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

2 SPECIFICATIONS REQUIRE CHANGES

Dear student,

well done improving your submission, there are still a couple of issues regarding logistic regression: Please refer to my comments in the pros and cons section and in the algorithm in layman's terms explanation section for more hints. It is extremely important to be aware of the pros and cons of the algorithm and have a sound and clear knowledge of the way the chosen algorithm works.

You're almost there, keep up your good work!

Classification vs Regression

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

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

Preparing the Data

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

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

Pro Tip: Data assessment:
When dealing with the new data set it is good practice to assess its specific characteristics and implement the cross validation technique tailored on those very characteristics, in our case there are two main elements:

1. Our dataset is **small**.

2. Our dataset is slightly **unbalanced**. (There are more passing students than on passing students)

We could take advantage of K-fold cross validation to exploit small data sets. Even though in this case it might not be necessary, should we have to deal with heavily unbalance datasets, we could address the unbalanced nature of our data set using Stratified K-Fold and Stratified Shuffle Split Cross validation, as stratification is preserving the preserving the percentage of samples for each class.
http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.StratifiedShuffleSplit.html
http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.StratifiedKFold.html
As for the initial train test split you can obtain stratification by simply using **stratify = y_all**:

```
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, test_size=95, random_state=42)
```

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.

There is an issue with the cons and the reason for choosing logistic regression, you write: "However if there isn't a high correlation between the observed and predicted variables, then the LR model can add little predictive value. I chose this model because there is a potential linear correlation and may provide a baseline for other models to compare to." I'm not sure that logistic regression requires correlation between variables and the predicted variable, could you please provide some references regarding this? (You might be right though I can't really understand the rationale behind it, please point at some links or references so that we can verify that)
Please discuss more detail as well the rationale for choosing logistic regression, what do you mean by 'potential linear correlation'? Could you provide some statistics to ground your statement? Why should there be a linear correlation in order to use logistic regression?

http://www.statisticssolutions.com/assumptions-of-logistic-regression/

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.

Optional: You could use a loop to go through the algorithms:

```
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.

Tip: Scalability, developing a proactive attitude
An interesting question you might address to further improve your answer, and show some business-oriented proactivity by anticipating the customer's needs, regards the scalability of your chosen algorithm: What would happen if you had to classify thousands or hundreds of thousands of students? What would happen with training time and with prediction time? Would you still choose the same algorithm? (Please note that this is relevant as not every algorithm scales linearly, some might scale exponentially, which means that the computational costs might grow exponentially with size. This is not an issue as for now but it would be if we were considering a much bigger number of students).
https://github.com/jeff1evesque/machine-learning/wiki/Algorithm:-Big-O-Notation

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.

The explanation is a bit too vague, by reading the provided description of the algorithm I'm not fully able to understand specifically how it actually works. You've managed to discuss the algorithm in a simple way though some fundamental elements regarding the way towards should be included in your discussion as well. You could try explaining the logic of the algorithm through everyday examples and metaphors and/or by including some plots that might help the audience understand the algorithm. Please try to discuss each steps that the algorithm takes in order to get to the final logistic function and briefly discuss in a simple way what the logistic function is and why it is used.

Here is an example regarding decision trees that might be helpful in providing you with a blueprint of what is expected:

1. A tree is characterized by a set of input variables (e.g. sex, age, etc.) and a set of outcomes in our case is either 'passed' or not 'passed'. The set of input variables are all the 30 "features" we have.
2. The tree is then "learned" by splitting the dataset set into subsets during training time. Different algorithms split the data into different subsets according to some criteria, trying to achieve the highest homogeneity or purity in the child nodes. The splitting is stopped when the subset has the same output variable or no further subset can yield any useful predictions.
3. Prediction..

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.

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

Pro Tip:
We can use a stratified shuffle split data-split which preserves the percentage of samples for each class and combines it with cross validation. This could be extremely useful when the dataset is strongly imbalanced towards one of the two target labels.

http://scikit-learn.org/stable/modules/generated/sklearn.cross_validation.StratifiedShuffleSplit.html

In this example a support vector classifier is used, any other classifier would do, please remind to change the parameters adapting them to the specific classifier you intend to deploy.

```
from sklearn.grid_search import GridSearchCV
from sklearn.svm import SVC
from sklearn.cross_validation import StratifiedShuffleSplit
from sklearn.metrics import f1_score
from sklearn.metrics import make_scorer
f1_scorer = make_scorer(f1_score, pos_label="yes")
parameters = { 'C' : [ 0.1, 1, 10, 100, 1000 ], 'gamma' : [ 0.0001, 0.001, 0.1, 10, 100 ] } # Some SVC parameters
ssscv = StratifiedShuffleSplit( y_train, n_iter=10, test_size=0.1) # 1. Let's build a stratified shuffle object
grid = GridSearchCV( SVC(), parameters, cv = ssscv , scoring=f1_scorer) # 2. Let's now we pass the object and the par
ameters to grid search
```

```
grid.fit( X_train, y_train ) # 3. Let's fit it
best = grid.best_estimator_ # 4. Let's reteieve the best estimator found
y_pred = best.predict( X_test ) # 5. Let's make predictions!
print "F1 score: {}".format( f1_score( y_test, y_pred, pos_label = 'yes' ))
print "Best params: {}".format( grid.best_params_ )
```

Quality of Code

Code reflects the description in the documentation.

RESUBMIT

DOWNLOAD PROJECT

Learn the best practices for revising and resubmitting your 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