UDACITY

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

Finding Donors for CharityML

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Very good work ! you showed a great understanding of the concepts being presented here, well done !

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**

Great job getting the numbers !

Preparing the Data

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

Good job ! For your reference, you can check out this article which explains when and why we use One Hot Encoding.

Evaluating Model Performance

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

Good job ! I recommend having a look at this great article to understand more about choosing the right metric for classification problems.

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 discussion ! You can take a look on this cheat sheet to understand more about model selection.

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

Well done implementing the pipeline ! You can learn more about the use of pipelines in ML reading the scikit-learn documentation

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

Well done ! For more information about random_state please take a look on this article

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.

Very good justification that takes into consideration computational cost, model performance, and the characteristics of the data !

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.

Great explanation !

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 implementation of GridSearch ! note that GridSearch is not the only technique available to us. Another similar technique worth looking is RandomizedSearchCV.

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.

indeed, well done !

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 very interesting features indeed !

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.

Good job getting the `feature_importances_` !
As you can see in this example, intuition about the feature importances in any ML problem is a good initial approach, but a thorough method is a better approach since its conclusions are based on the data relations between features and label.

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.

Good examination of the model's performance with the reduced feature set — the results are worse, but this might be acceptable if computational cost were a bigger issue.
We could also try adding back just a few more "important" features to see if the accuracy improves without increasing processing time significantly.
In general, feature reduction is a great way to fight the curse of dimensionality.
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!
Maybe something like this:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=0.8, whiten=True)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
clf_pca = (clone(best_clf)).fit(X_train_pca, y_train)
pca_predictions = clf_pca.predict(X_test_pca)
print("\nFinal Model trained on PCA data\n------")
print("Accuracy on testing data: {:.4f}".format(accuracy_score(y_test, pca
_predictions)))
print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, pca_pre
dictions, beta = 0.5)))
```

DOWNLOAD PROJECT

RETURN TO PATH