UDACITY

Return to "Machine Learning Engineer Nanodegree" in the classroom

Finding Donors for CharityML

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

🏆 Excellent job! I'm impressed with how you've grasped the main concepts of supervised learning using scikit-learn. 😎

And if you have more questions you can contact me on slack at @jameslee, or if I'm one of your study room mentors you can message me there.

Exploring the Data

Student's implementation correctly calculates the following:

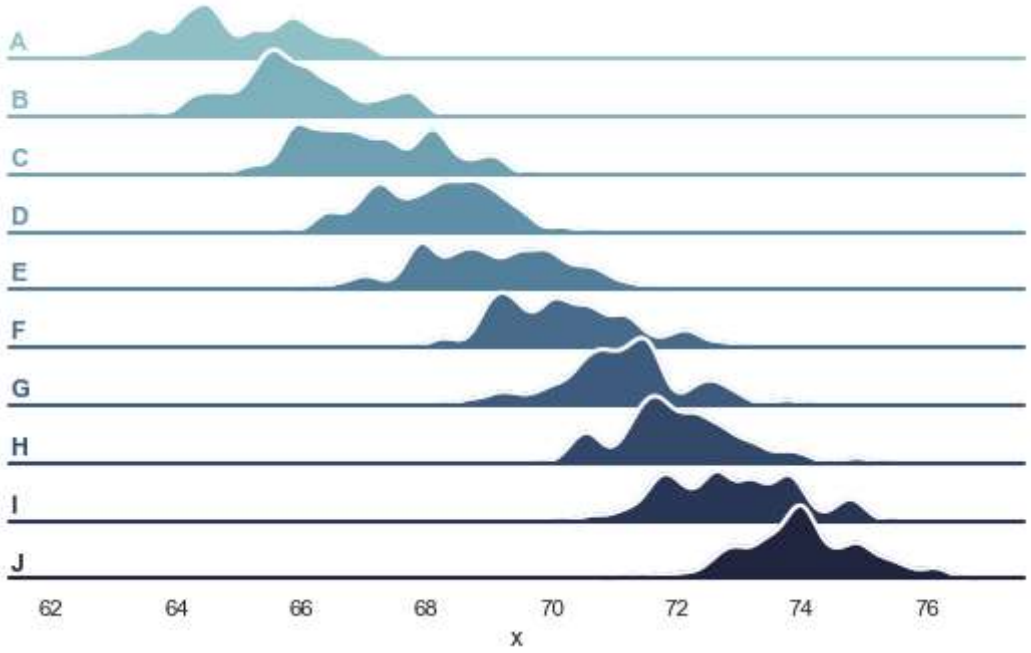
- **Number of records**
- **Number of individuals with income >$50,000**
- **Number of individuals with income <=$50,000**
- **Percentage of individuals with income > $50,000**

Great job getting all the dataset stats correct. 😄

Pro tip: visualizing data

The udacity template doesn't go into exploratory data analysis (EDA) that much, but it's usually a big part of any project.

e.g., another technique you could try out is a ridge plot in seaborn.



Preparing the Data

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

Excellent work encoding the features and target labels. 😀

Evaluating Model Performance

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

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.

Good discussion of the models and why you chose them. 😎

With model selection it's often a good idea to try out simpler methods like Logistic Regression or ElasticNet as a baseline, and then move on to other approaches such as your choice of Decision Trees.

(see this Microsoft guide on choosing an algorithm for more info)

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

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

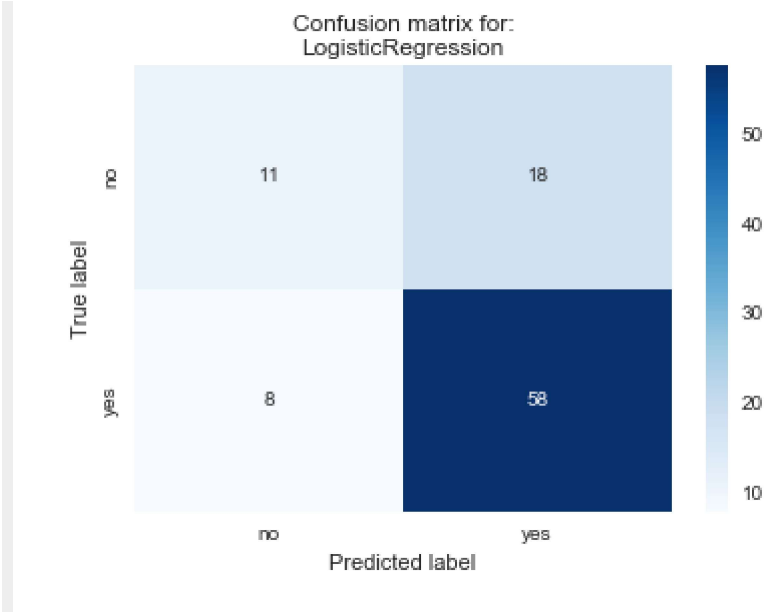
Great job generating the results with the 3 sample sizes, and setting random states on the classifiers to make your notebook results reproducible. 😉

To further examine the models, you can also get a closer look at they are predicting the labels by plotting a confusion matrix...

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Compute confusion matrix for a model
model = clf_A  # choose a model
cm = confusion_matrix(y_test, model.predict(X_test))
cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis] # normalize

# view with a heatmap
sns.heatmap(cm, annot=True, annot_kws={"size":30},
            cmap='Blues', square=True, fmt='.2f')
```



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.

Good justification of your choice of model by looking at factors such as the accuracy/F-scores and computational cost/time. 😄

If you're interested in the best predictive accuracy, the highest scores I've seen from students used either Gradient Boosting or AdaBoost, with F-scores around `0.75 - 0.77` on the test set.

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.

For more ideas on describing common ML models in plain english, you can also check out these descriptions:

- https://hackerbits.com/data/top-10-data-mining-algorithms-in-plain-english/
- http://www.kdnuggets.com/2017/08/machine-learning-algorithms-concise-technical-overview-part-1.html

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 job tuning the decision tree. 😎

When the grid search is done, you might want to also output the best parameters found using something like `grid_fit.best_params_` or simply `print(best_clf)`.

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.

💡 Pro tip: randomized search

For future reference, another common parameter tuning approach to try out is randomized search — with sklearn's `RandomizedSearchCV` method you can often get comparable results at a much lower run time.

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.

Excellent job extracting the feature importances of the model and discussing how the features could be considered important to making predictions.

You can also try finding important features by calculating permutation importances with the eli5 library:

```
import eli5
from eli5.sklearn import PermutationImportance

perm = PermutationImportance(best_clf, n_iter=10).fit(X_test, y_test)
eli5.show_weights(perm, feature_names = X_test.columns.tolist())
```

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 discussion of the model's performance with the reduced feature set — the results compare quite well with those generated using the full list of features. 😀

In general, feature reduction is a great way to fight the curse of dimensionality.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review