating a given binary feature is in the output. Recall, we separate our data into students who passed and those who failed. We then calculate the proportion of students with that feature in each case (the probability a student has that feature given that they passed/failed) and finally we compute the absolute value of the difference. Features where the difference is larger are more discriminating.

A simple way of binarising our numerical features is to choose a threshold and transform all the data points on one side of the threshold to 1 and the rest to 0. Ideally we would like to choose the threshold which maximises the predictive power of the resulting binary feature. On inspecting the distributions of our non-binary data for student who pass versus those who fail (see Appendix section A2), we find that there is an obvious choice of a threshold for binarising the data for almost all of them. Take for example the distributions for the data feature ‘Medu’ for students who passed and those who failed (shown in Figure 1). The frequency of the values 0-3 (secondary education and lower) are all higher for students who failed. Above this, value 4 (higher education) the frequency is higher for students who pass. Then the obvious choice for the threshold in this case is 3.5. This threshold gives us the largest difference between probabilities of the new binary feature, given the output.

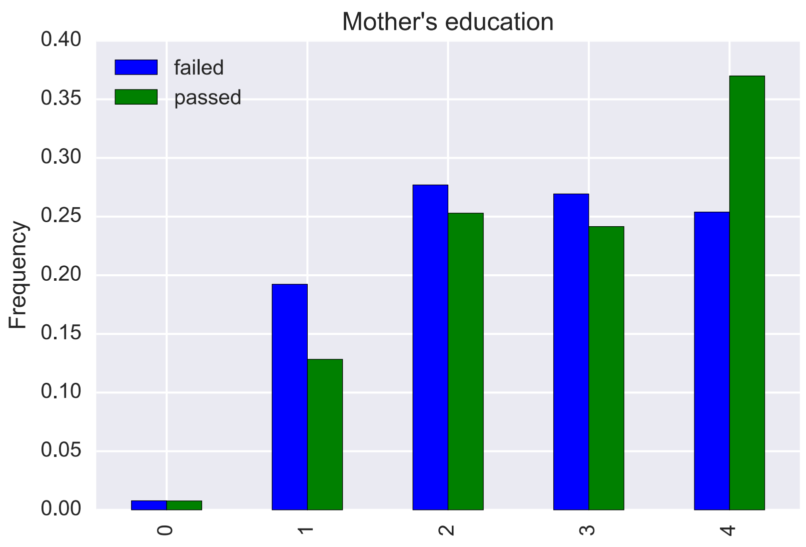

Figure 1: Comparison of distribution of feature ‘Medu’ for students who passed and failed

In general, we choose the threshold for a categorical feature so that, the difference between the frequencies of each class observed for students who pass and students who fail, changes sign across it.

§4.4.3 Results

Tables 5 show the results for Bernoulli Naive Bayes (after binarising the data as above) used straight ‘out of the box’. Accuracy is over 69% on our test sets for 300 training points and F_1 score is 47%. We notice that the performance metrics don’t show steady increase or decrease with training set size which again indicates the presence od noise. Precision tends to be greater than recall for our trained models. We note that prediction is very fast and training isn’t far behind.

BNB		Training set size		
		100	200	300
Time (seconds)	Training	0.002119	0.001749	0.002509
	Prediction	0.000475	0.000357	0.000512
F_1 score	Training set	0.608696	0.558559	0.552941
	Test set	0.492063	0.615385	0.472727
Precision	Training set	0.677419	0.62	0.661972
	Test set	0.326316	0.588235	0.541667
Recall	Training set	0.552632	0.508197	0.474747
	Test set	1	0.645161	0.419355
Accuracy	Training set	0.73	0.755	0.746667
	Test set	0.326316	0.736842	0.694737

Table 5: Bernoulli Naive Bayes results

§5. Choosing the Best Model

§5.1 Computational efficiency

Table 6 compares training and prediction times for our three models. We discussed the relative training and prediction times for SVM and KNN in section 4. We note that BNB has the fastest prediction time of all three algorithms and falls generally in between SVM and KNN for training time (although much closer to the lower end). If all three models performed equally well in making predictions, I would choose BNB as it would scale to larger datasets better for both training and predicting.

		Training set size		
		100	200	300
Training Time	SVM	0.016082	0.016467	0.046576
	7NN	0.000581	0.000667	0.001282
	BNB	0.002119	0.001749	0.002509
Prediction Time	SVM	0.000538	0.000678	0.001033
	7NN	0.00122	0.001715	0.003025
	BNB	0.000475	0.000357	0.000512

Table 6: Training and prediction times for our three models

That said, it's worth noting that almost all the times measured are of the order of one thousandth of a second (training times for SVM which are of the order of one hundredth of a second) and smaller. Indeed, the times are so small, they probably can't even be accurately measured. Looking at the times as a whole, for our dataset size, all three models are so fast, in both training and predicting, that it is not worth taking this into account in choosing a model. The wasted cost and effort involved in intervening, if a student is incorrectly predicted to fail, is likely to far outweigh that involved in training any one of these models and then making predictions based on it, furthermore the training data would have to increase in size by many orders of magnitude to even come close to being comparable. For these reasons, I would argue that it is more important to consider the prediction performance for the model than the computational expense.

§5.2 Predictive Performance

Figure 2 summarises the F_1 scores for all three models. Comparing the F_1 scores for 300 training points, seems to imply the order of the models in descending order of prediction performance is 7NN followed by BNB and then SVM, this order is reversed for 100 and 200 training points. The results for models SVM and BNB show significant volatility for increasing training set size. We also note that for two of the models, the test set gives a higher F_1 score that the training set for 200 points. This leads us to believe that a simple `StratifiedShuffleSplit` does not give an accurate estimate of the relative performance of our three models. To do this we take advantage of `GridSearchCV` and optimise over some model parameters since it's easy and fast to do so.

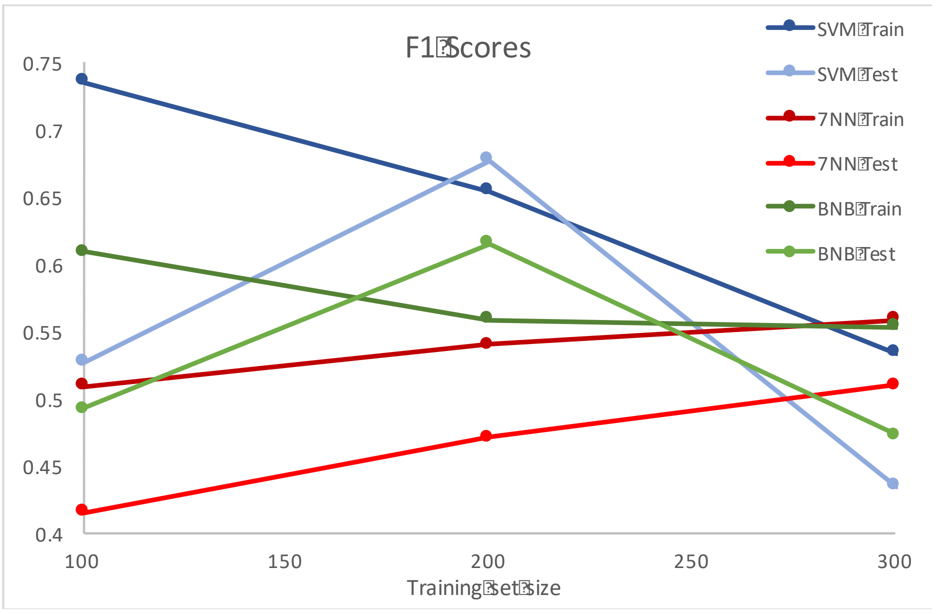

Figure 2: Training and testing F_1 score for varying training set size for our three models

§5.3 Final model choice: Bernoulli Naïve Bayes

Table 7 summarises our results using `GridSearchCV`. Based on these results, the best model (with the highest F_1 Score) is Bernoulli Naive Bayes. As a bonus, this is also the most computationally efficient model of the three.

	SVM	KNN	BNB
Grid Search Parameters	**kernel** (linear, poly, rbf) **C** (5, 10 ,20 ,40, 80)	**n_neighbors** (1,2,…,20)	**alpha** (1,2,…,20), **fit_prior** (True, False)
Best Parameters	**kernel** = linear **C** = 20	**n_neighbors** = 3	**alpha** = **fit_prior** = False
Best F_1 Score	`0.4623`	`0.3816`	`0.4975`

Table 7: `GridSearchCV` results for our three models

§5.4 Model Description: Bernoulli Naïve Bayes

Our chosen model is Bernoulli Naive Bayes. This model takes a probabilistic approach to the problem of determining if students are likely to pass or fail. The basic idea of the algorithm is that if the probability that a student will pass given their particular features is greater than the probability they will fail given those features then we predict the student will pass, otherwise we predict fail.

To calculate the required probabilities, we use Bayes Theorem,

$$P(y \mid \boldsymbol{x}) = \frac{P(y)P(\boldsymbol{x} \mid y)}{P(\boldsymbol{x})}$$

In our case it relates the probability, $P(y \mid \boldsymbol{x})$, of a student passing (or failing), that is $y = 0$ (or 1), given their features, $\boldsymbol{x} = (x_1 \cap x_2 \cap \ldots \cap x_n)$, to the probability of having those features assuming the student passed (or failed), $P(\boldsymbol{x} \mid y)$. The former probability is essentially the problem we are trying to solve and is intractable, whereas the latter probability is one that we are able to estimate from historical data.

Different Naive Bayes classifiers differ in the assumptions they make regarding the distributions of the features, x_i. For our problem most of the data is binary and the remaining data we convert. The Bernoulli distribution is the generalised distribution for a binary variable.

Naive Bayes methods apply Bayes Theorem with the 'naive' assumption of independence between the features[3]. This independence assumption means we can rewrite Bayes Theorem in a much more computationally efficient way[4],

$$P(y \mid \boldsymbol{x}) = \frac{P(y)\prod_{i=1}^{n} P(x_i \mid y)}{P(\boldsymbol{x})}$$

Since the denominator is independent of the value of y, it multiplies through both probabilities $P(y = 0 \mid \boldsymbol{x})$ and $P(y = 1 \mid \boldsymbol{x})$ equally, we don't need to calculate it. We have the following relationship,

$$P(y \mid \boldsymbol{x}) \propto P(y)\prod_{i=1}^{n} P(x_i \mid y) \qquad (1)$$

In training we estimate the individual probabilities on the right hand side of equation (1) for all y and x_i using our data. The probability of passing, $P(y = 0)$, is approximated by the proportion of the students in the training data which pass. The probability of failing, $P(y = 1)$, is then just one minus the probability of passing. To calculate the probabilities $P(x_i \mid y)$, we do something similar but we first separate the data into students who pass and those who fail. We can then use the data in the respective groups to estimate $P(x_i = 1 \mid y = 0)$ and $P(x_i = 1 \mid y = 1)$ for each feature by again looking at proportions of the students with that feature x_i. To calculate $P(x_i = 0 \mid y = 0)$ and $P(x_i = 0 \mid y = 1)$, we simply calculate one minus the previous corresponding probabilities.

To make a prediction for a given set of features $\boldsymbol{x}$ we evaluate the right hand side of the formula in both the case y = 0 (fail) and y = 1 (pass). Whichever value comes out greater determines our prediction for y.

§5.5 Fine tuning our model

For our chosen model, Bernoulli Naive Bayes, we tune the model parameters 'alpha' and 'fit-prior' using `GridSearchCV` and simultaneously optimise over the k best features for k running from 1 to 39 where the best feature is considered the most discriminating (as described in section 3.2).

§5.6 Final F_1 score

Our final F_1 score is 50%. To obtain this we use the 3 features most discriminating binary features with alpha = 9 and fit-prior = False.

[3] It turns out that even where there is dependence between features, the algorithm has proven effective in classification, the most well known problem being spam email filtering.

[4] Essentially the independence assumption allows us to write the joint probability of multiple events as the product of their marginals.

Appendix

§A1. Data descriptions

	Feature Name	Description	Type	Values
1	**school**	student's school	binary	"GP" or "MS"
2	**sex**	student's sex	binary	"F" or "M"
3	**age**	student's age	numeric	15 to 22
4	**address**	student's home address type	binary	"U" – urban "R" - rural
5	**famsize**	family size	binary	"LE3" - less or equal to 3 "GT3" - greater than 3
6	**Pstatus**	parent's cohabitation status	binary	"T" - living together "A" - apart
7	**Medu**	mother's education	ordinal	0 – none 1 - primary (4th grade) 2 – 5th to 9th grade 3 – secondary education 4 – higher education
8	**Fedu**	father's education		
9	**Mjob**	mother's job	categorical	"teacher" "health" care related civil "services" (e.g. administrative or police) "at_home" "other"
10	**Fjob**	father's job		
11	**reason**	reason to choose this school	categorical	close to "home" school "reputation" "course" preference "other"
12	**guardian**	student's guardian	categorical	"mother" "father" "other"
13	**traveltime**	home to school travel time	ordinal	1 - <15 min 2 - 15 to 30 min 3 - 30 min. to 1 hour 4 - >1 hour
14	**studytime**	weekly study time	ordinal	1 - <2 hours 2 - 2 to 5 hours 3 - 5 to 10 hours 4 - >10 hours
15	**failures**	number of past class failures	ordinal	n if 1<=n<3, else 4
16	**schoolsup**	extra educational support	binary	yes or no
17	**famsup**	family educational support		
18	**paid**	extra paid classes within the course subject (Math or Portuguese)		
19	**activities**	extra-curricular activities		
20	**nursery**	attended nursery school		
21	**higher**	wants to take higher education		
22	**internet**	Internet access at home		
23	**romantic**	with a romantic relationship		
24	**famrel**	quality of family relationships	ordinal	1 - very bad to 5 - excellent
29	**health**	current health status		
25	**freetime**	free time after school	ordinal	1 - very low

26	**goout**	going out with friends		to
27	**Dalc**	workday alcohol consumption		5 - very high
28	**Walc**	weekend alcohol consumption		
30	**absences**	number of school absences	numeric	0 to 93
31	**passed**	did the student pass the final exam	binary	yes or no

§A2. Histograms for Non-Binary Ordinal Data with Binarising Thresholds

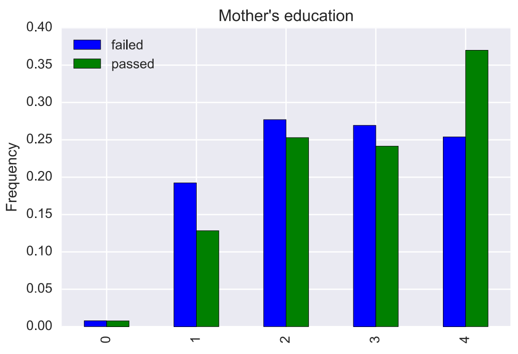

Figure 2: ‘Medu’ binarising threshold = 3.5

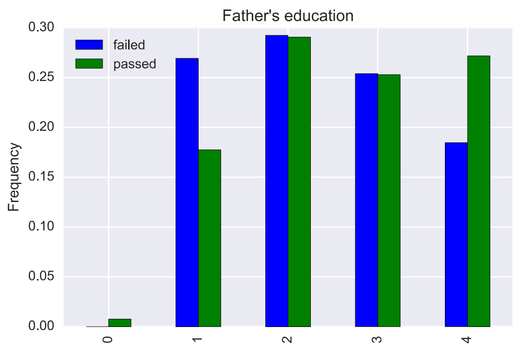

Figure 3: ‘Fedu’ binarising threshold = 3.5

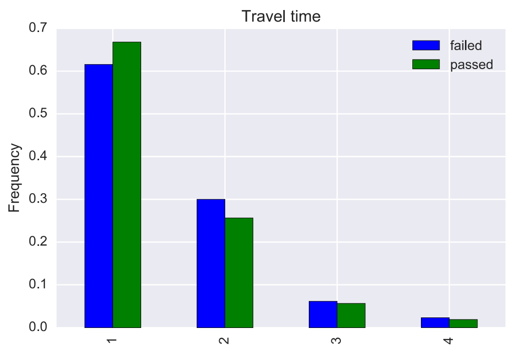

Figure 4: ‘traveltime’ binarising threshold = 1.5

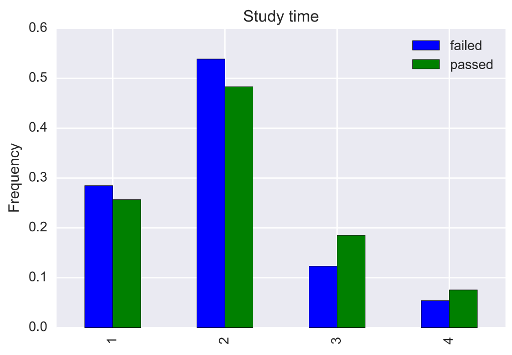

Figure 5: ‘studytime’ binarising threshold = 2.5

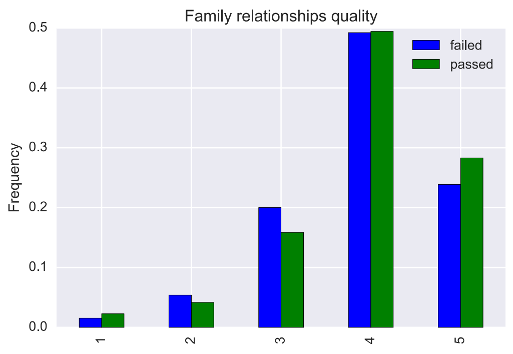

Figure 6: ‘famrel’ binarising threshold = 3.5

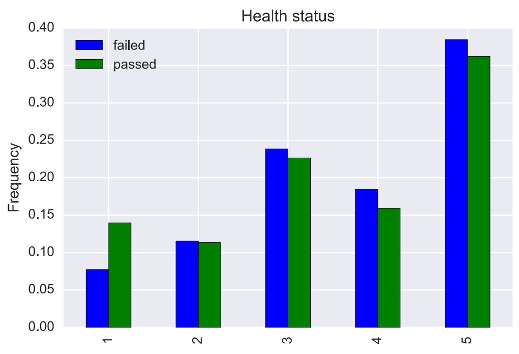

Figure 7: ‘health’ binarising threshold = 1.5

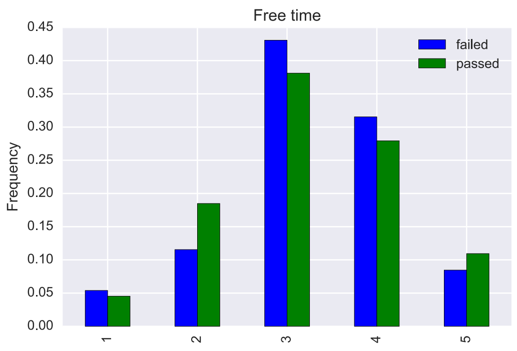

Figure 8: 'freetime' binarising threshold = 2.5

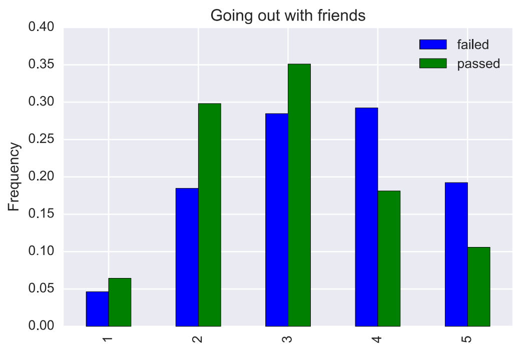

Figure 9: 'goout' binarising threshold = 3.5

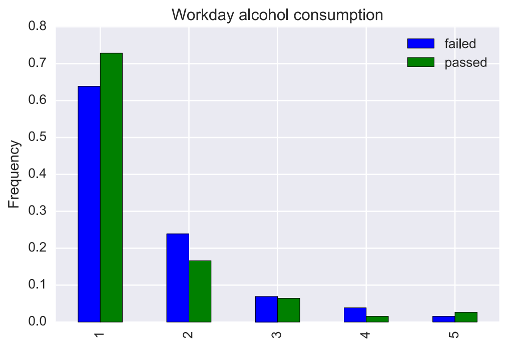

Figure 10: 'Walc' binarising threshold = 1.5

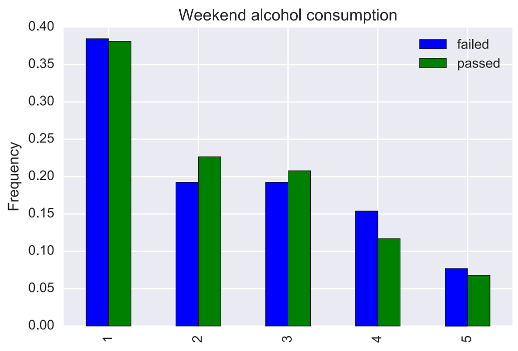

Figure 11: 'Dalc' binarising threshold = 3.5

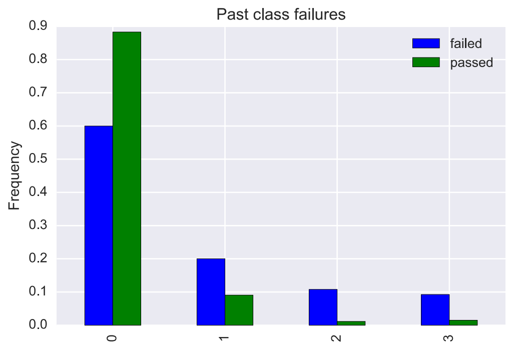

Figure 12: 'failures' binarising threshold = 0.5

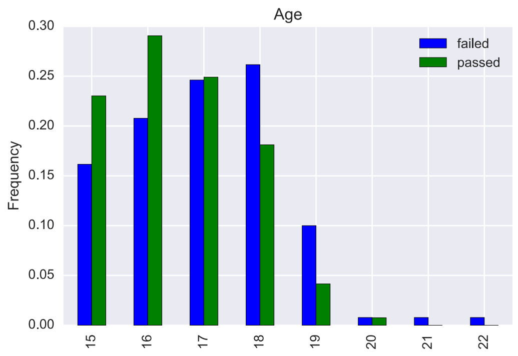

Figure 13: 'age' binarising threshold = 17.5

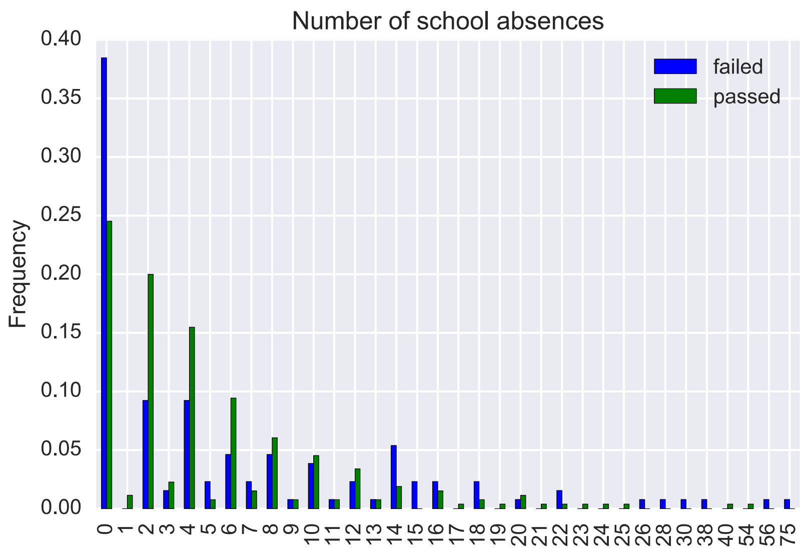

Figure 14: ‘absences’ binarising threshold = 0.5