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

6 SPECIFICATIONS REQUIRE CHANGES

Good submission here, as you have good understanding of these techniques. Just need some simple adjustments and you will be good to go, but shouldn't be too difficult. Really enjoyed your analysis. Keep up the great work!!

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**

All correct here and great use of pandas column slicing to receive these statistics!! Always a good idea to get a basic understanding of the dataset before doing more complex computations. As we now know that we have an unbalanced dataset, so we can plan accordingly.

Preparing the Data

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

Great work with the replace method, very pythonic!

```
income = income_raw.replace(('<=50K','>50K'), (0,1))
```

Another way we could do this is with LabelEncoder from sklearn. As this would be suitable with a larger categorical range of values

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
income = le.fit_transform(income_raw)
```

```
# print one hot
print income
# then we can reverse it with
print le.inverse_transform(income)
```

Evaluating Model Performance

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

Few issues here

- First off you should be calculating these metrics on the **entire dataset** and not simply the testing set
- Also make sure you `accuracy` score is a decimal value, so we can have the accuracy and F1 score on the same scale. Therefore don't multiply this by 100
- Also your `fscore` calculation is a bit low. Relook at your formula of

  ```
  fscore = (precision*recall)/((precision*beta**2)+recall)
  ```

  As you actually forgot the first part to this equation of `(1 + beta ** 2)`. Hint: Your fscore should be in between .29 and .30.

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.

Very nice! Great to know even outside of this project! You really should also provide some citations and links for your industry examples.

KNN

- Great! Since KNN is a lazy learning method classification time is nonlinear, training is fast(simply memorize the data) however classification is slow.
- The biggest strength is how simple and effective it is
- Robust to noisy data, as long as an appropriate `n_neighbor` is used.
- Effective with large amounts of data
- Sensitive to irrelevant or redundant features as all features contribute to the similarity

Decision Tree

- Typically very fast!
- Correct! Can handle both categorical and numerical features
- As we can definitely see here that our Decision Tree has an overfitting issue here and this is typical with Decision Trees and Random Forests.
- They are are easy to visualize. Very interpretable model. Check out export_graphviz
- Another great thing that a DT model and tree methods in sklearn gives us is a feature importance. Which we use later on.

Gaussian Naive Bayes

- Correct! Often used in spam detection as this is great for word processing.
- Good. As a limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of features which are completely independent. As it does desire independent features
- Often very quick, but typically not the best predictive algorithm

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

If you want to use the `sample` idea here(which is fine)

```
learner = clf.fit(X_train.sample(n=sample_size),y_train.sample(n=sample_size))
```

You need to also set the `random_state` in this `.sample()` method, so we can have the same training data points in each of the different classifiers. So we can get an accurate comparison of how the three models do on the same data.

Note(nothing needed): Typically we only use the `.head()` method to *peak* into our dataset and we don't really use it to subset out data. Therefore would recommend checking out list slicing. For example

```
y_train[:300]
```

This is much more pythonic.

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

Nice work subsetting with the appropriate training set size. However for this section make sure you also set the `random_state` for each model you use, if provided. Thus you should include a random_state in one of these models for reproducible results.

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.

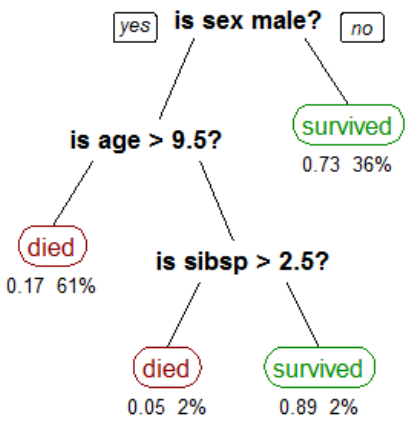
Awesome justification for your choice in your Decision Tree model, you have done a great job comparing these all in terms of predictive power and computational expense. Could also mention the extremely long prediction time for your KNN, so definitely not applicable with larger datasets.

Nice idea to go with the more interpretable and scalable Decision Tree.

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.

I think you have done a pretty good job describing how your decision tree would be trained here and glad that you have also used an example based on a feature in this dataset. This description would be great for someone who is not familiar with machine learning nor has a technical background.

Maybe another idea here would be to show and use a visual for support. Something like this(this is with the titanic dataset)



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 use of GridSearch here and the hyper-parameters of `max_depth` and `min_samples_split` are definitely the two most tuned hyper-parameters for a DecisionTreeClassifier. Would recommend using lower numbers for `max_depth`

The reason this is marked as *Requires Changes* is that you have a couple of code issues.

- Firs off relook at your code of

  ```
  grid_obj = GridSearchCV(clf,parameters)
  ```

  As you need to also pass in your scoring object. Since you actually evaluating your model based on accuracy(default) and not fbeta score.
- Also make sure you set a random state in your DecisionTreeClassifier for reproducible results

  ```
  clf = DecisionTreeClassifier(random_state = ....)
  ```

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.

You have good analysis here, however your table and analysis will need to be updated once your actually compute the correct accuracy, F1 score and use the appropriate evaluation metric in your gridSearch code.

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.

These are some great features to check out. Very intuitive.

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.

Could also use your gridSearch tuned DecisionTreeClassifier here

```
importances = best_clf.feature_importances_
```

But 4 out of 5 is not too shabby! As `feature_importances_` are always a good idea to check out for tree based algorithms. Some other ideas

- As tree based methods are good for this, if you use something like linear regression or logistic regression, the coefficients would be great to check out.
- Another idea to check out would be the feature_selection module in sklearn. As SelectKBest and SelectPercentile could also give you important features.

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.

You have good ideas here, but I am not really sure if you have mentioned if training time was a factor, would you consider using the reduced data as your training set? Can you please expand a bit more on this question? If the dataset were to grow or training time was a factor would you use this reduced dataset?

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

Student FAQ