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

4 SPECIFICATIONS REQUIRE CHANGES

Awesome job with your submission, as you have great intuition in these techniques. Just have a few minor issues here and you will be good to go. Keep up the great work!!

Classification vs Regression

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

Correct with "*Regression is used to predict continuous values. Classification is used to predict discrete value.*"

This is definitely a classification problem. To be even more thorough here it is a binary classification problem where we have the column `passed` containing yes and no.

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**

Nice use of pandas column slicing. However relook at `Graduation rate of the class: 0.67%`. As this would be a very difficult school with less that a one percent graduation rate. Therefore simple fix and just multiply this by 100.

Preparing the Data

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

Everything runs great!

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

Great work here with Sklearn's `train_test_split` as this method makes it much easier to split your data.

Note: Also great work setting a `random_state` here as well with your splitting function, as this allows for reproducible split of the data, thus your training and testing samples are not randomly generated each time you run this.

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.

You are very close here. Therefore please also include at least one weakness for your Logistic Regression classifier.

Logistic Regression:

- **REQUIRED**: Please list at least one weakness for your Logistic Regression model
- Correct that this model "*robust to noise can avoid overfitting*" since this is a linear classification
- Great for probabilistic outputs
- Works based on the sigmoid function

Support Vector Machine:

- Typically much slower, but the kernel trick makes it very awesome to find non-linearity in the data.
- Very memory efficient
- Also note here the SVM output parameter are not really interpretable.

Decision Tree:

- Probably the best part is the interpretability of the model, nice job!!
- We can definitely see here that our Decision Tree has an overfitting issue here and this is typical with Decision Trees and Random Forests.
- Another great thing that a Decision Tree in sklearn gives us is a feature importance

(Optional): Would also recommend expanding on your pros/cons. As it would be beneficial to know more of these even outside this project.

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.

Nice work running your code with the different desired algorithms and your outputs look reasonable.

The reason this is marked as *Requires Changes* is that you should also set a random_state for each model you use, if provided. Therefore you should include random_states in your `LogisticRegression()`, `DecisionTreeClassifier()` and `SVC()` for reproducible results.

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.

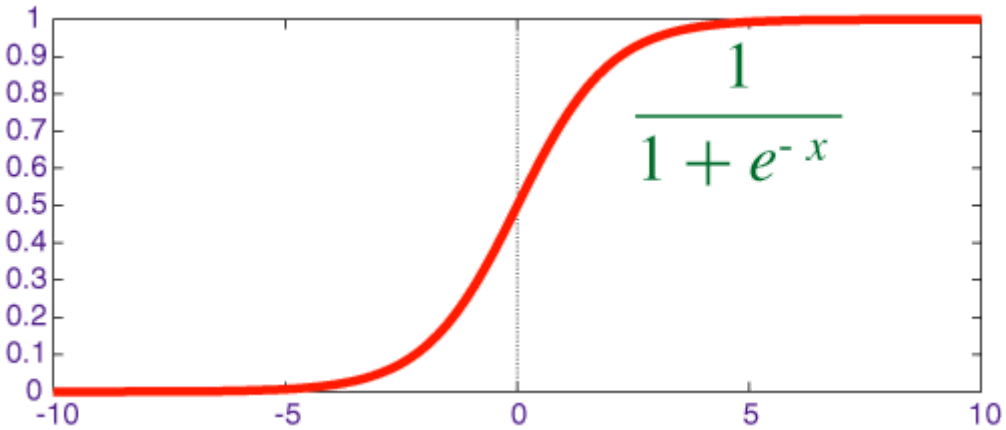
Great justification for your choice in your Logistic Regression, as you have done a great job comparing these all in terms of predictive power and computational expense. This classifier seems to have the best of both with high speed and testing accuracy. Good choice!!

Something to think about:

- Would you still choose this algorithm if the dataset size increased 100 times?

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.

You are off to a good start here with your model description, as this would definitely be good for someone who is not familiar with machine learning nor has a technical background. However for this section you should probably discuss the main thing based on how a LogisticRegression classifier works(sigmoid function(to get probabilities))



(Optional): For this section it would also be very beneficial to expand on how LogisticRegression actually makes a prediction. What is this "*student behavior indicates that he/she is likely to pass or predict fail*". Therefore maybe a more straight forward way to explain the algorithm is with a concrete example(which you have started doing). For example, If I were to use and describe a Support Vector Machine, I might say that many students who pass have parents who went to college, live near the school, don't have full-time jobs, etc (I'm making this up). A typical student who fails will have a full-time job, etc. The SVM algorithm will draw curves around these groups to separate them. When a new student comes along, SVM will see which side the new student falls on and will make a pass/fail 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.

Great use of GridSearch here and the hyper-parameter of `C` is definitely the most tuned hyper-parameters for an Logistic Regression. As the C parameter refers to the

- Regularization is applying a penalty to increasing the magnitude of parameter values in order to reduce overfitting. When you train a model such as a logistic regression model, you are choosing parameters that give you the best fit to the data. This means minimizing the error between what the model predicts for your dependent variable given your data compared to what your dependent variable actually is.

The problem comes when you have a lot of parameters (a lot of independent variables) but not too much data. In this case, the model will often tailor the parameter values to idiosyncrasies in your data -- which means it fits your data almost perfectly. However because those idiosyncrasies don't appear in future data you see, your model predicts poorly.

Could also simply do this

```
parameters = [{'penalty': ['l1', 'l2'],'C':param_range, 'tol':param_range}]
```

Pro Tip: As 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 sklearn's StratifiedShuffleSplit to make sure the labels are evenly split between the validation sets.

```
from sklearn.cross_validation import StratifiedShuffleSplit
cv = StratifiedShuffleSplit(y_train)
grid_obj = GridSearchCV(estimator=clf, param_grid=parameters, scoring=f1_scorer, cv=cv, n_jobs=2)
```

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

Nice job comparing your tuned model to your untuned. We could also examine the confusion matrix

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
%matplotlib inline
pred = clf.predict(X_test)
sns.heatmap(confusion_matrix(y_test, pred), annot = True, fmt = '')
```

Quality of Code

Code reflects the description in the documentation.

Everything looks and runs good, just be wary of code reuse. It you ever are coping and pasting code, it should have its own function. Therefore you could have for loops for your different model runs.

```
for size in [100, 200, 300]:
    train_predict(clf_A, X_train[:size], y_train[:size], X_test, y_test)
```

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

Rate this review

Student FAQ