UDACITY

PROJECT

Building a Student Intervention System

A part of the Machine Learning Engineer Nanodegree Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

CONGRATULATIONS!

Outstanding job with the report, I'm impressed with how well you've iterated on the project.

Applying machine learning often requires more than just fitting algorithms to data — business customers (such as the school board) also need to have results presented effectively, so it's great to see how well you've grasped the main points of the project and communicated them.

Congrats again, and keep up the good work with the next project!

Classification vs Regression

Student is able to correctly identify which type of prediction problem is required and provided reasonable justification.

For another look at how you could approach a Python data analysis, you can step through this example machine learning notebook.

Exploring the Data

Student response addresses the most important characteristics of the dataset and uses these characteristics to inform their decision making. Important characteristics must include:

- **Number of data points**
- **Number of features**
- **Number of graduates**
- **Number of non-graduates**
- **Graduation rate**

Great work getting the dataset stats!

Suggestion: look at imbalanced target classes
As you can see we have an imbalanced proportion of "yes" & "no" labels and will want to make sure the metric we're using for model evaluation is capturing how well the model is actually doing.

In this project we use the F1 score. As a benchmark for our final model, we can use the F1 score from predicting all "yes" values:

```
from sklearn.metrics import f1_score
print "\nF1 score for predicting all 'yes': {:.4f}".format(
    f1_score(y_true = ['yes']*n_passed + ['no']*n_failed, y_pred = ['yes']*n_students, pos_label='yes',
average='binary'))
```

```
F1 score for predicting all 'yes': 0.8030
```

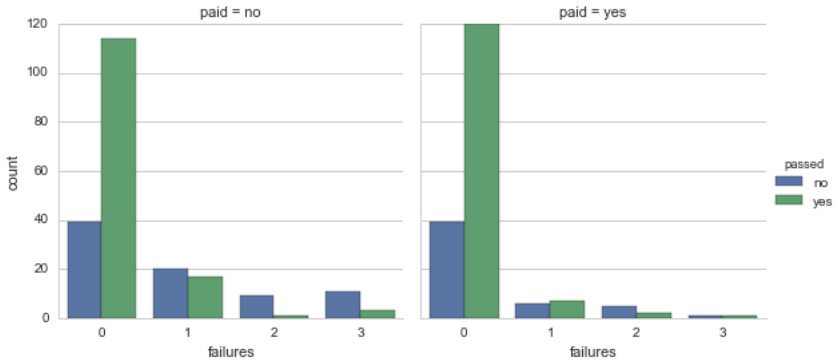
Preparing the Data

Code has been executed in the iPython notebook, with proper output and no errors.

Pro tip: exploratory analysis
Although the project template doesn't go into exploratory data analysis (EDA) that much, it's usually a big part of any data analysis project.

One nice method to explore is factor plots in seaborn. Below you can see an example of looking at counts of students who passed & failed, with a breakdown by the features "failures" and "paid"...

```
%matplotlib inline
import seaborn as sns
sns.factorplot("failures", col="paid", data=student_data, hue='passed', kind="count");
```



Training and test sets have been generated by randomly sampling the overall dataset.

Terrific job creating the train and test sets and using the `stratify` param to preserve the class imbalances across train and test sets. Kudos! 😎

For more thoughts on creating train and test (and validation) sets, check out quora to read about issues with splitting data, and selecting training data points.

Training and Evaluating Models

Three supervised models are chosen with reasonable justification. Pros and cons for the use of each model are provided, along with discussion of general applications for each model.

⭐ Great discussion of the models and why you chose them!

There are a multitude of issues to consider in choosing the best machine learning algorithm for your problem, and it's not always easy to know which model to use — with model selection it's often a good idea to try out simpler methods like Logistic Regression as a benchmark, and then move on to other approaches such as SVM, Decision Trees, and Ensemble methods.

Further reading:
You can also check out this guide from microsoft azure on choosing an algorithm, and this paper showing that rbf SVM and ensembles work best with binary classification.

All the required time and F1 scores for each model and training set sizes are provided within the chart given. The performance metrics are reasonable relative to other models measured.

We can also simplify the implementation a bit by using loops over the training sizes without creating new variables for them...

```
# loop thru models, then thru train sizes
for clf in [clf_A, clf_B, clf_C]:
    print "\n{}: \n".format(clf.__class__.__name__)
    for n in [100, 200, 300]:
        train_predict(clf, X_train[:n], y_train[:n], X_test, y_test)
```

Choosing the Best Model

Justification is provided for which model seems to be the best by comparing the computational cost and accuracy of each model.

Good justification of your choice of model by looking at factors such as the models' accuracy and computational cost/time!

Although model complexity and computational cost isn't that much of a factor here due to the small size of our dataset, it can become more important with larger datasets. Unlike traditional algorithms, most machine learning algorithms perform soft-computing — trying to find and approximate models to predict data.

In contrast to a sorting algorithm, for example, there is not necessarily a single fixed finite computational cost, and even Google is having to train engineers to put machine learning first.

Student is able to clearly and concisely describe how the optimal model works in laymen terms to someone what is not familiar with machine learning nor has a technical background.

⭐ Nice discussion of the Random Forest model in a way that's understandable by a non-technical audience. Well done!

For further ideas on describing common ML models in plain english, see here:
https://rayli.net/blog/data/top-10-data-mining-algorithms-in-plain-english

The final model chosen is correctly tuned using gridsearch with at least one parameter using at least three settings. If the model does not need any parameter tuning it is explicitly stated with reasonable justification.

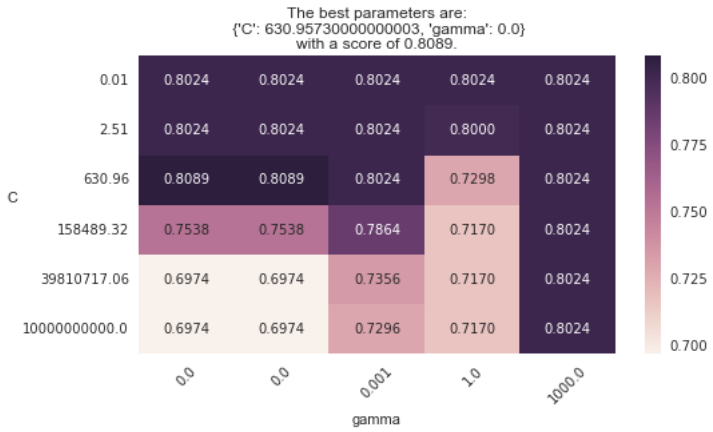
⭐ Excellent work here tuning the model, and it's also great that you outputted the best parameters found with `print grid_obj.best_params_`. Well done! 😎

Suggestion: Look at grid scores
1) You can also look at the scores of each of the parameter settings with `grid_obj.grid_scores_` ...

```
from IPython.display import display
display(pd.DataFrame(grid_obj.grid_scores_))
```

There you'll see that the grid search is performing a stratified K-Fold cross val with 3 folds as the default. If you want to play with this setting for the grid search, you can set the `cv` parameter on the `grid_obj` variable.

2) An interesting idea to experiment with is visualizing the results of a grid search. For example, with SVM tuning you could produce a heatmap of grid scores (below uses seaborn)...



The F1 score is provided from the tuned model and performs approximately as well or better than the default model chosen.

Nice work discussing the performance of the tuned model, and congrats on improving the F1 test score!

Pro tip: randomized search
For future reference, another common parameter tuning approach to try out is randomized search. With something like sklearn's `RandomizedSearchCV` method you can often get comparable results at a much lower run time.

Quality of Code

Code reflects the description in the documentation.

1) For other ideas on exploring scikit-learn, you can check out this list of 10 case studies.

2) And if you haven't been recommended Sebastian Raschka's Python ML book by another reviewer, it's definitely worth a look! 😎

DOWNLOAD PROJECT

Have a question about your review? Email us at review-support@udacity.com and include the link to this review.

RETURN TO PATH

Student FAQ