UDACITY

Return to "Machine Learning Engineer Nanodegree" in the classroom

Finding Donors for CharityML

REVIEW

CODE REVIEW

HISTORY

Requires Changes

7 SPECIFICATIONS REQUIRE CHANGES

Very impressive submission here, as you have good understanding of these techniques and you now have a solid understanding of the machine learning pipeline. 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

```
# TODO: Number of records where individual's income is more than $50,00
0
n_greater_50k = data[(data['income'] =='>50K')]
```

```
# TODO: Number of records where individual's income is at most $50,000
n_at_most_50k = data[(data['income'] =='<=50K')]
```

You should be showing the Number of records where individual's income is more than $50,000 / at most $50,000. Not showing the actual dataframe.

For example.

```
n_greater_50k = len(data[data.income == '>50K'])
```

Preparing the Data

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

Great work with the apply method, very pythonic!

```
income =  income_raw.apply(lambda x: x.replace('<=50K','0').replace('>50K','1')).astype(int)
```

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.

Your results aren't quite right. The `recall` is correct, but all the others aren't right. For your reference,

the correct values are:

```
Naive Predictor: [Accuracy score: 0.2478, F-score: 0.2917]
```

Also make sure you use the formula of

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

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 job mentioning some real-world application, strengths / weaknesses and reasoning for your choice! Great to know even outside of this project!

Gradient Boosting / AdaBoost

- Combines multiple weak learners which can eventually lead to a more robust model, typically reduce the variance.
- They can fit noisy data.
- Another great thing that a Gradient Boosting model and tree methods in sklearn gives us is feature importances. Which we use later on.
- Good for the class imbalance
- Can choose your 'weak learner'

Random Forest

- Combines multiple decision trees which can eventually lead to a more robust model, typically reduce the variance.
- Can handle both categorical and numerical features
- As we can definitely see here that our Random Forest has an overfitting issue. This is typical with Decision Trees and Random Forests.

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

Very nice implementation of the `train_predict()` function!

One thing to be aware of in the future, in `fbeta_score()`, you can't switch the `y_true` and `y_pred`

parameter arguments. `y_true` has to come first. Check out this example to demonstrate this

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, fbeta_score
clf = DecisionTreeClassifier(max_depth=5, random_state=1)
clf.fit(X_train, y_train)
# the correct way
print(fbeta_score(y_test, clf.predict(X_test), 0.5))
# the incorrect way
print(fbeta_score(clf.predict(X_test), y_test, 0.5))
```

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

Nice work setting random states in these models and subsetting with the appropriate training set size! Could also check out the variable `results` as this gives numeric output. Maybe put it in a dataframe.

```
for i in results.items():
    print(i[0])
    display(pd.DataFrame(i[1]).rename(columns={0:'1% of train', 1:'10%
of train', 2:'100% of train'}))
```

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.

> "Based on F-score there are two model : gradiant boost classifier and adaboost classifier. the gradiant boosting perfomed better than adaboost on testing and training.
> RandomForestClassifier performed well on training data than testing data.
> f-score and accuary score in gradiant and adaboost seems to be steady in training and testing.
> gradient perfomed better in training model whereas adapost perfomed better in testing model."

This is all good analysis, but it seems like you haven't provided any discussion based on

WHICH OF THE THREE MODELS YOU BELIEVE TO BE MOST APPROPRIATE FOR THE TASK OF IDENTIFYING INDIVIDUALS THAT MAKE MORE THAN $50,000.

Which one did you choose? The AdaBoost or Gradient Boosting 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.

I, personally, really like your description of how your model would be trained. However your description is probably a bit too advanced for someone who is not familiar with machine learning nor has a technical background with terms such as "binary classification", "quality data, outliers, and noisy data", etc...

Therefore try not to use machine learning terminology in your description and try and *dumb* this down a bit more. Being able to explain models in laymen's terms is a critical thing to be able to do.

For example

- Logistic Regression is a model that gives the probability of occurrence (or non-occurrence) of an event. For example, is a person likely a donor or not? This model gives a S-curve, is used to form probability of an event with given the features. The logistic regression model tries to find a decision boundary that separates the input data into two regions (donors and non-donors). Data are classified based on the probability; 0(non-donors) if less than 0.5 and 1(donors) if greater than 0.5. Once a model has been built, it can take new data and predict the probability of the person likely to be donor or non-donor.

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 `n_estimators` and `learning_rate` are definitely the most tuned hyper-parameters for a GradientBoostingClassifier.

Pro Tip: With an unbalanced dataset like this one, one idea to make sure the labels are evenly split between the validation sets a great idea would be to use sklearn's StratifiedShuffleSplit

```
from sklearn.model_selection import StratifiedShuffleSplit
cv = StratifiedShuffleSplit(...)
grid_obj = GridSearchCV(clf, parameters,scoring=scorer, cv=cv)
```

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

Update the naive predictor after your code updates above.

Pro Tip: We could also examine the final confusion matrix. A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
%matplotlib inline

pred = best_clf.predict(X_test)
sns.heatmap(confusion_matrix(y_test, pred), annot = True, fmt = '')
```

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.

Please try and give some more specific answers to the questions of

- How do these five features compare to the five features you discussed in Question 6?
- If you were close to the same answer, how does this visualization confirm your thoughts?
- If you were not close, why do you think these features are more relevant?

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.

If training time was a factor, would you consider using the reduced data as your training set?

RESUBMIT

DOWNLOAD PROJECT

Learn the best practices for revising and resubmitting your project.

RETURN TO PATH

Rate this review