UDACITY

‹ Return to "Machine Learning Engineer Nanodegree" in the classroom

Finding Donors for CharityML

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

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

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 classifier, 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, and nice discussion.

As you have noted, 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.

Lastly, Naive Bayes a simple algorithm, but it can be very powerful, as it is fast, and works well for small data. As Naive Bayes is built on probability, it is also robust to outliers / missing data. However, Naive Bayes assumes that every feature is independent of others. In practice, this assumption may not hold. So the performance of Naive Bayes is limited by the validity of the assumption. Violation of the assumption may degrade its performance.

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/

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.

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.

Although it is not the fastest, for this small data, we can focus more on the classification performance, and your choice of Random Forest is very reasonable.

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 of Random Forest. Here is another example that really helped me to understand Random Forest:

http://blog.echen.me/2011/03/14/laymans-introduction-to-random-forests/

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)
```

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.

> The result from the optimized model is well improved compared to the results from the unoptimized mode.

Although not the case here, 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.

> As we can see output above, the results from the reduced data are slightly lower than from full data but if training time was a factor, it would be better to use the reduced data as our training set.

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