UDACITY

PROJECT

Building a Student Intervention System

A part of the Machine Learning Engineer Nanodegree Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Requires Changes

1 SPECIFICATION REQUIRES CHANGES

Overall this is an excellent submission, with solid code implementations and answers in most sections. You do have some adjustments to make in order to meet all the specs, but you're very close to completion. Keep at it! 😃

Classification vs Regression

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

Nice job "classifying" this classification problem. 😉

Let's hope that we can accurately predict which students pass vs fail, because there's good evidence that student interventions actually work.

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

Required: adjust stats
You got most of these, but the number of features in `n_features` should not include our target column.

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_student
s, 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: try pandas groupby() method
You might also want to look into using pandas groupby function — for example, you can use it to see summary statistics on the passing and failing students separately...

```
passed_groupby = student_data.groupby('passed')
passed_groupby.describe()
```

See more on pandas features here:
http://dataconomy.com/14-best-python-pandas-features/

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

Terrific job creating the train and test sets, and setting a random state to make your results reproducible by others!

Pro tip: stratify the data
Given the imbalanced proportion of passes to fails, we want to ensure that our training and test data both have similar proportions of "yes" and "no" labels.

To preserve the class imbalance, we can use the "stratify" parameter in `train_test_split`...

```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, stratify=y_all, te
st_size=0.24, random_state=42)

print "Grad rates:\nTrain set = {:.2f}%".format(100 * (y_train == 'yes').mean())
print "Test set = {:.2f}%".format(100 * (y_test == 'yes').mean())
```

```
Grad rates:
Train set = 67.00%
Test set = 67.37%
```

Note that stratifying will maintain the class imbalance across both sets.

In contrast, your current train/test splits show the following class imbalances...

```
Train set = 68.33%
Test set = 63.16%
```

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 — 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.

(1) **Small data**
A big issue when justifying our model choice is the small amount of data. This should make training time with more complex models like SVM less of an issue, and we should be well served with your selection of a high bias classifier like Logistic Regression.

(2) **Interpretability**
Model interpretability is another important factor to consider, and it will be nice knowing that Logistic Regression can produce a model which can be interpreted by the school board and reveal factors that are highly predictive of student performance.

(3) **Accuracy**
Lastly, in order to achieve highly predictive results in the first place though, it's likely we'll need a non-linear classifier, and your approaches Gradient boosting and SVM's may be effective here. Let's hope they work well!

Further reading:
You can also check out this list of Machine Learning cheat sheets for more info on using the models, 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.

Excellent work generating the results and setting random states to make them reproducible!

We can also simplify the implementation a bit more by using loops directly over the training sizes...

```
# 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 model choice by looking at factors such as the models' accuracy and computational cost/time!

Suggestion: Use a benchmark
To get an idea of how good our scores are, another thing we can do is compare the results of the models with a benchmark of predicting all "yes" values on the test set...

```
from sklearn.metrics import f1_score
print "F1 score for predicting all \"yes\" on test set: {:.4f}".format(
    f1_score(y_test, ['yes']*len(y_test), pos_label='yes', average='binary'))
```

```
F1 score for predicting all "yes" on test set: 0.7742
```

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 model in a way that's understandable by a non-technical audience. Well done!

If you wanted, you could also provide additional thoughts on the "logistic" function that gives the model its name. For example...

- Logistic Regression takes features about previous students (eg, their age, gender, etc) and creates a model that assigns a "weight" to these features that assess their importance in determining whether a student passed or failed.
- When we want to predict an outcome for a new student , we take the new student's features and combine them with the feature weights to create a single summed up value.
- A final summed up value is applied to a function (called a "sigmoid") that then predicts the probability that a student will pass or not.

https://www.quora.com/What-is-logistic-regression

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.

Great job tuning the model!

Suggestion: show best parameters found

After the grid search, you could also output the best parameters found using something like `grid_obj.best_params_` or simply `print clf`.

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, ... although it's too bad we didn't improve the F1 test score.

Suggestion: scale the features

Feature scaling can often have a big impact on the performance of machine learning algorithms. You might also experiment with trying to improve the F1 even more by first normalizing the features...

```
from sklearn.preprocessing import Normalizer
normer = Normalizer()
X_train = normer.fit_transform(X_train)
X_test = normer.transform(X_test)

parameters = {
    'class_weight':[None, 'balanced'],
    'penalty':['l1', 'l2'],
    'C': np.logspace(-3,2,31)
    }

# The default grid search uses 3 folds; use the 'cv' param to change this
grid_obj = GridSearchCV(clf, parameters, cv=4,
                        scoring=f1_scorer, verbose=1,
                        n_jobs=16, pre_dispatch='2*n_jobs')
```

Here's more reading on scaling and normalizing.

Quality of Code

Code reflects the description in the documentation.

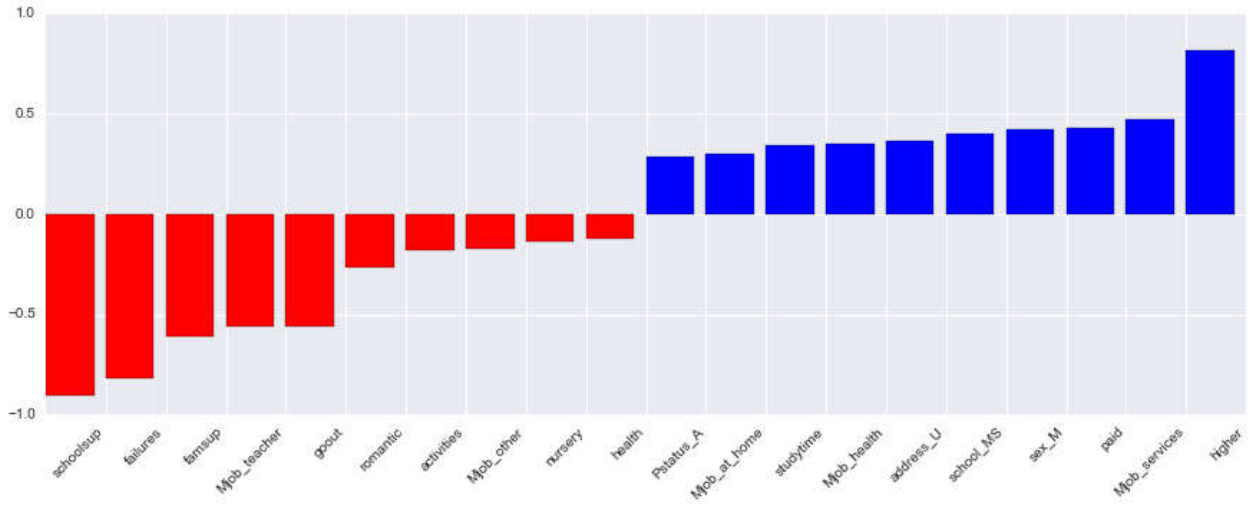
Pro tip: look at coefficients

You can also visualize the important coefficients in the logistic classifier with something like this...

```
import matplotlib.pyplot as plt
%matplotlib inline

# look at top 10 features
coef = clf.coef_.ravel()  ## use the variable name for the Logistic classifier
pos_coef = np.argsort(coef)[-10:]
neg_coef = np.argsort(coef)[:10]
look_coef = np.hstack([neg_coef, pos_coef])

# plot the coeffs
plt.figure(figsize=(15, 5))
colors = ["red" if c < 0 else "blue" for c in coef[look_coef]]
plt.bar(np.arange(20), coef[look_coef], color=colors)
feats = np.array(X_all.columns)
plt.xticks(np.arange(1, 21), feats[look_coef], rotation=45, ha="right");
```



(**Note**: Above only shown as an example, not generated with your submitted notebook)

RESUBMIT

DOWNLOAD PROJECT

Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

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