UDACITY

PROJECT

Finding Donors for CharityML

A part of the Machine Learning Engineer Nanodegree Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Requires Changes

1 SPECIFICATION REQUIRES CHANGES

Dear student,

This is good work.

Please see my comments and notes.
Please also let me know if you've found my review helpful or not. I am always interested to improve my reviews, so please let me know how I did for you.
I directly receive your feedback from your star rating and comments about the review

thanks
-Gilad

Exploring the Data

Student's implementation correctly calculates the following:

- **Number of records**
- **Number of individuals with income >$50,000**

- **Number of individuals with income <=$50,000**
- **Percentage of individuals with income > $50,000**

So you missed one small detail here which is that your percentage is getting rounded down by python. This can be fixed by doing something like this

```
greater_percent = 100*float(n_greater_50k)/n_records
```

if you add do that, you cast one number as a float and then python won't round your math down. This is no longer needed in python 3 fyi.

Preparing the Data

Student correctly implements one-hot encoding for the feature and income data.

Evaluating Model Performance

Student correctly calculates the benchmark score of the naive predictor for both accuracy and F1 scores.

The pros and cons or application for each model is provided with reasonable justification why each model was chosen to be explored.

Please list all the references you use while listing out your pros and cons.

This is a good discussion, you go into each algorithm and demonstrate a good understanding of what assumptions they are making, what pro's and con's exist.

A couple of small points

> Our data consists of some numerical but many categorical features

this is not so true after the OHE, we have mostly categorical features.

> The training time shouldn't be an issue, as the dataset is not too large.

Also, it's not much of an issue because of the nature of the problem. It's mailing out letters to find donors.

Some tips for writing up summaries of algorithms

Don't editorialize:

ADJECTIVES IMPLY OPINIONS

- very fast
- highly susceptible
- very simple
- performs poorly
- performs very well
- when the data is large (large is a relative term, what *is* large actually? 100? 1000? 1 million?)
- when the data is small (see above)

These are all essentially opinions of the writer. They aren't substantiated as facts, they are just what someone believes or thinks.

Instead it's much better to write with quantitative and objective prose.

Quantitative writing

PRESENT ONLY THE *FACTS*

- the algorithm increases with exponential time
- is able to capture non linear boundaries
- is restricted to linear separations
- makes the assumption that the underlying data is normally distributed
- if the data is > 300k samples ...
- if the data is < 1000 samples ....

When we write objectively without adjectives, we let the reader make their *own* conclusions about the algorithms (or our research). This also causes the reader to trust your expertise much more, because they aren't being asked to simply believe what you are saying. The reader will think for themselves, and draw their own conclusions. Instead of deciding whether or not they believe your narrative, they will be grateful for the information you have provided them. It shifts the focus from "do I trust this writer?" to "this writer has given me useful information" and "this writer is knowledgable about the subject"

These tips will aid all your writing.

Student successfully implements a pipeline in code that will train and predict on the supervised learning algorithm given.

Student correctly implements three supervised learning models and produces a performance visualization.

A note on random states

The `random_state` parameter of many algorithms (classifiers / models) in sklearn are used to "fix" your model, so it can be reproduced over and over. Sometimes models or methods, require an element of randomness. For example shuffling and splitting the data is required to be random. However, since the results change ever so slightly each time you re-run the method -- it can be frustrating (scores will change slightly). For this reason Udacity has in the past required students to set the `random_state` variable in the methods.

By assigning a `random_state = 1` variable in the argument to the method -- we *actually remove all randomness* from the method. In essence, the algorithm becomes deterministic (fixed) and the results will always be the same.

I get it for "double checking" someone elses implementation. But I don't like them in general. Why?
Because it creates a slice of the model, which isn't representative of the real model. The real model has randomized parameters, so it should randomly change every time you instantiate it. This means that typically to get real results, you need to run the model multiple time (hundreds) in order to know *in general* how it will perform.
When the difference between two models is <1%, can we be sure one model is better than the other? Especially when we've randomly created a static version of that model by freezing it's random variables? It's not really possible. So I would advise that the better solution would be to run the experiment (with randomness) a few hundred times and then average the results. This would be a much better indicator of what kind of performance you can get.
ShuffleSplit is a great method to do this kind of experiment.

Keep in mind I offer this as advice for your future work, using a random_state here is absolutely fine, but I always like to teach best practices when I can offer them, and I believe this to be one.

Improving Results

Justification is provided for which model appears to be the best to use given computational cost, model performance, and the characteristics of the data.

I agree with you here entirely. At the end of the day most often, we simply want the highest performing algorithm in terms of the error metric. This is why it's so critical to choose an error metric that accurately represents the problem which you are trying to solve. In this notebook we've chosen f-beta with the value 0.5, and the best algorithm should be the one that does best on that metric.
Training and prediction times are often minor considerations. There are cases where this matters (real-time scenarios, but in my experience it's not the general case). So for this problem, since we probably only need to make predictions once in a while (maybe once a day) and can train separately (to update with new data), I wouldn't worry about training or prediction times.

The requirement of the company in this case is mailing out letters. It's not a real-time scenario. It's certainly going to be done on a regularly scheduled time-frame. Even if we had to train for 2

hours or 2 days, I'd prefer to train as long as needed in order to get the best possible results from our model.

Student is able to clearly and concisely describe how the optimal model works in layman's terms to someone who is not familiar with machine learning nor has a technical background.

This is an OK start, but you need to make it a lot simpler and easier to understand.

Laymans terms refers to terms which your grandmother could understand. Your grandfather, or your children. Your wife or husband.
So while I actually really like your explanation here, you need to go higher level in order to meet specifications. The idea here is to learn how to explain things to your boss or co-workers.
You may not realize it, but you've learned a lot of jargon so far in the MLND! You know what iterative, weak / stronger learners, models etc etc are. However Your co-workers boss and mother do not.
So go even higher level and talk about this model in the highest possible terms.
Try to think of examples that children could understand.

The reason it's important to do this is because often at work, you will be asked to explain how your models work. Being able to offer some intuitive understanding goes a long way with colleagues respect.

This is actually a very difficult example so don't get frustrated. Feel free to google "intuitive understanding of X model" for inspiration.

A few points about what you have written so far

- boosting can use any learner, provided it is a weak learner. It's typically done on trees, but not required
- you should mention that they are weak learners
- you should mention that it uses some method (the gradient) to iteratively focus on the problems that it did worst. It focus the new trees on the sections it did badly last time.

At the worst case you can just show people this great comic
https://xkcd.com/1838/
No seriously, you can't do that, sorry -- I'd get in trouble if you did. 😂

The final model chosen is correctly tuned using grid search 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 checking parameters. Often times we may feel this part of the work is tiresome, but by fine tuning the parameters to fit the data better, we can boost performance considerably.

Also note you can get the final parameters of the model from gridsearchCV by accessing it's `best_params_` attribute.
see the docs here
http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

gridsearch N-estimators

I wouldn't be surprised if `n_estimators` will always choose the highest value in gridsearch. more estimators will always yield a higher score, so gridsearch will always choose the highest. The trade-off is in computation time, which gridsearch doesn't evaluate.

sometimes I've seen n-estimators selected at a lower than maximum amount, I think this happens with ensembles, perhaps to keep the overall ensemble weaker (and avoid overfitting).

Student reports the accuracy and F1 score of the optimized, unoptimized, and benchmark models correctly in the table provided. Student compares the final model results to previous results obtained.

Feature Importance

Student ranks five features which they believe to be the most relevant for predicting an individual's' income. Discussion is provided for why these features were chosen.

Nice discussion. It's actually quite hard to know what will be important. Your thought process is lucid and clear.

Student correctly implements a supervised learning model that makes use of the `feature_importances_` attribute. Additionally, student discusses the differences or similarities between the features they considered relevant and the reported relevant features.

The features in the visualization could be more relevant than the ones you predicted because they are all continuous attributes as opposed to categorical ones. Our original categorical features got OHE, so it's possible their "power" was diluted somewhat.
Either way it's always interesting to do this analysis especially if we consider gathering more data.

You can also do feature selection on the non-OHE features quite easily with many different methods.
http://scikit-learn.org/stable/modules/feature_selection.html

Student analyzes the final model's performance when only the top 5 features are used and compares this performance to the optimized model from Question 5.

I agree entirely, I wouldn't drop the other features unless I really needed to go fast.
sometimes though we may even see an increase in our classifiers performance by dropped features which aren't important. This is due to the curse of dimensionality -- our models may perform better if the dimensionality of the data gets reduced.
So it's good to check.

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)

RETURN TO PATH

Rate this review

Student FAQ Reviewer Agreement