UDACITY

PROJECT

Finding Donors for CharityML

A part of the Machine Learning Engineer Nanodegree Program

PROJECT REVIEW

CODE REVIEW

NOTES

SHARE YOUR ACCOMPLISHMENT!

Meets Specifications

Congratulations! Your revised submission is perfect, and you have done a great job to successfully completed this project on classification. Keep up your awesome 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**

Well done on getting the statistics. We can also examine if there are any missing values using `pd.info()`, and view a few samples using `pd.head()`.

From the data exploration, we notice that the number of income no greater than 50k (`n_at_most_50k`) is more than three times the number of income greater than 50k (`n_greater_50k`). Therefore the data is unbalanced. Some techniques to handle unbalanced data include:

- Stratification, which preserves the relative portion of samples for each class
- Using precision / recall / F1 score as performance metric, rather than accuracy.

Preparing the Data

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

Good job to transform the feature. As an additional reference, here is another method to transform income through Python `map` :

```
dic = {'<=50K' : 0, '>50K' : 1}
income = income_raw.map(dic)
```

Evaluating Model Performance

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

Well done to get the scores. For the naive predictor, it is interesting to note that the precision is equivalent to accuracy, and recall remains at 1 consistently.

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 good choice of the models.

As you have noted, KNN is an intuitively simple algorithm, but it requires a good definition of 'distance', which may be hard to find for high dimensional data. However, we can apply dimension reduction techniques like PCA to reduce the dimension before applying kNN. We will learn more about PCA in subsequent lessons.

Logistic Regression has low computation cost, and works well when there are noise in the data. Essentially, it assigns different weights to each feature and sums the weighted features. The sum then goes through the logistic function, which yields the final probability. Comparing Logistic Regression with Decision Tree, Logistic Regression generally works better for small data size, whereas Decision Tree tends to outperform as the data size grows. For categorical features, we generally need to use techniques such as one hot encoding to preprocess them.

Decision tree is the building block for Random Forest. Decision tree is popular in that it is easy to interpret, and can handle mixed type of features, both categorical and numerical (which is typical in most datasets nowadays). However, it tends to overfit easily. Other than managing overfitting by cross validation, we often use ensemble methods such as Random Forest or AdaBoost to improve the performance.

Ensemble methods are very popular in data science competitions. There are two main categories of ensemble methods: bagging and boosting. Both build a strong model on a set of weak models. The difference is that bagging generates the weak learners with equal probability, whereas boosting tries to generate new models to target for cases where the previous models fail. Random Forest is a bagging technique, and examples of boosting include AdaBoost, and XGBoost. You can read more about their differences here: https://quantdare.com/what-is-the-difference-between-bagging-and-boosting.

How to choose the right model：

This is largely application dependent, but we can start with the two cheat sheets below for some general guidelines on how to choose the right model:

http://scikit-learn.org/stable/tutorial/machine_learning_map/
https://docs.microsoft.com/en-us/azure/machine-learning/machine-learning-algorithm-cheat-sheet

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

Great job to implement the classification pipeline. In addition to the F-score and computation time, we can also evaluate the models in more details using other sklearn functions, such as the confusion matrix and classification report. For example:

```
#confusion matrix example
cm = confusion_matrix(y_test, y_pred)
print cm

# classification report example
target_names = ['class 0', 'class 1', 'class 2']
cr = classification_report(y_true, y_pred, target_names=target_names)
print cr
```

Ref:
http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#example-model-selection-plot-confusion-matrix-py
http://scikit-learn.org/stable/modules/model_evaluation.html#classification-report

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

```
# TODO: Initialize the three models
clf_A = KNeighborsClassifier()
clf_B = LogisticRegression(random_state=11)
clf_C = RandomForestClassifier(random_state=12)
```

It is awesome that you set the random states of the models. Here we are required to set the `random_state` of the models explicitly, and we need to use the same random states for the algorithms with default parameters and for the algorithm optimization via grid search later. Using the same random states ensures that we can have a fair comparison between the default and optimized algorithms.

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.

Your choice of Logistic Regression is very reasonable, and it is great that you have accounted for both the computation cost and classification performance.

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.

Nice explanation. Essentially, logistic regression assigns different weights to each feature and sums the weighted features. The sum then goes through the logistic function, which yields the final probability. The weights are found through training such that they best separate the two classes. You can find a good introduction here: http://www.theanalysisfactor.com/explaining-logistic-regression/

Udacity provides a good overview of most of the algorithms. After taking a couple of online courses, my personal preference is to get a deeper dive into the algorithms, and here are two of my favorite books (the former is introductory, and the latter is more involved):

- **Introduction to Statistical Learning**: http://www-bcf.usc.edu/~gareth/ISL/
- **The Elements of statistical Learning**: https://web.stanford.edu/~hastie/ElemStatLearn/

Both of the above books have very good discussion on Logistic Regression, and many other algorithms.

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.

```
grid_fit = grid_obj.fit(X_train, y_train.astype('int'))
```

It is noteworthy that here we are using the training set to tune the model, and keeping the test set away to avoid data leakage. As a general guideline, we should never touch the test set during model training and tuning, and should only use it for final model evaluation.

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.

> Both F-score and Accuracy Score of the optimized model are a bit better than those of the unoptimized ones.

It is possible that the tuned model does not significantly improve over untuned one, or it could even give worse result than untuned one. The first checkpoint is whether the list of tuning parameters includes the default parameters. Sometimes the default parameters can yield the best performance.

We may observe different results if we run the code with different random splits. This is largely due to the small and unbalanced dataset, and we can use techniques like `StratifiedShuffleSplit` to ensure that the ratio of the two classes are well maintained. For example:

```
from sklearn.cross_validation import StratifiedShuffleSplit
...
ssscv = StratifiedShuffleSplit( y_train, n_iter=10, test_size=0.1)
grid = GridSearchCV(clf, parameters, cv = ssscv , scoring=f1_scorer)
grid.fit( X_train, y_train )
...
```

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.

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.

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 suppose we can use the reduced data set (one with only the 5 important features) and still get a comparable model performance, but this should be done only when we have huge amounts of data because in that case the gain in training time on the reduced set will neglect the small hit in model performance.

I absolutely agree. In general, a model with the full feature set should outperform the model with reduced feature set, as more features contribute more information. However, in real life projects, we often have to balance computation time against the model performance. If the data size is large, it is a good idea to select important features to simplify the model.

DOWNLOAD PROJECT

RETURN TO PATH

Student FAQ