Preparing the data set

I started by splitting up the data set, into a test set(20%) and a training set(80%). This would ensure the resulting algorithm would perform to the same standard on other datasets of the same format, without introducing any snooping bias. To ensure that both datasets were reflective of the entire data set itself I used stratified sampling. Already knowing that the highest education level achieved by a student would be an important factor, I chose to base the sampling on it.

With the data ready to be inspected, the first thing that became apparent was that the imd band attribute was missing values for a large quantity of students. Although simply removing it as an attribute would be easiest, the imd band region is highly linked to the region attribute of a student. Therefore, I chose to fill each missing imd band with that of the median band for that students region.

Both the vle and studentVle tables seemed highly informative, however, they both contained numerous entries per student. To use each resource as a unique feature would lead to an input having over 6000 features, making it likely to be over fitted. As a result I chose to use some derived features that would hopefully give a good representation of the data. These were the total number of clicks made by a student, the total number of resources they visited and the average number of days between which they accessed these resources.

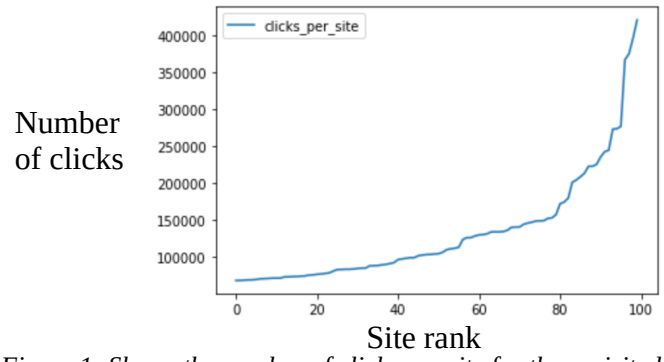
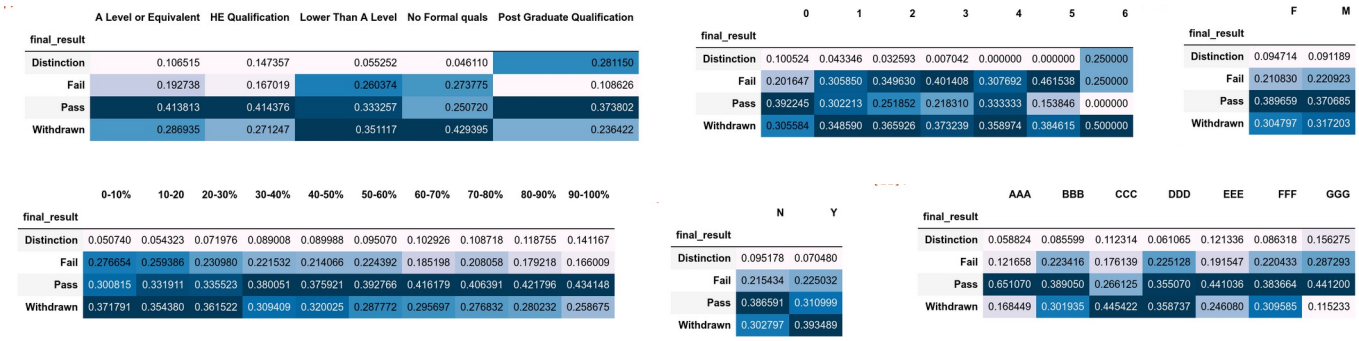


Figure 1: Shows the number of clicks per site for those visited by less than the average number of students per presentation

While these summarised how students interacted, they did not reflect what they interacted with. I thought that the resources that would be most informative would be those, not visited by everyone in a presentation but that were visited frequently by a minority. Therefore, I first identified those visited by less than the average number of students per presentation with more than 1000 clicks. Then sorting these by and plotting against their total number of clicks it became clear that the top 20 of these resources were visited far more per person than the rest, see figure 1. As a result I chose to use the number of clicks for each of these 20 sites as attributes.

A large number of the features provided were categorical, to be used this required them to be in a numerical format. To achieve this I used scikitLearns LabelBinarizer. This ultimately encoded each category as a one-hot vector. In addition to this, features such as the total number of clicks made by a student were very large compared to others. This could lead to unwarranted importance being placed on these attributes, therefore, I chose to standardise them using using sciKitLearns StandardScalar.

Feature selection



final_result	A Level or Equivalent	HE Qualification	Lower Than A Level	No Formal quals	Post Graduate Qualification
Distinction	0.106515	0.147357	0.055252	0.046110	0.281150
Fail	0.192738	0.167019	0.260374	0.273775	0.108626
Pass	0.413813	0.414376	0.333257	0.250720	0.373802
Withdrawn	0.286935	0.271247	0.351117	0.429395	0.236422

final_result	0	1	2	3	4	5	6
Distinction	0.100524	0.043346	0.032593	0.007042	0.000000	0.000000	0.250000
Fail	0.201647	0.305850	0.349630	0.401408	0.307692	0.461538	0.250000
Pass	0.392245	0.302213	0.251852	0.218310	0.333333	0.153846	0.000000
Withdrawn	0.305584	0.348590	0.365926	0.373239	0.358974	0.384615	0.500000

final_result	F	M
Distinction	0.094714	0.091189
Fail	0.210830	0.220923
Pass	0.389659	0.370685
Withdrawn	0.304797	0.317203

final_result	0-10%	10-20	20-30%	30-40%	40-50%	50-60%	60-70%	70-80%	80-90%	90-100%
Distinction	0.050740	0.054323	0.071976	0.089008	0.089988	0.095070	0.102926	0.108718	0.118755	0.141167
Fail	0.276654	0.259386	0.230980	0.221532	0.214066	0.224392	0.185198	0.208058	0.179218	0.166009
Pass	0.300815	0.331911	0.335523	0.380051	0.375921	0.392766	0.416179	0.406391	0.421796	0.434148
Withdrawn	0.371791	0.354380	0.361522	0.309409	0.320025	0.287772	0.295697	0.276832	0.280232	0.258675

final_result	N	Y
Distinction	0.095178	0.070480
Fail	0.215434	0.225032
Pass	0.386591	0.310999
Withdrawn	0.302797	0.393489

final_result	AAA	BBB	CCC	DDD	EEE	FFF	GGG
Distinction	0.058824	0.085599	0.112314	0.061065	0.121336	0.086318	0.156275
Fail	0.121658	0.223416	0.176139	0.225128	0.191547	0.220433	0.287293
Pass	0.651070	0.389050	0.266125	0.355070	0.441036	0.383664	0.441200
Withdrawn	0.168449	0.301935	0.445422	0.358737	0.246080	0.309585	0.115233

Figure 2: Highest level of education, previous number of attempts, gender, imd band, disability and module presentation(from left to right, top to bottom)

I first chose to focus on the relationships between categorical features. I did so by calculating the percentage of each result in each category. Looking at figure 2 it became clear that certain features such as gender had effectively no correlation with a students final result. Whilst on the other hand, features such as highest_education and disability had a clear correlation. In addition to this some attributes (such as age and imd band) also correlated highly with each other suggesting that maybe not all of them need be used as input.

For the relationships between the numerical attributes, on the other hand, I chose to use pandas corr method to generate a correlation matrix. This showed that all of the derived attributes other than those relating to clicks per site also shared a very high correlation, suggesting that maybe only one of them need be used as an input feature.

Based on this information I chose to start with the features code module, code presentation, highest education, region, imd band, age band, disability, total number of clicks per student, number of previous attempts, studied credits and total number of clicks per resource, for each of the top 20 resources.

Choosing a model

I had decided early on that I would limit the number of output types to just pass (pass and distinction) and fail(fail and withdrawn), meaning I would be implementing a binary classifier. Before committing to a model I decided to carry out cross validation on several classifiers to get a better idea of how they would likely perform, see figure 3.

```
Means: Accuracy:  0.8414894655964005
Means: Accuracy:  0.838804489917424
Means: Accuracy:  0.6840527126118684
Means: Accuracy:  0.8447877764207267
Means: Accuracy:  0.8446727163300316
```

Figure 3: Logistic Regression, Stochastic Gradient Descent, Naive Buyers, Support Vector Machine

All but Naive Byers performed surprisingly well. Therefore, I chose to use a combination of them to create two separate algorithms. The first would be a one vs one classifier, using a linear SVM to give a prediction for one of the four outcome types. This would then be fed into a SGD classifier along with the original input to predict which of the two main types the instance actually belonged to. The use of one vs one classification first followed by binary classification provides two different looks at the data, the intersection of which would enable the latter classification to identify new trends. The powerful but quick nature of SVMs made it an obvious choice to use in a one vs one format.

For the second, however, I used Linear Regression to first classify the inputs, this time only into the two main outcome types. I then used a Random Forest in a similar way to the second part of the first classifier, taking the output of the Logistic Regression along with the original input to give final classification.

Parameter Selection

SVM followed by SGD:

```
Best score and grid: 0.8477024310433684  {'grid_1': {'C': 50, 'gamma': 1, 'kernel': 'rbf'}, 'grid_2': {'alpha': 0.
01, 'loss': 'log', 'n_jobs': -1, 'penalty': 'l2'}}
```

Logistic Regression followed by Random Forests:

```
Best score and grid: 0.878538049988116  {'grid_1': {'C': 0.0001, 'penalty': 'l1', 'solver': 'liblinear'}, 'grid_2
': {'max features': 16, 'n estimators': 10}}
```

Figure 4: Optimal parameters and corresponding score found for each algorithm

Having chosen two models to pursue I then needed to find an optimal combination of parameters that would give the best possible predictions. To do so I decided to adapt sklearns grid search with cross validation such that it would explore all possible combinations across both components of each algorithm at once, not just independently. This would explore far more combinations and hopefully provide greater accuracy. The parameters that performed best (and so the ones I chose) for each algorithm can be seen in figure 4.

Comparison and Evaluation

In [39]:
```
from sklearn.metrics import accuracy_score

'''Reduces test output categories to just two - Pass or Fail'''
def classes_to_bin(res_data):
    final_res = []
    for i in range(len(res_data)):
        if res_data[i] == "Pass":
            final_res.append(0)
        elif res_data[i] == "Distinction":
            final_res.append(0)
        elif res_data[i] == "Fail":
            final_res.append(1)
        elif res_data[i] == "Withdrawn":
            final_res.append(1)
    return final_res

bi_class_test_res = np.asarray(classes_to_bin(test_results.values.ravel()))
log_reg_and_rand_bi.fit(prepared_data, bi_class_res)
prediction_test = log_reg_and_rand_bi.predict(prepared_test_data)
accuracy_score(bi_class_test_res, prediction_test)
```

Out[39]: 0.8015032980518484

In [31]:
```
from sklearn.metrics import accuracy_score

'''Reduces test output categories to just two - Pass or Fail'''
def classes_to_bin(res_data):
    final_res = []
    for i in range(len(res_data)):
        if res_data[i] == "Pass":
            final_res.append(0)
        elif res_data[i] == "Distinction":
            final_res.append(0)
        elif res_data[i] == "Fail":
            final_res.append(1)
        elif res_data[i] == "Withdrawn":
            final_res.append(1)
    return final_res

bi_class_test_res = np.asarray(classes_to_bin(test_results.values.ravel()))
ovo_svm_and_sgd.fit(prepared_data, four_class_res)
prediction_test = ovo_svm_and_sgd.predict(prepared_test_data)
accuracy_score(bi_class_test_res, prediction_test)
```

Out[31]: 0.8285013038809633

Figure 5: Performance on test data for: Logistic Regression and Random Forests, left, and SVM and SGD right

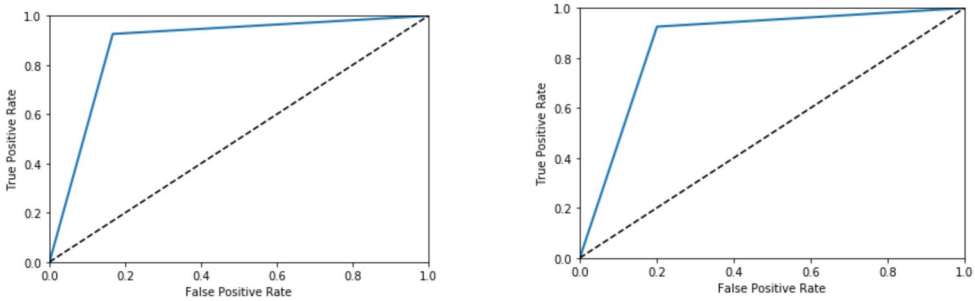


Figure 6: ROC curves for SVM followed by SGD, left, and Logistic Regression followed by Random Forests, right

Although the Logistic Regression and Random Forest classifier was seen to perform significantly better on the training data, both classifiers performed much more similarly on the test set. Furthermore, the false positive rate for the SVM and SGD classifier was lower making it more reliable. This change is likely due to the fact that Random Forests are known to over fit the data. In future this will require me to be more careful and stringent about the parameter search procedure, given this is the likely cause of such over fitting. To conclude the SVM and SGD classifier is best.