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

3 SPECIFICATIONS REQUIRE CHANGES

Good submission here, as you have good understanding of these techniques. Just need a few simple adjustments and you will be good to go(sorry for having to go against the previous reviewer), 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.

I apologize for marking this as *Requires Changes*, but you have actually not turned *records with "<=50K" to 0 and records with ">50K" to 1*. To check this out use `zip(income, income_raw)`. Therefore if you want to use the `.get_dummies()` method, please index with the column name to do this

```
income = pd.get_dummies(income_raw, drop_first=True)['>50K']
```

Evaluating Model Performance

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

Again, I apologize for marking this as *Requires Changes*, but your accuracy scores should be fully calculated "by hand", not using the scikit-learn provided functions. I know it's a bit harder, but this is just so we can be sure you have grasped the meaning of this metric. Therefore please manually calculate your accuracy score by hand. Look into using the variables you have calculated in the first section to do this.

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

Very nice work here. Great to know even outside of this project!

Support Vector Machine

- Good! Typically much slower, but the kernel trick makes it very awesome to find non-linearity in the data.
- Memory efficient, as they only have to remember the support vectors.
- Also note here the SVM output parameter are not really interpretable.
- We can use `probability = True` to calculate probabilities, however very slow, as it uses five fold CV

Gaussian Naive Bayes

- Often used in spam detection as this is great for word processing.
- Good with "*It makes a very strong assumption on the independence among features.*" 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, however this really isn't too big of an issue in practice.
- Also nice idea to use it here, as it does have a fast compute time.

Random Forest

- Ensemble of weak learners which can eventually lead to a more robust model, just make sure you set an appropriate depth for your decision tree
- Very nice! As we can definitely see here that our Random Forest has an overfitting issue here and this is typical with Decision Trees and Random Forests.
- Another great thing that an Random Forest model and tree methods in sklearn gives us is a feature importance. Which we use later on.
- Great for unbalanced datasets!

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

Very nice implementation!

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` here, as this give numerical output.

```
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.

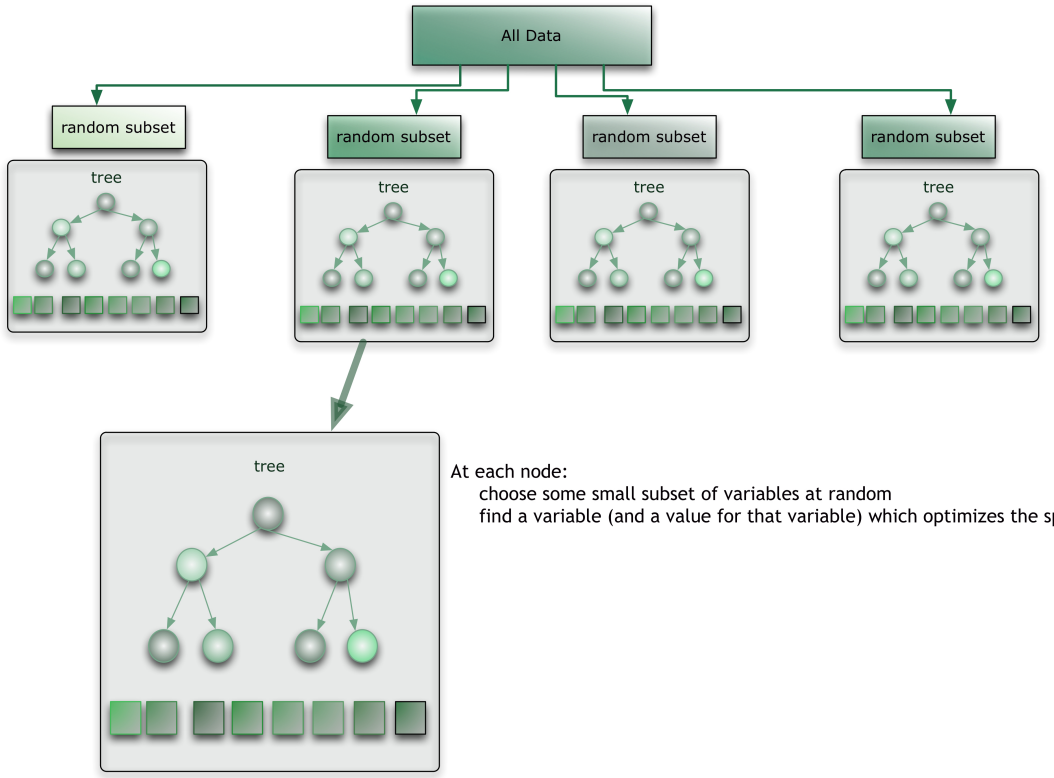
Awesome justification for your choice in your Random Forest Classifier, you have done a great job comparing these all in terms of predictive power and computational expense. I would agree with your ideas here. Glad that you have mentioned the longer training time of your SVM, as this model in O(n^3) complexity, so definitely not applicable with larger datasets. Hopefully we can get this overfitting issue down with some parameter tuning!

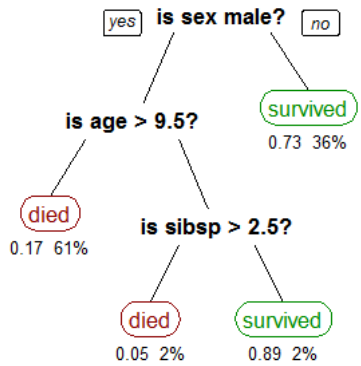
Student is able to clearly and concisely describe how the optimal model works in layman's terms to someone who is not familiar with machine learning nor has a technical background.

Awesome description of how your Random Forest model is created based on these decision trees. As your comment of "*voting or average over the outcome of all the decision trees*" is key. Therefore lastly for this section can you please go into a bit more detail in terms of how a decision tree is actually created. Think about the game 20 questions, and maybe an example of a feature in the dataset could be a good idea.

Maybe some visuals could help support your analysis

-





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 `n_estimators` are definitely two of the most tuned hyper-parameters for an RandomForestClassifier. I would highly recommend using a larger range for your `n_estimators` parameter, as we typically see this value in the hundreds!

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 to use sklearn's StratifiedShuffleSplit

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

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.

Good work comparing your tuned model to the untuned one. Maybe increasing the `n_estimators` could help out a bit more.

Pro Tip: We could also examine the final confusion matrix

```
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.

Could also use your gridSearch tuned RandomForestClassifier here

```
importances = best_clf.feature_importances_
```

It is tough to guess these features, which is why we have machine learning models to do this for us! 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.

Would agree with your ideas. Another idea, instead of solely picking a subset of features, would be to try out algorithms such as PCA. Which can be handy at times, as we can actually combine the most correlated/prevalent features into something more meaningful and still can reduce the size of the input. You will see this more in the next project!

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