student_intervention

Fernando Hernandez

January 18, 2016

1 Project 2: Supervised Learning

1.1 Building a Student Intervention System

1.2 1. Classification vs Regression

Your goal is to identify students who might need early intervention - which type of supervised machine learning problem is this, classification or regression? Why?

This is a classic classification problem because we are trying to predict a discrete binary outcome - whether the student will pass or fail.

1.3 2. Exploring the Data

Now, can you find out the following facts about the dataset? - Total number of students - Number of students who passed - Number of students who failed - Graduation rate of the class (%) - Number of features

```
Total number of students: 395
Number of students who passed: 265
Number of students who failed: 130
Number of features: 31
Graduation rate of the class: 0.67%
```

1.4 3. Preparing the Data

In this section, we will prepare the data for modeling, training and testing.

1.4.1 Identify feature and target columns

It is often the case that the data you obtain contains non-numeric features. This can be a problem, as most machine learning algorithms expect numeric data to perform computations with.

Let's first separate our data into feature and target columns, and see if any features are non-numeric. **Note**: For this dataset, the last column (**passed**) is the target or label we are trying to predict.

```
Feature column(s):-
['school', 'sex', 'age', 'address', 'famsize', 'Pstatus', 'Medu', 'Fedu', 'Mjob', 'Fjob', 'reason', 'gu
Target column: passed

Feature values:-
  school sex  age address famsize Pstatus  Medu  Fedu     Mjob      Fjob  \
0     GP   F   18       U     GT3       A     4     4  at_home   teacher
```

```
1     GP   F   17       U     GT3       T     1     1  at_home     other
2     GP   F   15       U     LE3       T     1     1  at_home     other
3     GP   F   15       U     GT3       T     4     2   health  services
4     GP   F   16       U     GT3       T     3     3    other     other

    ...    higher internet  romantic  famrel  freetime goout Dalc Walc health  \
0   ...       yes       no        no       4         3     4    1    1      3
1   ...       yes      yes        no       5         3     3    1    1      3
2   ...       yes      yes        no       4         3     2    2    3      3
3   ...       yes      yes       yes       3         2     2    1    1      5
4   ...       yes       no        no       4         3     2    1    2      5

  absences
0        6
1        4
2       10
3        2
4        4

[5 rows x 30 columns]
```

1.4.2 Preprocess feature columns

As you can see, there are several non-numeric columns that need to be converted! Many of them are simply `yes`/`no`, e.g. `internet`. These can be reasonably converted into `1`/`0` (binary) values.

Other columns, like `Mjob` and `Fjob`, have more than two values, and are known as *categorical variables*. The recommended way to handle such a column is to create as many columns as possible values (e.g. `Fjob_teacher`, `Fjob_other`, `Fjob_services`, etc.), and assign a `1` to one of them and `0` to all others.

These generated columns are sometimes called *dummy variables*, and we will use the `pandas.get_dummies()` function to perform this transformation.

```
Processed feature columns (48):-
['school_GP', 'school_MS', 'sex_F', 'sex_M', 'age', 'address_R', 'address_U', 'famsize_GT3', 'famsize_LE3'
```

1.4.3 Split data into training and test sets

So far, we have converted all *categorical* features into numeric values. In this next step, we'll split the data (both features and corresponding labels) into training and test sets.

We'll use sklearn's StratifiedShuffleSplit to split our data.

A stratified split will ensure that we get a similar distribution of the target variable, 'passed', in both the training and testing sets.

This should be useful if our target y variable is highly unevenly distributed. It should also be slightly helpful in our case of having the constraint of a fairly small training/testing dataset since a 'bad' split with very little variety in the test set could cause poor estimates of model performance.

We should also explore the data a little bit to get a feel for the data. We'll also set the seed to make the analysis reproducible.

We can use the same indices to split the original dataset which didn't contain the dummy variables. This is because we can more easily explore the original training set with plotting libraries.

```
Training set: 300 samples
Test set: 95 samples
```

1.5 2.1 Exploring the Data - Revisited

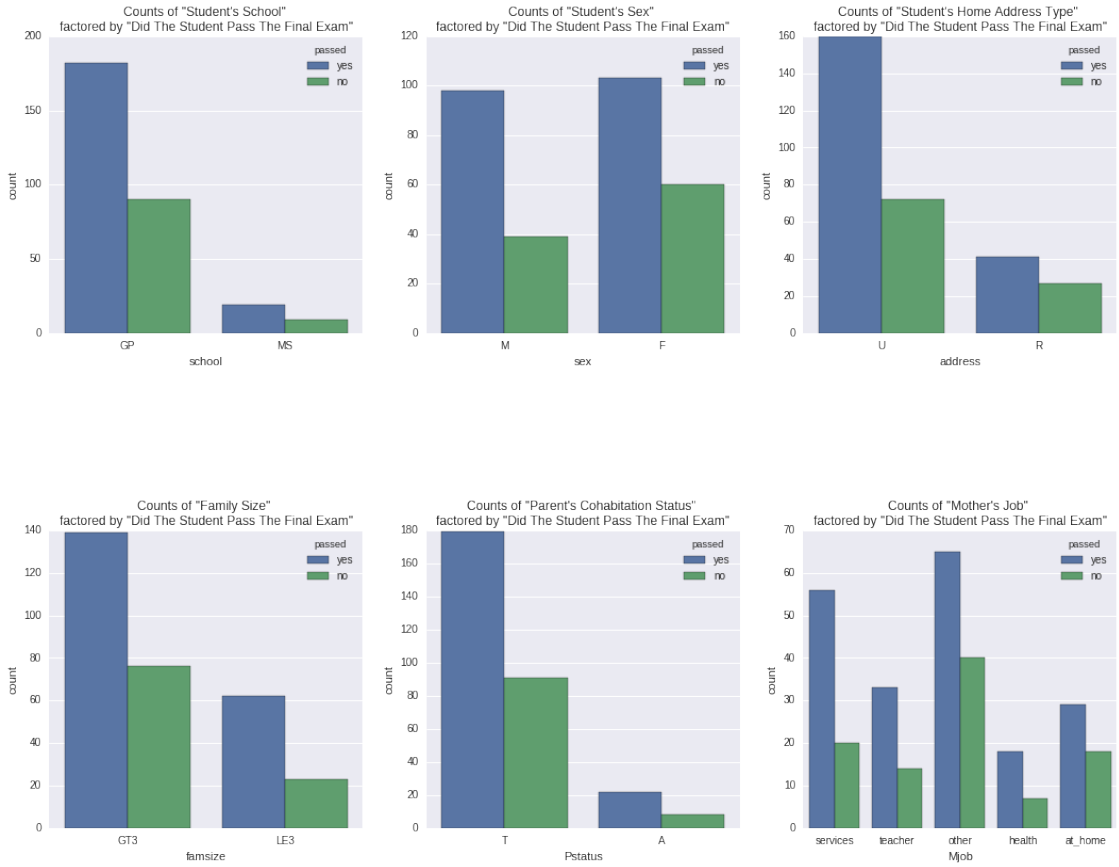
Now that we have a testing split pulled out and locked away in vault, let's get a feel for the training data.

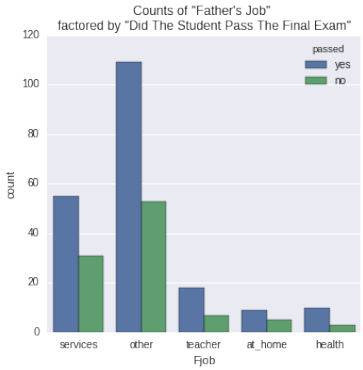
We don't want to go exploring the full dataset and risk peeking at our test data and influencing our model selections using any of the test data.

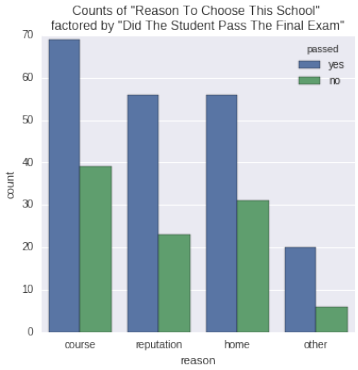
Although note: the training and test data should be distributed roughly the same anyway if we assume they are independent and identically distributed (i.i.d.) and split at random.

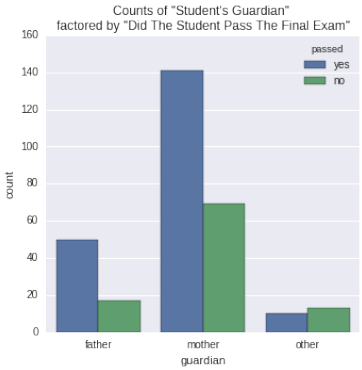
It should also be noted that we are not attempting to draw any conclusions about the data, but instead just get a feel for the data.

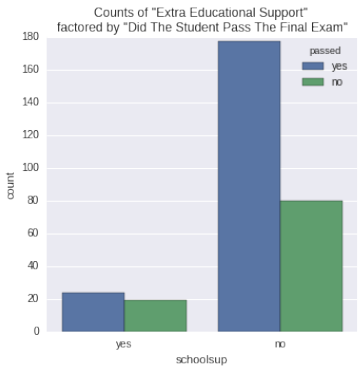
Here, we can plot the counts of the different categorical variables, factored by whether the student passed or not.

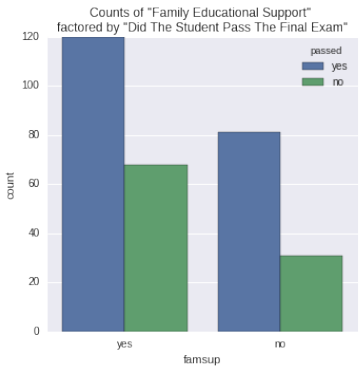



Counts of "Father's Job"
factored by "Did The Student Pass The Final Exam"
passed
yes
no
count
services
other
teacher
at_home
health
Fjob

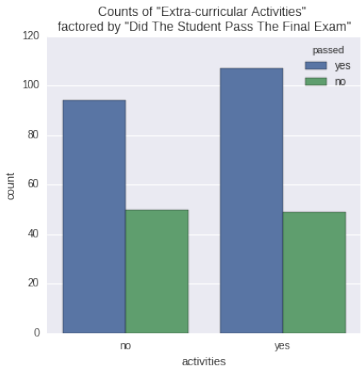

Counts of "Reason To Choose This School"
factored by "Did The Student Pass The Final Exam"
passed
yes
no
count
course
reputation
home
other
reason

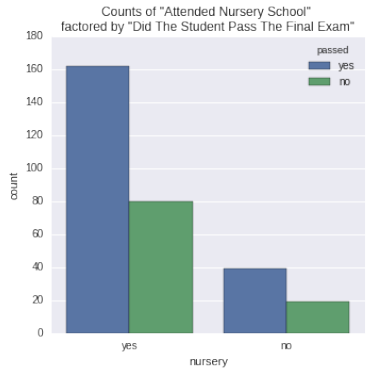

Counts of "Student's Guardian"
factored by "Did The Student Pass The Final Exam"
passed
yes
no
count
father
mother
other
guardian

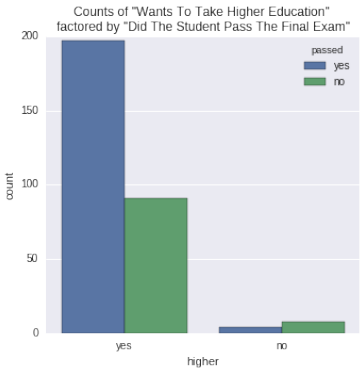

Counts of "Extra Educational Support"
factored by "Did The Student Pass The Final Exam"
passed
yes
no
count
yes
no
schoolsup

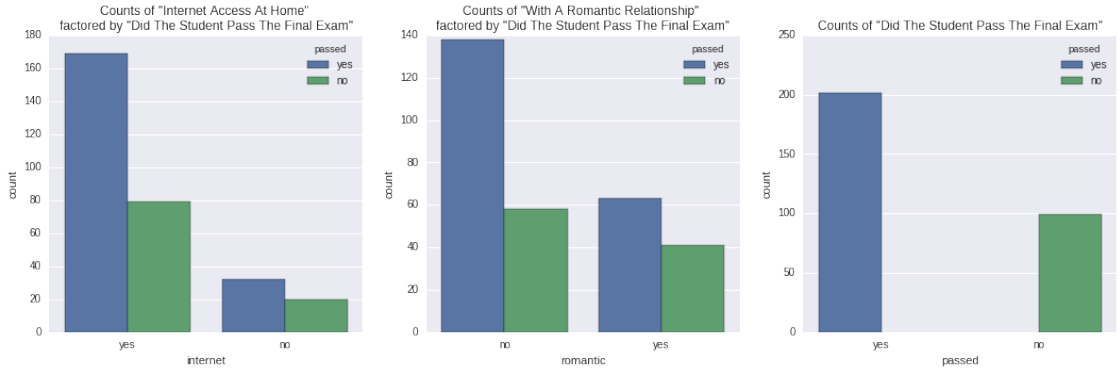

Counts of "Family Educational Support"
factored by "Did The Student Pass The Final Exam"
passed
yes
no
count
yes
no
famsup


Counts of "Extra Paid Classes Within The Course Subject"
factored by "Did The Student Pass The Final Exam"
passed
yes
no
count
no
yes
paid


Counts of "Extra-curricular Activities"
factored by "Did The Student Pass The Final Exam"
passed
yes
no
count
no
yes
activities


Counts of "Attended Nursery School"
factored by "Did The Student Pass The Final Exam"
passed
yes
no
count
yes
no
nursery


Counts of "Wants To Take Higher Education"
factored by "Did The Student Pass The Final Exam"
passed
yes
no
count
yes
no
higher



In most cases, there were not enough students in all of the choices available for each category to draw any meaningful conclusions.

Some distributions did show that there might be differences.

- "Internet access at home"

Those who had access to internet at home, had about twice as many students pass, while those without internet access were fairly close in counts of pass vs fail.

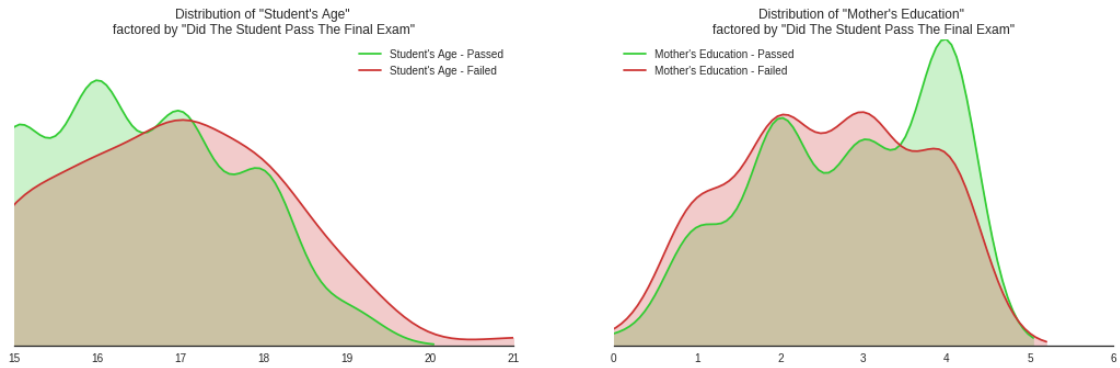
- "With a romantic relationship"

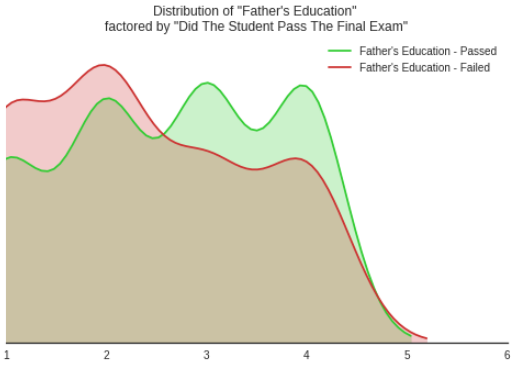
Those not in a romantic relationship seemed to pass relatively more frequently than those in relationships.

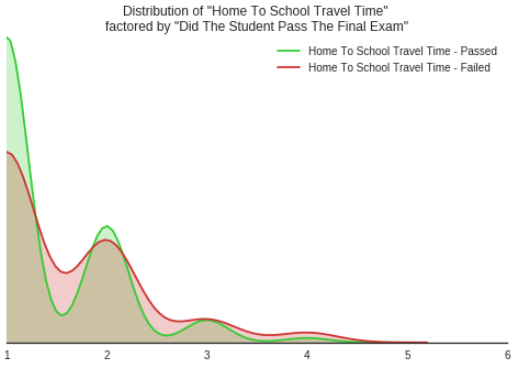
"Extra Education Support", "Student's School", and "Wants to take higher education" also showed hints of differing distributions depending on answers, but again, the number of students in each answer were too low to draw any conclusions by eye.

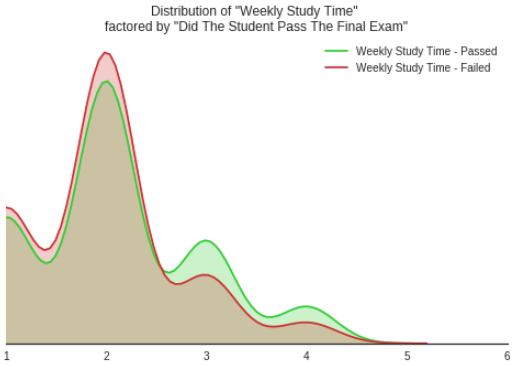
And again, we are not drawing any conclusions, but perhaps a decision tree classifier might fair well if pass/fail distributions are different depending on answers to some questions.

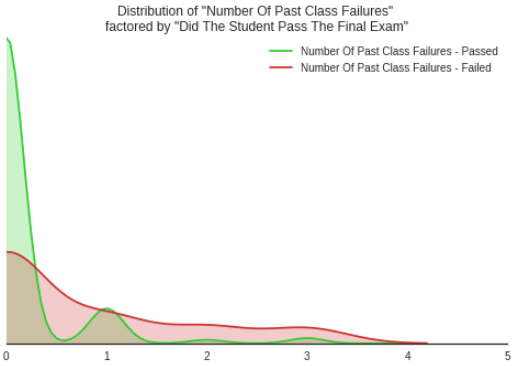
Next, let's take at the distributions of the more continuous numeric variables.

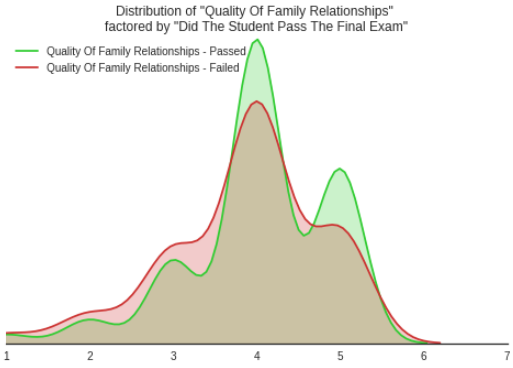


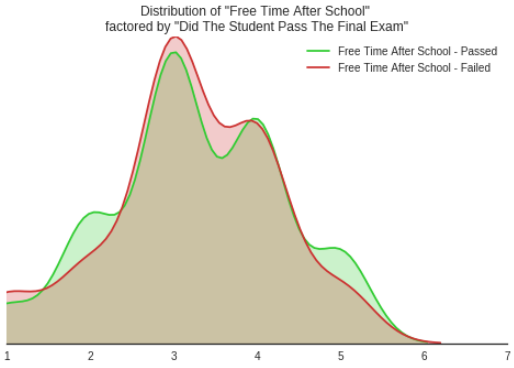

Distribution of "Father's Education"
factored by "Did The Student Pass The Final Exam"
Father's Education - Passed
Father's Education - Failed
1
2
3
4
5
6

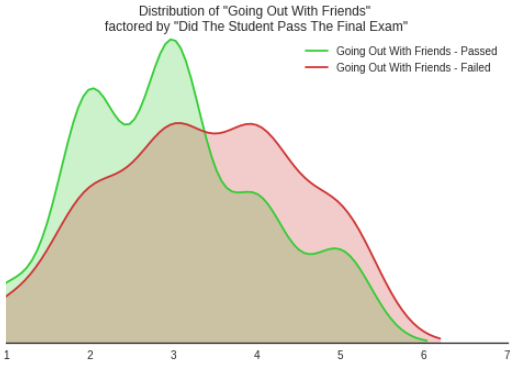

Distribution of "Home To School Travel Time"
factored by "Did The Student Pass The Final Exam"
Home To School Travel Time - Passed
Home To School Travel Time - Failed
1
2
3
4
5
6

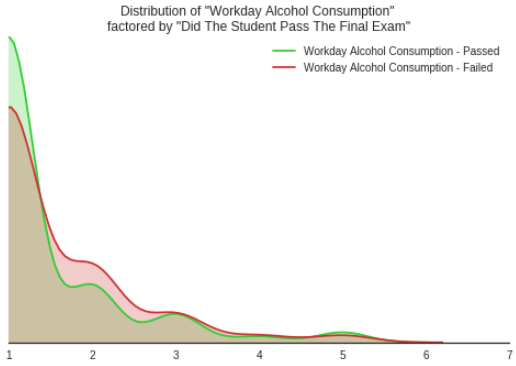

Distribution of "Weekly Study Time"
factored by "Did The Student Pass The Final Exam"
Weekly Study Time - Passed
Weekly Study Time - Failed
1
2
3
4
5
6

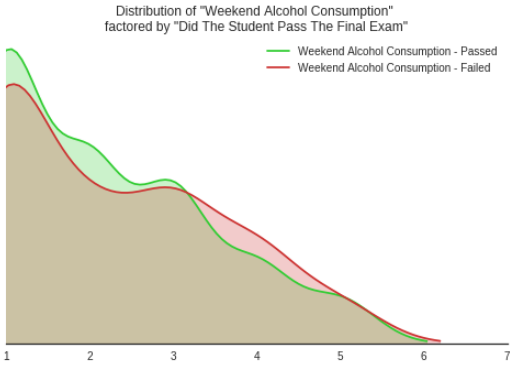

Distribution of "Number Of Past Class Failures"
factored by "Did The Student Pass The Final Exam"
Number Of Past Class Failures - Passed
Number Of Past Class Failures - Failed
0
1
2
3
4
5

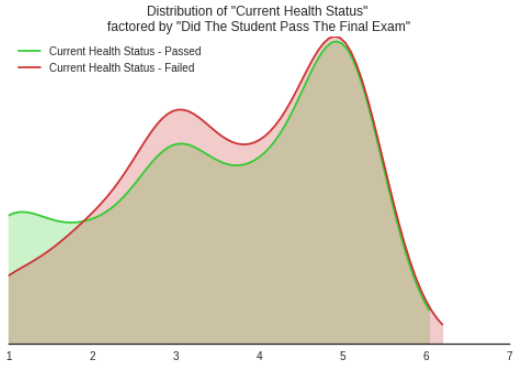

Distribution of "Quality Of Family Relationships"
factored by "Did The Student Pass The Final Exam"
Quality Of Family Relationships - Passed
Quality Of Family Relationships - Failed
1
2
3
4
5
6
7

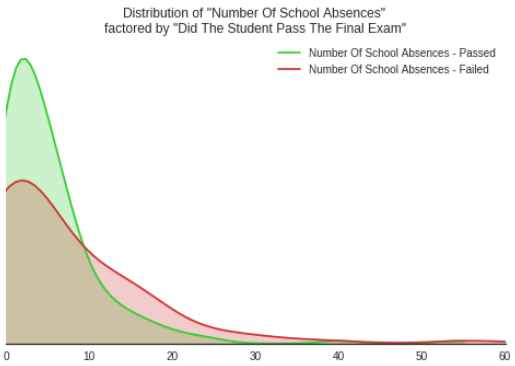

Distribution of "Free Time After School"
factored by "Did The Student Pass The Final Exam"
Free Time After School - Passed
Free Time After School - Failed
1
2
3
4
5
6
7











Here we can see the estimated probability distributions. A couple stand out as being possibly different for students who passed vs failed.

- "Number of past failures"

Of those students who passed, there was a very high probability of having 0 past failures. But of those failed, there was a more even distribution of probable values between 0-4 times. This is looks to potentially be large indicator of failing.

- "Going out with friends"

Passing students peaked around 2-3 times a week, while the probability mass of failing students centered around 4 times a week.

- "Mother's Education"

Passing students' mothers had a much higher probability mass at 4 (higher education) than non-passing students. Non-passing students' mothers also had a fairly higher probability mass at 1 (primary education (4th grade)) than passing students mothers' at 1.

"Student's Age" also shows possible signs of separation among the probability distributions.

Again, we can't draw any conclusions, but this also hints at including a decision tree (or nearest-neighbor) based machine learning algorithm for splitting the data based on key variables.

This would have the added benefit of giving some actionable variables to focus on when attempting to improve passing rate, something that a more opaque machine learning model like a neural net or support vector machine may not be able to provide.

1.6 4. Training and Evaluating Models

Choose 3 supervised learning models that are available in scikit-learn, and appropriate for this problem. For each model:

- What are the general applications of this model? What are its strengths and weaknesses?
- Given what you know about the data so far, why did you choose this model to apply?
- Fit this model to the training data, try to predict labels (for both training and test sets), and measure the F1 score. Repeat this process with different training set sizes (100, 200, 300), keeping test set constant.

Produce a table showing training time, prediction time, F1 score on training set and F1 score on test set, for each training set size.

Note: You need to produce 3 such tables - one for each model.

If we were simply trying to measure how well a school's changes are doing, pure prediction accuracy/F1 score maximization might be the goal.

For this particular case, I'd argue that maximum prediction accuracy/F_1 score is not our most important goal, but rather inference with high prediction accuracy.

Due to this criteria, > 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.

I'd argue that in this case, we are also trying to provide some actionable items to address to possibly increase the passing rate. In this case, I'd recommend more transparent supervised learning models which reduce irrelevant variables and produce important ones.

1.6.1 Three Supervised Learning Algorithms

Decision Trees Decision Trees biggest strength is that they are easy to intepret. Decision Trees would give a very direct a simple way to walk down a 'tree' of decision splits at certain features to identify which student's are likely to pass, or be at risk for failure.

Furthermore, they invariant to the scale of features, and "are immune to the effects of predictor outliers" Hastie et. al (2013). They also perform their own internal feature selection, and so become highly resistant to feature variable outliers.

One weakness is that Decision Trees have very high variance since it can grow arbirarily deep. This can cause Decision Trees to (over)fit training data very well, yet predict new data poorly, reducing accuracy.

Cross-validation will be required to get better performance when tuning such parameters as max depth.

Random Forests I would also include the more robust RandomForest model, with feature importance as a measure of important actionable variables. This method would in essence compute many trees, and uncover important features by finding those features that decrease total node impurity on average over all of the trees computed in the ensemble.

A major strength of Random Forests is their model variance reduction by way of bagging.

Bagging Bagging is a technique which averages many noisy but unbiased models to retain low bias, while reducing variance. In our case, Random Forests will accomplish this by averaging across many Decision Trees.

In bagging, many replications of the training dataset are made using random sampling with replacement. Each of these datasets are used to build a model and tested on the out-of-samples not used. Then all models vote and their results are averaged together greatly reducing the model variance.

Since each generated tree is identically distributed, the expecation of an average of any number of trees is the same as the expectation of any individual tree. This means that the bias essentially remains the same for any number of bagged trees.

One weakness though is that you do have to grow many more trees. Many tree must be grown, especially if we have many (irrelevant) features since the more irrelevant features we have, the lower the chance of including relevant features in each randomly sampled dataset/tree. We can try to address this by tuning the max_features parameters in a grid search tuning.

This can become a computational problem if using a high K-fold cross-validation procedure in conjunction with a deep parameter grid-search. (Luckliy we can make use of common multi-core processors to speed this up significantly.)

Another weakness is the slightly loss of interpretability. We can find the most important features, on average, but it is not as simple as a decision tree for identifying exact paths to different groups of target students.

Logistic Regression A third method might be logistic regression with an L1 regularizer to perform a sort of feature selection to keep only the most important variables in the model.

Here we might be able to find the features that can have the most impact on accuracy/F1 score while ignoring less important ones. L1 regularization strength is a parameter to address this in grid search tuning.

A strength is that Logistic Regression (with L1 penalty) is a fairly simple and very highly optimized quadratic programming model, so it should be almost instaneous to train for our datasets.

(While Logistic Regression has a closed form solution (for up to maybe 10,000's of students), there is no closed form solution when including L1 regularization.)

It is also fairly interpretable as we can see negative and positive correlation strengths for relevant features.

One obvious weakness is that logistic regression assumes passing/failing students are linearly separable by a weighted sum of the features that have been measured. This gives this model fairly low bias, and much less variance than a single Decision Tree.

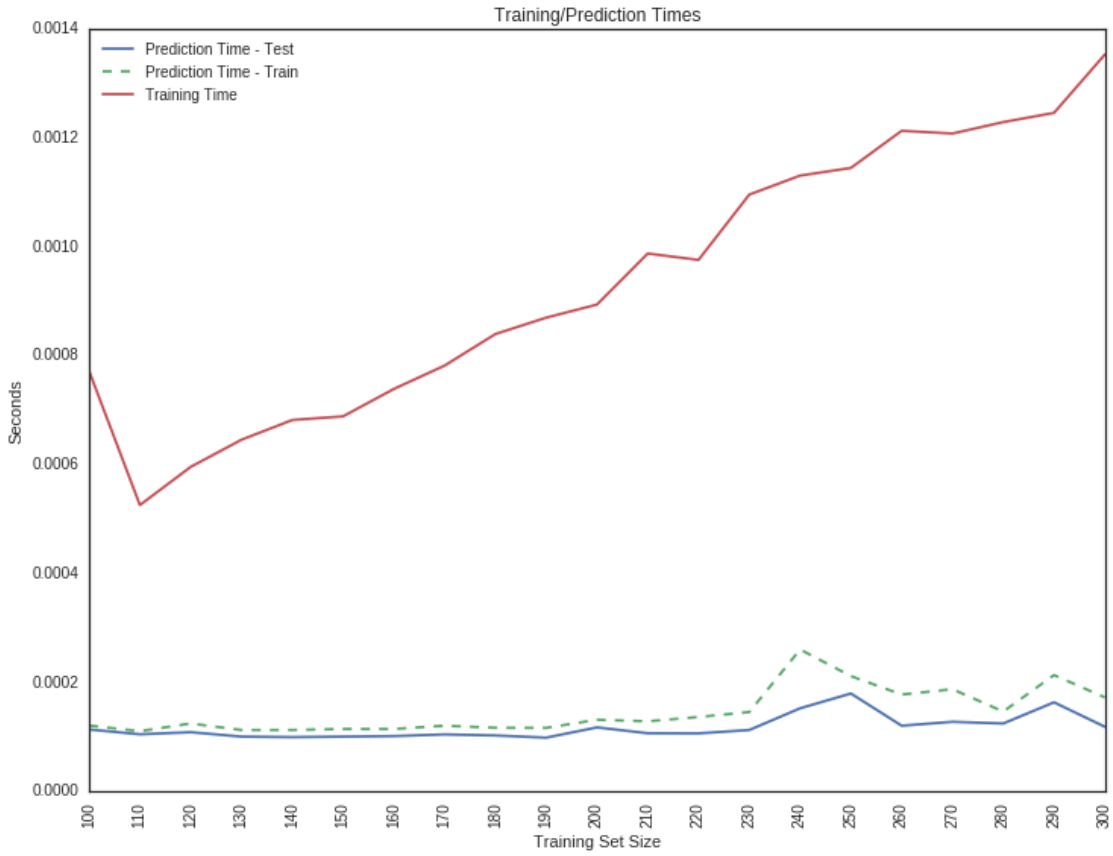
SVM or neural networks might deliver high accuracy as well, but we would only be measuring the prediction accuracy of the given set of measured variables with no concrete idea of what to address to possibly increase the passing rate of future students.

For now, our training set is small enough that we might be forgiven for foregoing a validation set and using 5-fold cross-validation in our grid-search in its place.

1.7 Decision Tree

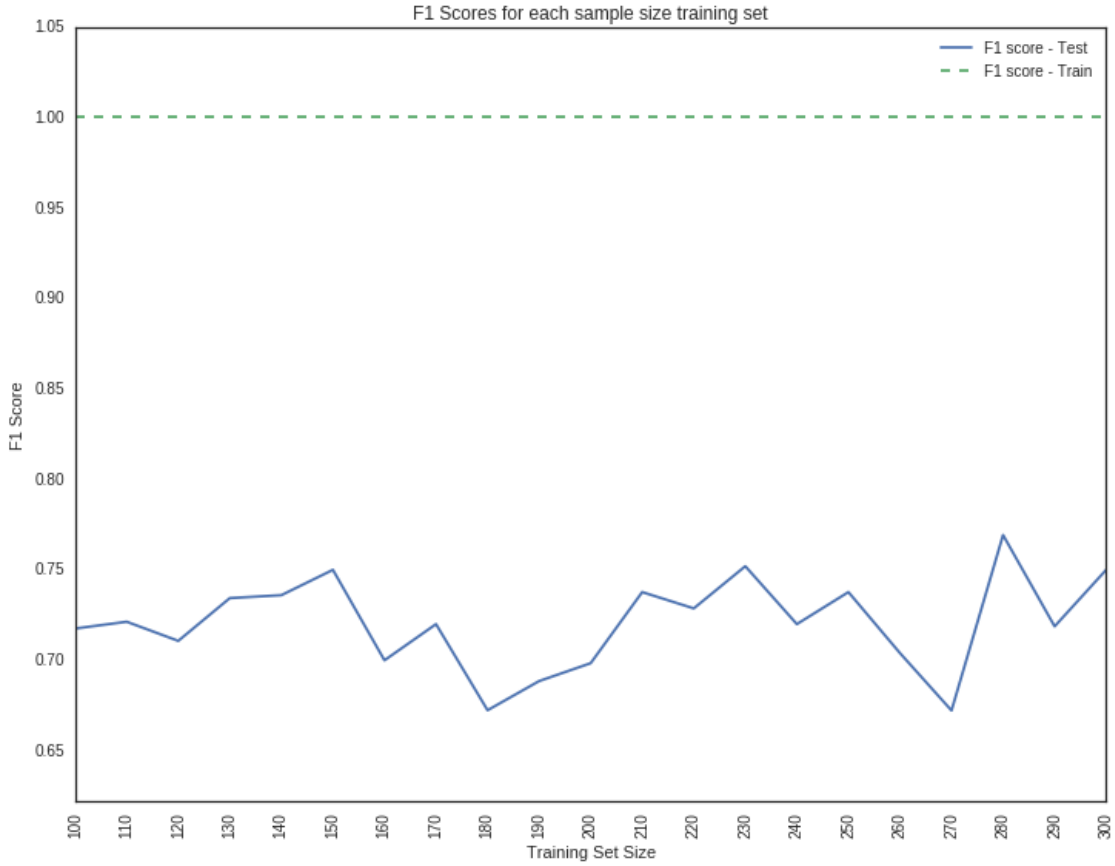
Out[21]:

	F1_test	F1_train	Predict_time_test	Predict_time_train	Training_size	Training_time
0	0.717557	1	0.000113	0.000120	100	0.000771
1	0.721311	1	0.000104	0.000110	110	0.000525
2	0.710744	1	0.000108	0.000124	120	0.000595
3	0.734375	1	0.000100	0.000112	130	0.000645
4	0.736000	1	0.000099	0.000112	140	0.000681
5	0.750000	1	0.000100	0.000114	150	0.000688
6	0.700000	1	0.000101	0.000114	160	0.000738
7	0.720000	1	0.000104	0.000120	170	0.000781
8	0.672414	1	0.000102	0.000116	180	0.000839
9	0.688525	1	0.000098	0.000116	190	0.000869
10	0.698413	1	0.000117	0.000131	200	0.000893
11	0.737705	1	0.000106	0.000128	210	0.000987
12	0.728682	1	0.000106	0.000136	220	0.000975
13	0.752000	1	0.000112	0.000145	230	0.001095
14	0.720000	1	0.000152	0.000260	240	0.001130
15	0.737705	1	0.000179	0.000211	250	0.001144
16	0.704000	1	0.000120	0.000177	260	0.001212
17	0.672269	1	0.000127	0.000187	270	0.001207
18	0.769231	1	0.000124	0.000146	280	0.001228
19	0.718750	1	0.000163	0.000213	290	0.001245
20	0.750000	1	0.000118	0.000172	300	0.001352



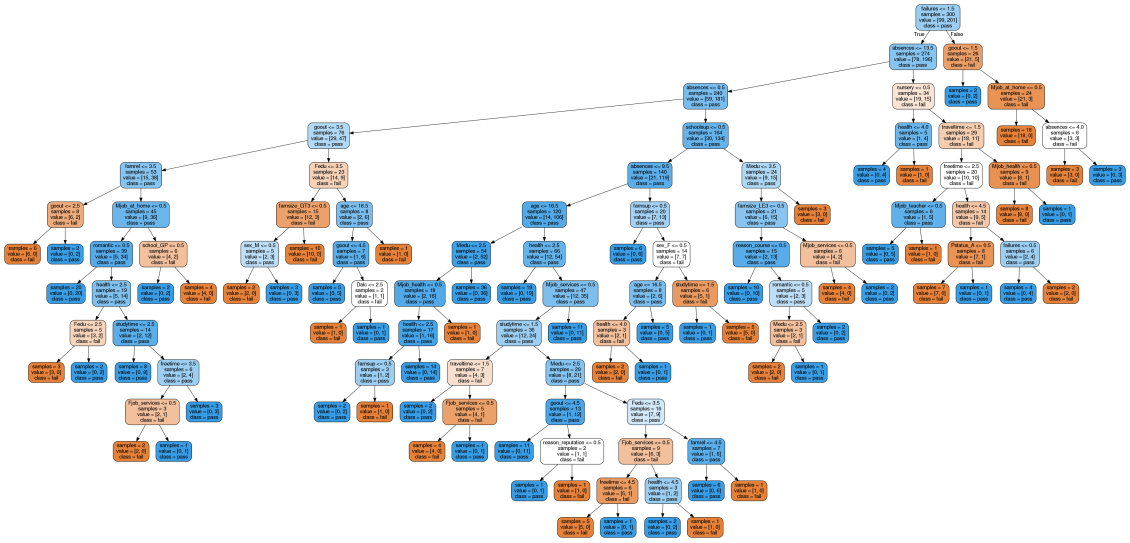
Training time looks to increase fairly linearly as we train the model on more data. Training times are currently in the sub-milliseconds so more training data should have minimal impact on computation and memory requirements for a single computer.

Prediction times stays at negligible values.



Test scores seem to get slightly better with more data and should show slightly more of an increase with more proper tuning.

It ended up at around 0.74-0.75 F1 score on the full training dataset of 300. `Out[28]:`



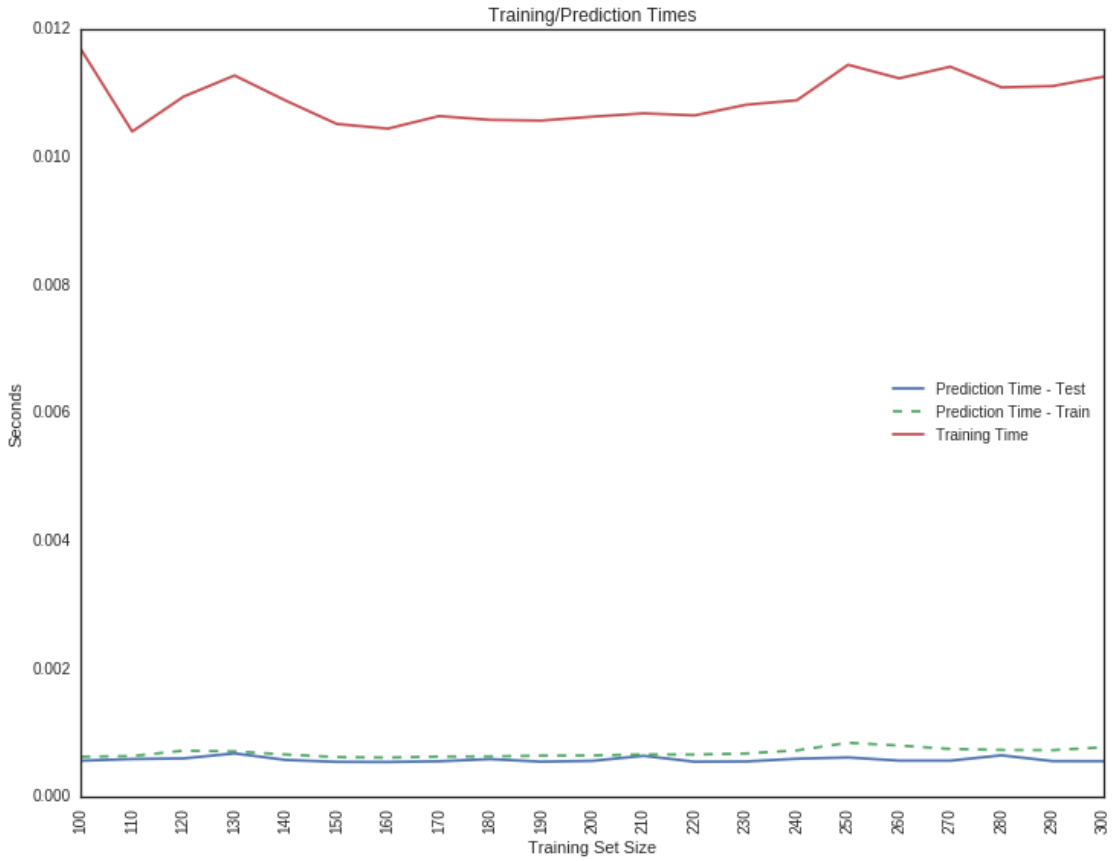
Here, we can visualize the tree and see that this tree knows the training set extremely well, but will need some tuning to generalize better to unseen data.

By default, our Decision Tree doesn't really have many bounds on how big it can grow to fit the data making it quite large.

1.8 Random Forest

Out[29]:

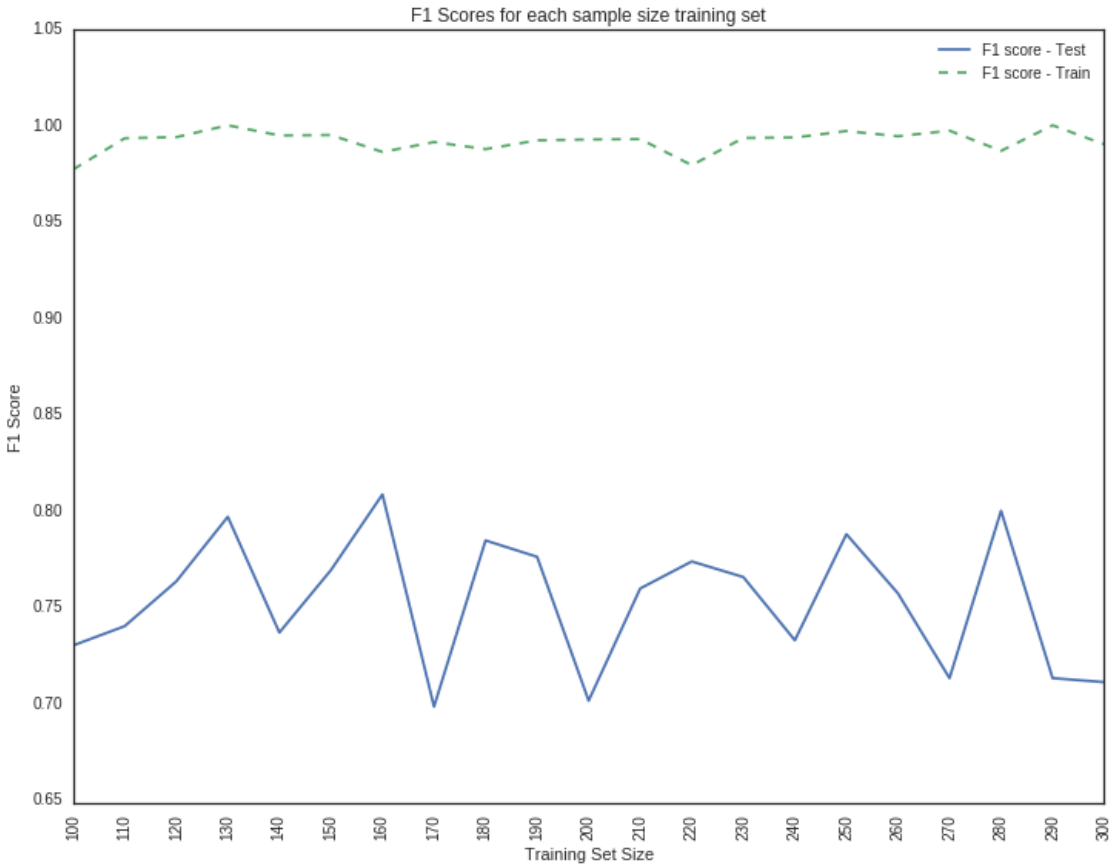
	F1_test	F1_train	Predict_time_test	Predict_time_train	Training_size	Training_time
0	0.730159	0.977099	0.000560	0.000622	100	0.011673
1	0.740157	0.993289	0.000586	0.000633	110	0.010393
2	0.763359	0.993865	0.000596	0.000717	120	0.010937
3	0.796875	1.000000	0.000674	0.000706	130	0.011269
4	0.736842	0.994764	0.000572	0.000657	140	0.010877
5	0.769231	0.994975	0.000542	0.000617	150	0.010511
6	0.808511	0.986175	0.000540	0.000609	160	0.010438
7	0.698413	0.991304	0.000551	0.000625	170	0.010634
8	0.784615	0.987654	0.000585	0.000627	180	0.010575
9	0.776119	0.992188	0.000545	0.000639	190	0.010564
10	0.701493	0.992647	0.000558	0.000642	200	0.010625
11	0.759690	0.992857	0.000636	0.000658	210	0.010677
12	0.773723	0.979310	0.000544	0.000656	220	0.010645
13	0.765625	0.993421	0.000548	0.000673	230	0.010811
14	0.732824	0.993671	0.000591	0.000721	240	0.010882
15	0.787879	0.997033	0.000611	0.000841	250	0.011436
16	0.757143	0.994318	0.000561	0.000797	260	0.011224
17	0.713178	0.997230	0.000561	0.000744	270	0.011406
18	0.800000	0.986737	0.000644	0.000728	280	0.011083
19	0.713178	1.000000	0.000554	0.000725	290	0.011103
20	0.711111	0.990148	0.000553	0.000771	300	0.011250



Random Forests are still fast given the size of our dataset, but much slower comparitavely to our other models.

The default trees grown are a paltry 10, so training time should increase as we increase this number to reduce model variance in our grid tuning.

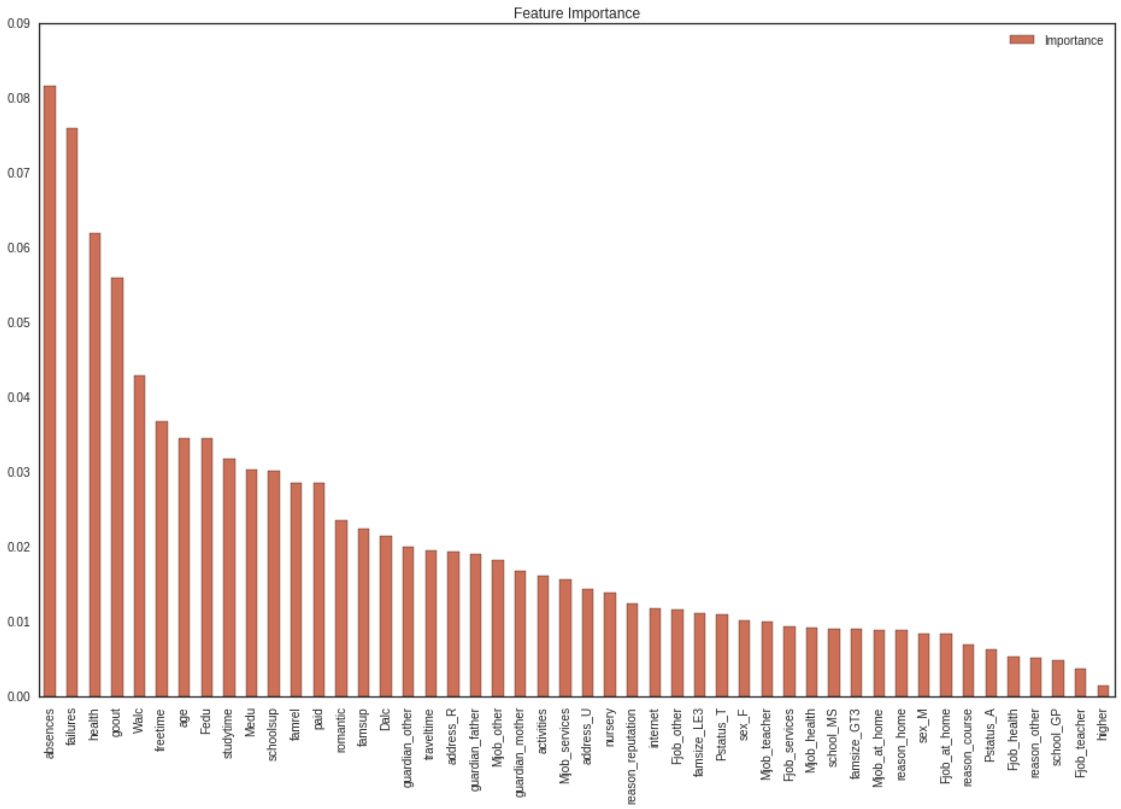
Predictions times though are still negligible.



Our untuned model looks to be stuck vacillating around a F1 score range of 0.70 to 0.79 on the test data. It ended up around 0.71 F1 score on the full training dataset of 300.

With only 10 trees grown in the default sklearn Random Forest implementation, it isn't too surprising to see high variance like Decision Tree model. Number of trees grown is definitely a parameter we should tune (or at least increase from 10) to lower the model variance.

```
Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff90a32ed50>
```



We can pull out the feature importances and see that many top important variables seem to be the same general ones we noticed in our cursory exploratory plots.

The order of some of the top features might change slightly between training runs, but the same top features stay near the top.

1.9 Logistic Regression

For logistic regression, we will standardize the features by by removing the mean and scaling to unit variance.

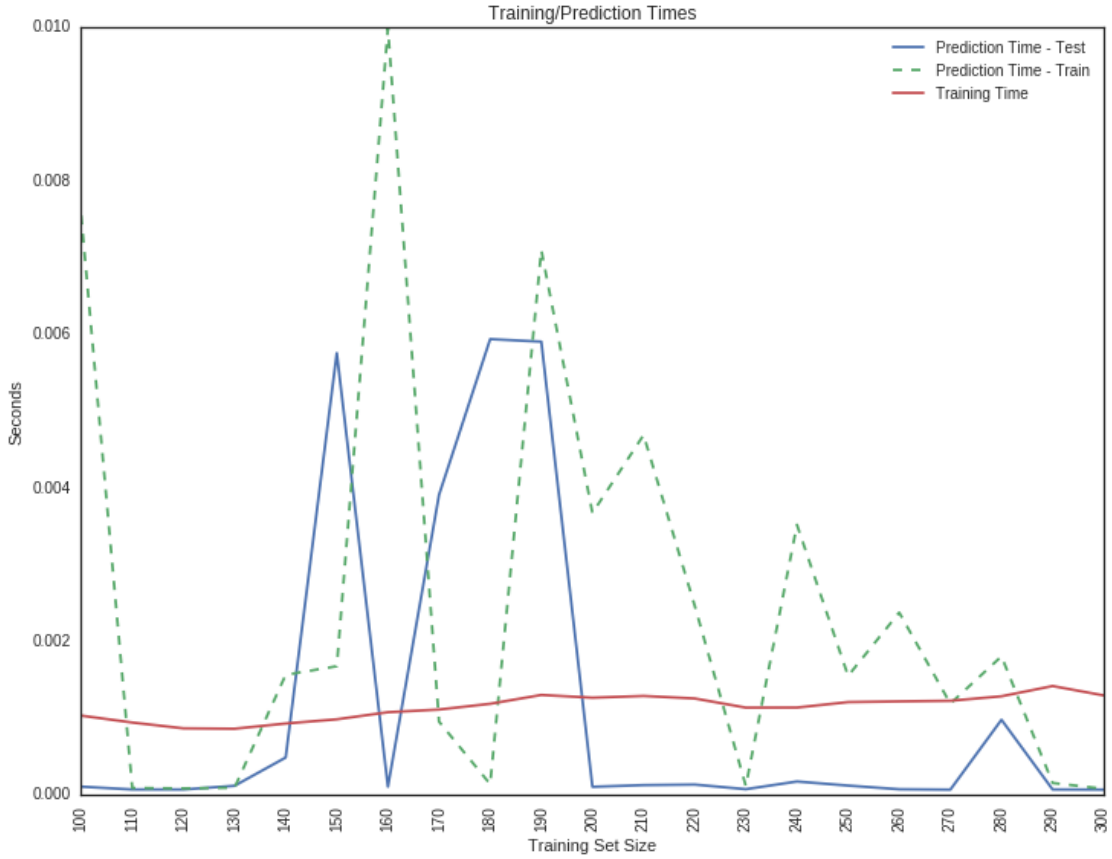
> Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual feature do not more or less look like standard normally distributed data (e.g. Gaussian with 0 mean and unit variance).
>
> For instance many elements used in the objective function of a learning algorithm (such as the RBF kernel of Support Vector Machines or the L1 and L2 regularizers of linear models) assume that all features are centered around 0 and have variance in the same order.
>
> If a feature has a variance that is orders of magnitude larger that others, it might dominate the objective function and make the estimator unable to learn from other features correctly as expected.

This is so everything is on the same scale when weighting our variables, which could be on vastly different scales (age vs absences vs etc.) and in turn, affect model performance.

This can also help our algorithm avoid under/overflow errors among other things.

Out[36]:

	F1_test	F1_train	Predict_time_test	Predict_time_train	Training_size	Training_time
0	0.725806	0.934307	0.000103	0.007542	100	0.001028
1	0.694215	0.911392	0.000066	0.000086	110	0.000938
2	0.715447	0.923977	0.000066	0.000080	120	0.000862
3	0.717557	0.903226	0.000114	0.000089	130	0.000857
4	0.761905	0.855721	0.000482	0.001559	140	0.000925
5	0.784000	0.872038	0.005750	0.001671	150	0.000980
6	0.771654	0.868421	0.000102	0.009980	160	0.001072
7	0.769231	0.857143	0.003904	0.000949	170	0.001107
8	0.769231	0.853755	0.005931	0.000135	180	0.001182
9	0.775194	0.851852	0.005898	0.007091	190	0.001297
10	0.775194	0.865248	0.000101	0.003667	200	0.001261
11	0.750000	0.844595	0.000123	0.004682	210	0.001284
12	0.750000	0.836601	0.000130	0.002460	220	0.001251
13	0.724409	0.833333	0.000070	0.000111	230	0.001132
14	0.748092	0.837758	0.000170	0.003513	240	0.001133
15	0.738462	0.836565	0.000117	0.001546	250	0.001204
16	0.742424	0.833333	0.000069	0.002370	260	0.001213
17	0.738462	0.829897	0.000064	0.001184	270	0.001221
18	0.764706	0.840796	0.000974	0.001800	280	0.001280
19	0.779412	0.839329	0.000066	0.000151	290	0.001413
20	0.768116	0.836028	0.000064	0.000075	300	0.001291



Training time increases linearly with more examples, and is the fastest of the three algorithms at these training set sizes.

Prediction time is so quick on the order of micro-seconds that small fluctuations in the state of our computer can cause it to artificially seem to randomly increase.

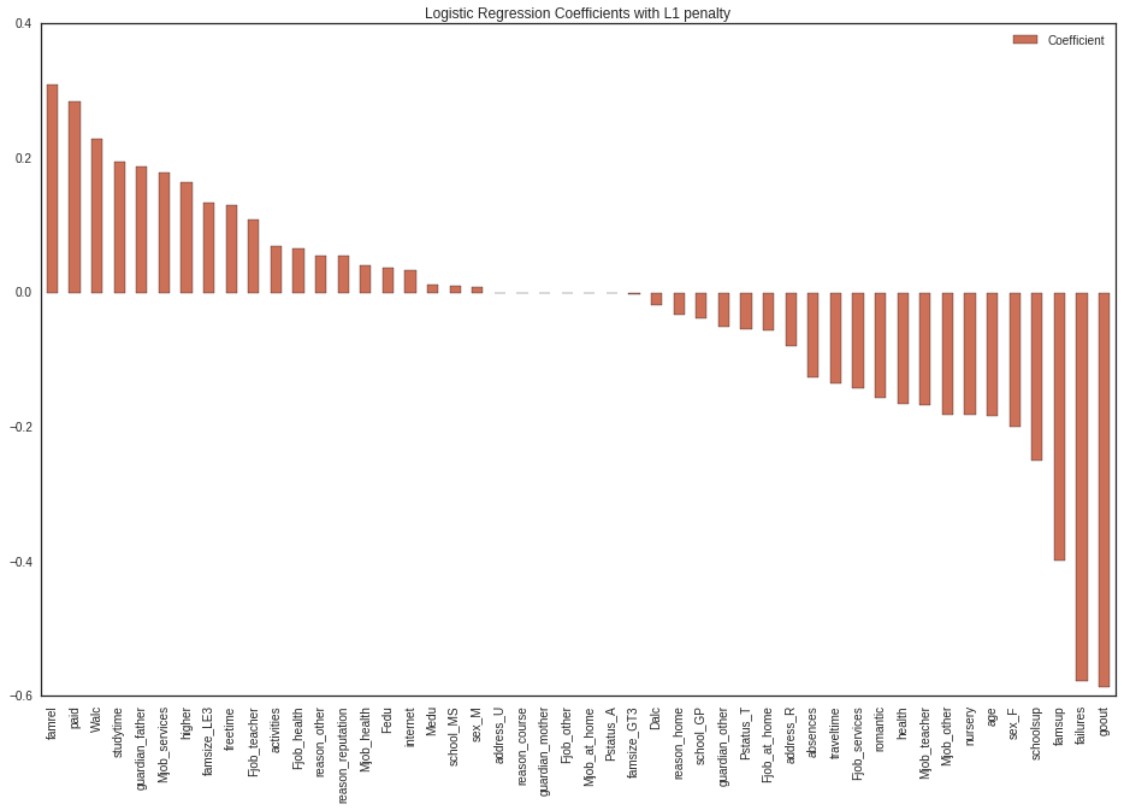
It ended up around 0.768 F1 score on the full training dataset of 300.

The F1 score is more realistic on the training set as we get more data, probably due in part to the L1 regularization.

The test set F1 score seems to be increasing and converging with the training set F1 score.

At about 140 samples, the model makes a jump in F_1 score where it stabilizes and slightly increases with more data.

```
Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff909d9e250>
```



Out[42]:

	Coefficient	Feature
famrel	0.309445	famrel
paid	0.283755	paid
Walc	0.228120	Walc
studytime	0.195553	studytime

Top 4 positively correlated features.

Out[43]:

	Coefficient	Feature
reason_course	0.000000	reason_course
guardian_mother	0.000000	guardian_mother
Fjob_other	0.000000	Fjob_other
Mjob_at_home	0.000000	Mjob_at_home
Pstatus_A	0.000000	Pstatus_A
famsize_GT3	-0.003295	famsize_GT3

Here we can see that indeed the L1 penalty has forced some feature weights to 0.

Out[44]:

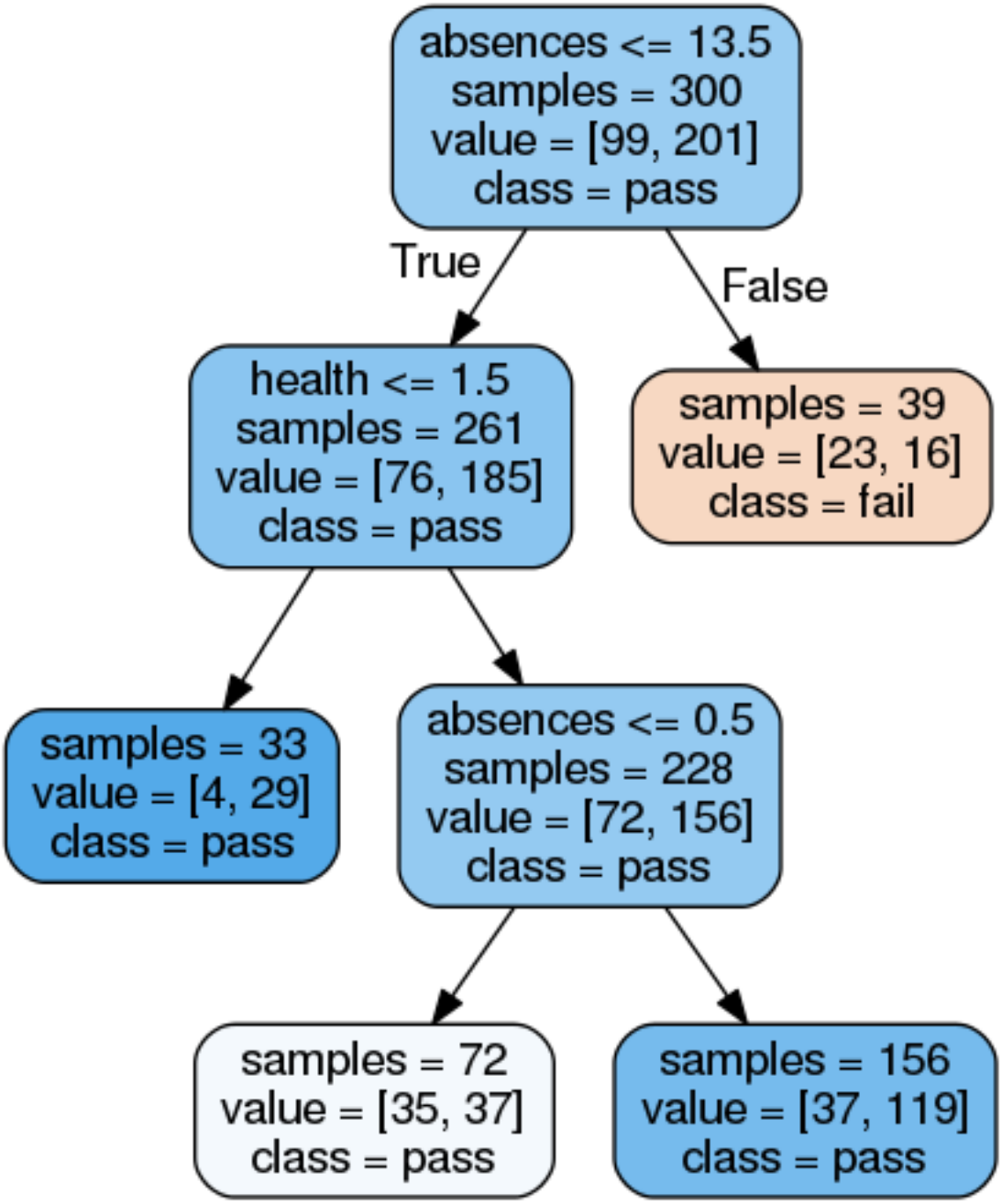
	Coefficient	Feature
schoolsup	-0.250496	schoolsup
famsup	-0.397680	famsup
failures	-0.578628	failures
goout	-0.586460	goout

Top 4 negatively correlated features.

1.10 Fine-tune your model and report the best F1 score

1.11 Decision Tree Tuning

```
Fitting 5 folds for each of 1344 candidates, totalling 6720 fits

[Parallel(n_jobs=-1)]: Done 2284 tasks      | elapsed:    2.1s
[Parallel(n_jobs=-1)]: Done 6720 out of 6720 | elapsed:    5.0s finished


Best cross-validated tuned F1 score for Decision Tree: 0.818863479075

Best parameters tuning Decision Tree:

Out[95]: {'criterion': 'gini',
          'max_depth': 8,
          'max_features': 5,
          'min_samples_leaf': 10,
          'min_samples_split': 60}

Out[53]:
```



Visualizing the final tuned Decision Tree, we can see that it is much smaller. This creates a model with more bias, and less variance. It would also generalize slightly better since it is less overfit to the training set.

But, as we can see from many of the leaf nodes still containing a fair amount of pass/fail students, it's not terribly accurate. In fact, only when splitting at absences greater than or equal to 13.5, did the model predict a student would fail.

Another problem is that it still has very high variance. Retraining/retuning can produce a completely different tree with completely different variables to split on. This makes finding the most important variables extremely hard using only a Decision Tree.

Out[55]:

	F1_test	F1_train	Predict_time_test	Predict_time_train	Training_time
0	0.780142	0.813187	0.000105	0.000156	0.000592

1.12 Random Forest Tuning

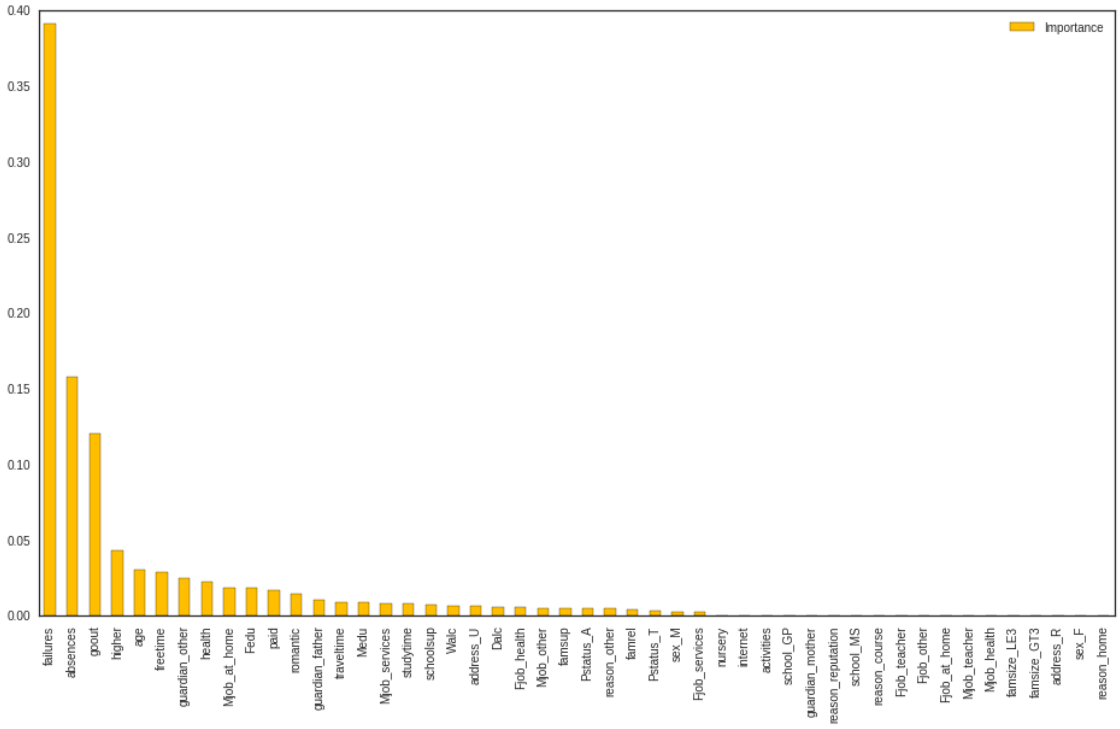
```
Fitting 5 folds for each of 280 candidates, totalling 1400 fits

[Parallel(n_jobs=-1)]: Done 258 tasks      | elapsed:   12.3s
[Parallel(n_jobs=-1)]: Done 558 tasks      | elapsed:   26.5s
[Parallel(n_jobs=-1)]: Done 1058 tasks      | elapsed:   51.4s
[Parallel(n_jobs=-1)]: Done 1400 out of 1400 | elapsed:  1.2min finished


Best cross-validated tuned F1 score for Decision Tree: 0.826192867707

Best tuned parameters for Random Forest:
```

Out[99]: {'criterion': 'gini', 'max_depth': 4, 'max_features': 30, 'n_estimators': 50}

Out[60]:

	Feature	Importance
32	failures	0.391542
47	absences	0.157802
43	goout	0.120214
38	higher	0.043425
4	age	0.030484

Out[79]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff90ab06990>



Here we can see that past failures, absences, and going out during the week seem to be very important variables when identifying students who may pass.

This estimate of importance also has low variance since these are estimates from many estimators from many bagged datasets. These should be top important variables even if the model is tuned multiple times.

Out[65]:

	F1_test	F1_train	Predict_time_test	Predict_time_train	Training_time
0	0.8	0.836518	0.001781	0.002373	0.110616

1.13 Logistic Regression Tuning

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits

[Parallel(n_jobs=-1)]: Done 150 out of 150 | elapsed:    0.7s finished
```

Out[68]:

```
GridSearchCV(cv=5, error_score='raise',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
       fit_params={}, iid=True, n_jobs=-1,
       param_grid={'penalty': ['l1', 'l2'], 'C': [0.05, 0.001, 0.01, 0.1, 1, 10, 100, 0.2, 0.3,
       pre_dispatch='2*n_jobs', refit=True,
       scoring=make_scorer(f1_score, pos_label=yes), verbose=1)
```

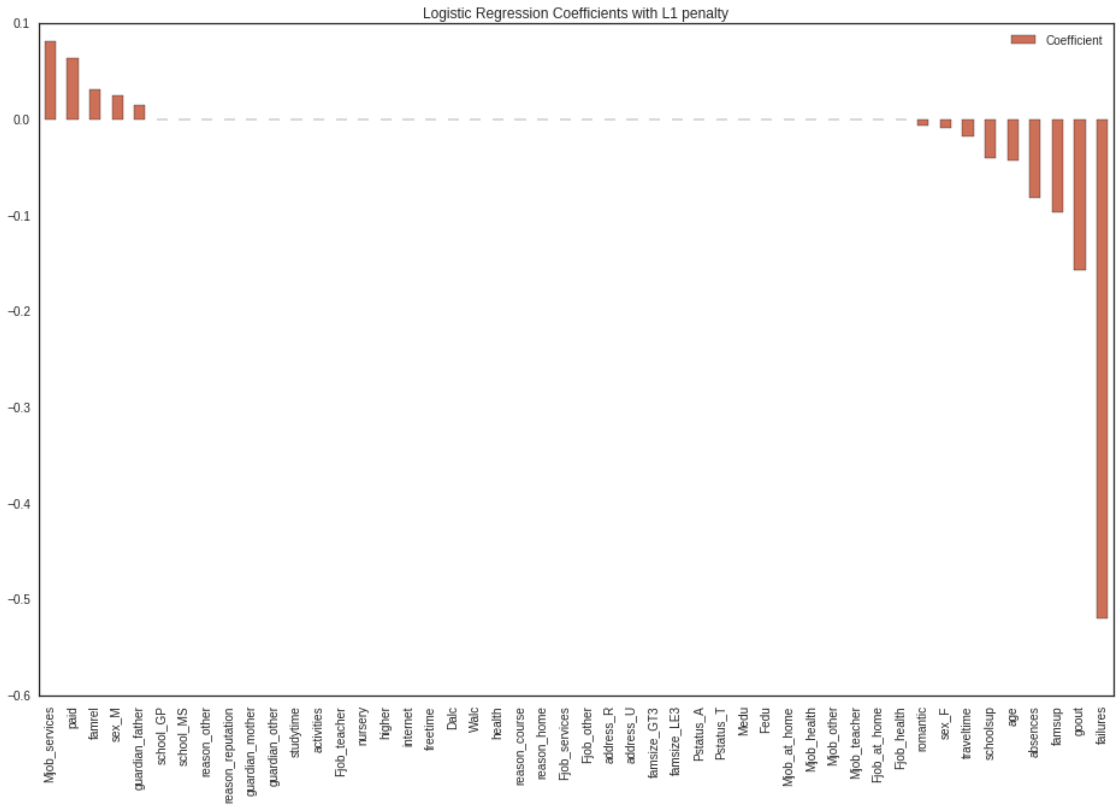
```
Best cross-validated tuned F1 score for Logistic Regression: 0.820130664244

Best tuned parameters for Logistic Regression:
```

Out[104]:

```
{'C': 0.1, 'penalty': 'l1'}
```

Out[80]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff909bb5110>
```



Here we can see that under the tuned parameters(C=1, penalty=L_1), many variables' coefficients were shrunk to 0, meaning they were unimportant in the final prediction.

Grid search reaffirmed our intuition that L_1 lasso penalty might work better than the L_2 default penalty.

'Failures', 'going out during the week', and 'absences' are still top (negative) variables agreeing for the most part with our Random Forest findings.

Out[76]:

	F1_test	F1_train	Predict_time_test	Predict_time_train	Training_time
0	0.791946	0.828633	0.000084	0.000146	0.001322

1.14 5. Choosing the Best Model

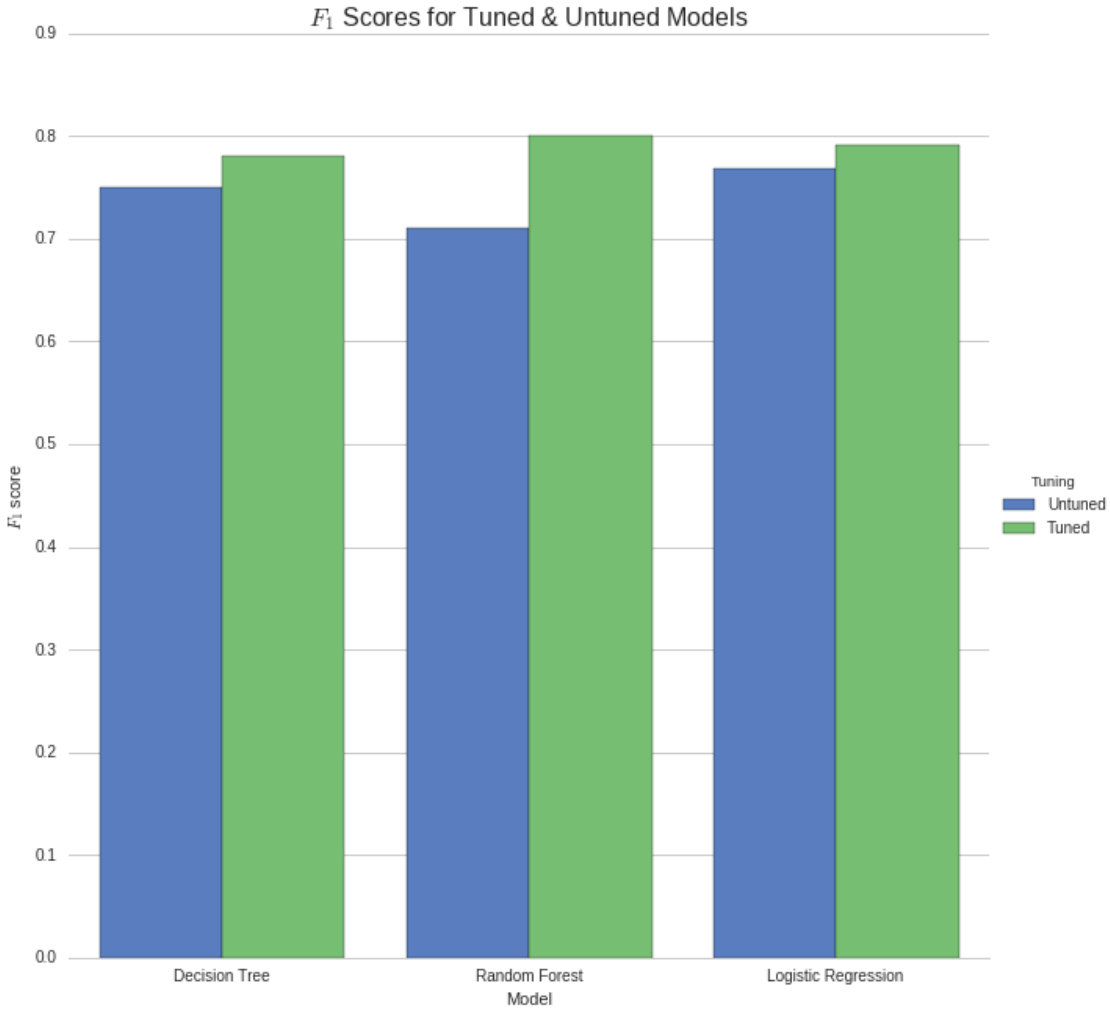
Out[74]:

	F1_test	F1_train	Model
Decision Tree	0.750000	1.000000	Decision Tree
Decision Tree	0.780142	0.813187	Decision Tree
Random Forest	0.711111	0.990148	Random Forest
Random Forest	0.800000	0.836518	Random Forest
Logistic Regression	0.768116	0.836028	Logistic Regression
Logistic Regression	0.791946	0.828633	Logistic Regression

	Predict_time_test	Predict_time_train	Training_size
Decision Tree	0.000118	0.000172	300
Decision Tree	0.000105	0.000156	NaN
Random Forest	0.000553	0.000771	300
Random Forest	0.001781	0.002373	NaN

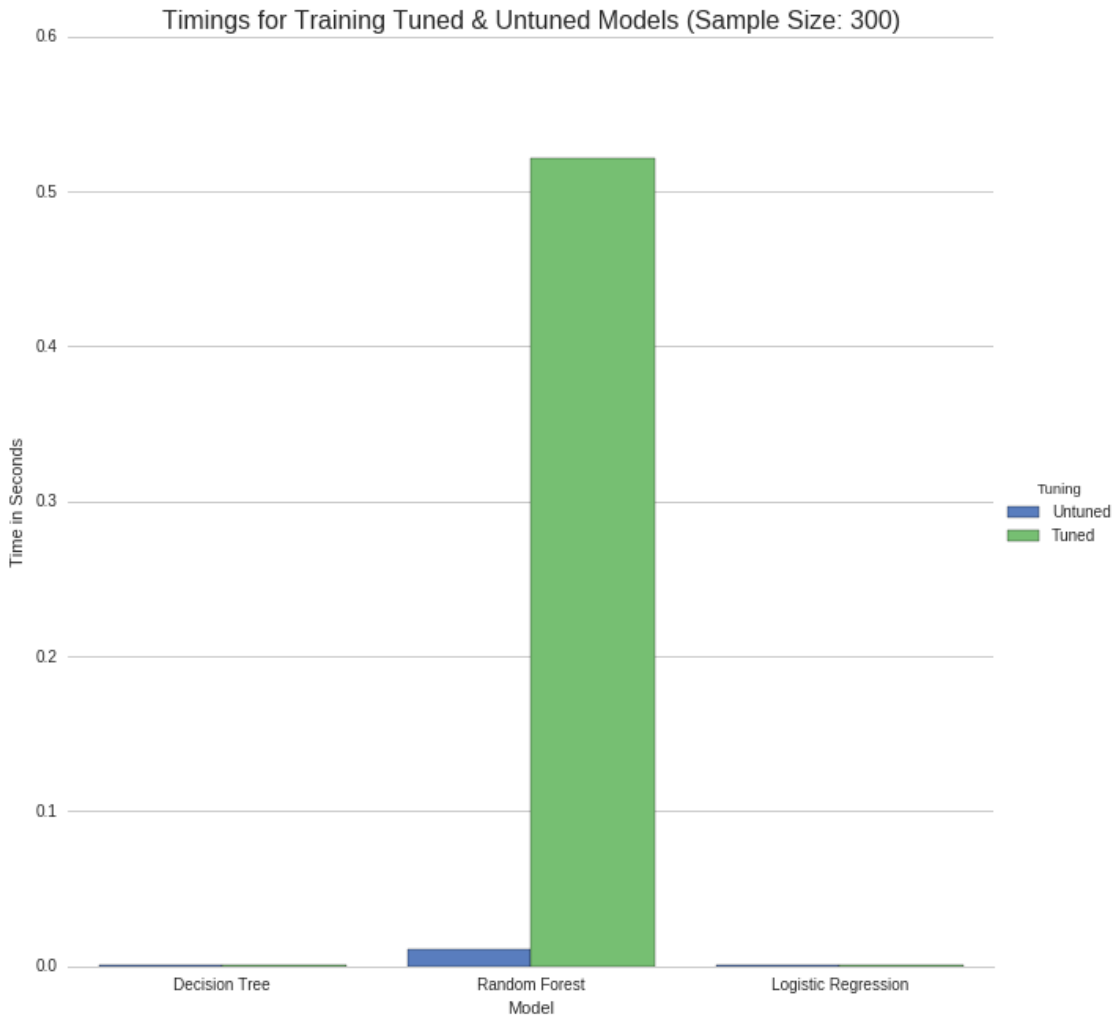
```
Logistic Regression          0.000064          0.000075          300
Logistic Regression          0.000092          0.001313          NaN
```

	Training_time	Tuning
Decision Tree	0.001352	Untuned
Decision Tree	0.000592	Tuned
Random Forest	0.011250	Untuned
Random Forest	0.110616	Tuned
Logistic Regression	0.001291	Untuned
Logistic Regression	0.000992	Tuned



When comparing across the models, Random Forests ended up at the highest F_1 score at 0.8. It also showed the biggest improvement post-grid-search.

All models did show at least a modest improvement in F_1 after grid search model tuning with Decision Tree ending up at 0.78 and Logistic Regression (L_1) ending up at 0.79 F_1 scores respectively.



Now here's the elephant in the room. Random Forests take far more computational time to train than either Decision Trees or Logistic Regression.

This particular tuned model uses 50 bagged estimators ensembled together.

Many times this can be justified by better accuracy/F_1 performance, but in our case, the gains may not be great enough to justify the costs given our dataset and its properties, and our specific resource limitations.

> Based on the experiments you performed earlier, in 1-2 paragraphs explain to the board of supervisors what single model you chose as the best model. Which model is generally the most appropriate based on the available data, limited resources, cost, and performance?

In this analysis, I found Logistic Regression (L_1) to be the best compromise between our constraints (computational resources used/performance/available data) and finding interpretable important actionable variables for identifying target students to increase graduation rate.

A future analysis *with more data* might use Logistic Regression/Random Forest/other models as a feature selector, then use Decision Trees to identify paths to target students.

But with this particular dataset size, this approach didn't yield any significant gains.

In 1-2 paragraphs explain to the board of supervisors in layman's terms how the final model chosen is supposed to work (for example if you chose a Decision Tree or Support Vector Machine, how does it make a prediction).

The final Logistic Regression model uses positive and negative weights to multiply against each variable. It then takes a linear combination of all of these variables and their weights and squashes that score between 0 and 1. This is then the probability of the student passing.

If the probability is over 50%, the model predicts that the student will pass, if not, it predicts the student will fail.

This particular model had the added benefit of shrinking many weights zero, effectively saying that their contribution to the overall score is zero. Thus this only gives us a sub-selection of the most important variables for identifying passing and failing students.

The final tuned F_1 score on the test set for our Logistic Regression (L_1) model was:

```
Out[113]: 0.79
```