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

Excellent submission here and nice adjustment with your algorithm description. Hopefully going back here was very beneficial in understanding how logistic regression works. You clearly have a solid grasp on these techniques. Wish you the best of luck throughout this program!!

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**

All correct here. It is definitely a good idea to get an idea of the distribution of our target variable here. As we now know that we have an imbalance of students, so we can plan accordingly. Also we know that if we were evaluating based on accuracy, our "dumb" classifier should predict 67%.

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.

With using an unbalanced dataset like this one with only 395 rows, and about 33% of students failing and 67% passing it is a great idea to use train_test_split's stratify argument to make sure the labels are evenly split between the training and testing dataset. Try this

```
X_train, X_test, y_train, y_test = train_test_split(X_all, y_all, train_size=300, stratify = y_all, random_state=42)

# see the distribution of labels
print np.mean(y_train == 'no')
print np.mean(y_test == 'no')
```

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.

Nice adjustment here in terms of your logistic regression with the pros and cons of the model. Another really cool thing would be to check out the BigO Notation of the different algorithms

Logistic Regression:

- For logistic regression you have to solve an optimization problem, and this depends on the optimizer. With L-BFGS an iteration takes linear time on the size of the training set and I've never needed more than 100 iterations to converge to a reasonable value (but I'm not aware of any worst-case bounds).

Support Vector Machine:

- The standard SVM training has O(n^3) time and O(n^2) space complexities, where n is the training set size. But recent approaches are inverse in the size of the training set.

Decision Tree:

- For an unpruned decision tree is O(mtry * n log(n)), where n is the number of objects and mtry is the number of attributes,

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.

Great job with your double for loop

```
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.

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 job with your description of a logistic regression, as this would be good for someone who is not familiar with machine learning nor has a technical background with the use of "*seperated into two 'regions', one for each class. Just like putting a cardboard into a box that seperates the red balls and blue balls inside the box*".

To go into a bit more detail in terms of how logistic regression works(might be a bit advanced for non-technical, but for your sake)

- After the initial training using the training data Logistic Regression gives each feature a weight by minimizing a cost function.
- When we are predicting we take all feature values multiply them with their weights accordingly
- We then sum up all the results.
- This final result is applied to a special function called Sigmoid/Logistic function which only outputs values between 0-1 which can be interpreted as the probability of our positive label.

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 work here with your Logistic Regression tuning.

- Typically another really good idea when optimizing a Logistic Regression classifier would be to run feature scaling before hand to prevent some

features from overpowering others. Therefore could check out using StandardScaler.
- Another more advanced idea would be to use something like pipelining to combine gridSearch and features scaling. Therefore check out this blog post for some cool ideas (https://civisanalytics.com/blog/data-science/2016/01/06/workflows-python-using-pipeline-gridsearchcv-for-compact-code/)

Note: Based on this high computation time of running your GridSearch algorithm. As one limitation of GridSearch is that it can be very computationally expensive when dealing with a large number of different hyperparameter and much bigger datasets. Therefore if we are limited on time, as we are here we could also use

- RandomizedSearchCV which can sample a given number of candidates from a parameter space with a specified distribution. Which performs surprisingly well!

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

Good job running your chosen params on the desired data.

Quality of Code

Code reflects the description in the documentation.

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