UDACITY

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Requires Changes

7 SPECIFICATIONS REQUIRE CHANGES

Overall, good work on those parts of the project you completed. Hope this review helps you to continue your work on those sections not meeting specifications. However if you still have questions/comments, please don't hesitate to reach us, we'll be glad to help you.

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.

Well done recognizing this is a classification problem.

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

Well done getting the numbers, however note the dataframe columns also include the label, so in order to calculate the total number of features you need to subtract the label column from the final result.

```
: # TODO: Compute desired values - replace each '?' with an appropriate expression/function call

n_students = len(student_data.index)
n_features = len(student_data.columns)
n_passed = sum([1 for y in student_data['passed'] if y == 'yes'])
n_failed = sum([1 for n in student_data['passed'] if n == 'no'])
grad_rate = 100.*n_passed/(n_passed + n_failed)

print "Total number of students: {}".format(n_students)
print "Number of students who passed: {}".format(n_passed)
print "Number of students who failed: {}".format(n_failed)
print "Number of features: {}".format(n_features)
print "Graduation rate of the class: {:.2f}%".format(grad_rate)
```

```
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: 67.09%
```

Preparing the Data

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

Your code perfectly works. As a side comment, I think this is an excellent reference. Note IPython notebooks are a great tool and markdown notation is becoming more and more popular, so it is definitely worth to invest some time to learn more about it.

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

Well done using train test split to split your data, this is a simple and easy to use function to create the different sets.

As a side comment, note that because the data set is very small, the results of the different tests can vary a bit depending on how the data is split. It goes beyond the scope of the project, but a more thorough way to evaluate the different models would involve running the code multiple times (using different `random_state` values) to see the effect of different splits.

Training and Evaluating Models

The pros and cons of application for each model is provided with reasonable justification why each model was chosen to explore.

This part isn't quite meeting specifications yet because pros and cons of the classifiers weren't discussed. In this section it is requested you include a table or similar where it is addressed pros/cons of the classifiers attempted.

For your reference, Scikit includes great documentation for most of these classifiers, for example: SVM or GaussianNB or Random Forest. Of course more information can be found in internet after a quick search, for example this Quora post.

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 F_1 score. Repeat this process with different training set sizes (100, 200, 300), keeping test set constant.

Produce a table showing training time, prediction time, F_1 score on training set and F_1 score on test set, for each training set size.

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

As a side comment, although you included your reasons to use these classifiers, it would be great if you also include reasons based on the classifier's main characteristics, to use them in this particular problem, for example: why DT is particularly interesting in this problem?

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.

Well done using F1 (harmonic mean of precision and recall) to evaluate your algorithms results and including more training set sizes in your tables. As a side note, to explain your final results to your audience it is always better use precision (of those selected for intervention, how many really need intervention?) and recall (of those who need intervention, how many of them were identified?) since these scores are easier to interpret.

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.

Although you included a good explanation to decide which model seems to be the best, it is part of the rubric to perform the model selection before the tuning phase.

5. Choosing the Best Model

- 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 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).
- Fine-tune the model. Use Gridsearch with at least one important parameter tuned and with at least 3 settings. Use the entire training set for this.
- What is the model's final F_1 score?

What is requested here is to clearly compare and contrast the time (training and test) and F1 scores between the three models chosen. Note in the project it is defined: *"(...)with limited resources and budgets, the board of supervisors wants you to find the most effective model with the least amount of computation costs (you pay the company by the memory and CPU time you use on their servers)(...)"* .

So, basically you need to identify which is the most appropriated classifier in this case in terms of computational cost (time) and performance (F1 over test set).

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.

Although explanations used non-technical language, which is great!, and there is a deep explanations on how the model handle the different student's characteristics, I miss some more information on how the model decides whether a student needs intervention or not. According to your description it is its proximity to another student to decide, however the algorithm is not using a single student (except in the case where `n_neighbors=1`) but the results of `k` neighbors. So please include some more information to clarify this point.

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.

Well done tuning your SearchGridCV object to optimize `f1` score, however the reason for marking this section off is because the entire dataset is used during the fitting phase. Note it is important to maintain some data completely unseen for validation, using the entire dataset is going to produce biased results during the validation phase since there is not unseen data anymore.

```
#Perform grid Search
def gridIt(clf, params):
    grid_clf = grid_search.GridSearchCV(clf, params, scorer)
    print clf.__class__.__name__
    print "Grid search time:", timeTraining(grid_clf, X_all, y_all)
    print "Parameters of tuned model: ", grid_clf.best_params_
    y_pred, predict_t = predictAndTime(grid_clf, X_test)
    print "f1_score and prediction time on X_test, y_test: "
    print F1(y_test, y_pred), predict_t
    print '------------------\n'
```

```
# Return the classifier's training time
def timeTraining(clf, X_train, y_train):
    start = time.time()
    clf.fit(X_train, y_train)
    end = time.time()
    return "{:.3f} s".format(end - start)
```

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

See comments above.

Quality of Code

Code reflects the description in the documentation.

The reason to mark this section off is because in your submission you need to include a `pdf` report with your responses. Yes I know, since a notebook is used, everything should be included there, however since many students are not very confortable working entirely with notebooks, we decided to include the `pdf` report. So please in your next submission include it also.

Deliverable

iPython notebook with all the steps marked with TODOs completed, and code blocks executed, showing desired outputs.
Project report in PDF format (3-5 pages), answering the following questions.

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

Rate this review