PROJECT

Finding Donors for CharityML

A part of the Machine Learning Engineer Nanodegree Program

PROJECT REVIEW | CODE REVIEW | NOTES

Meets Specifications

SHARE YOUR ACCOMPLISHMENT

★★★★★

CONGRATULATIONS!

Outstanding job with the report, I'm impressed with how you've iterated on the project.

For a guide to approaching almost any machine learning problem, you can check out this blog post by a Kaggle grandmaster, and if you haven't already been recommended this Python ML book by another reviewer, it's definitely worth a look. 😎

Exploring the Data

Student's implementation correctly calculates the following:

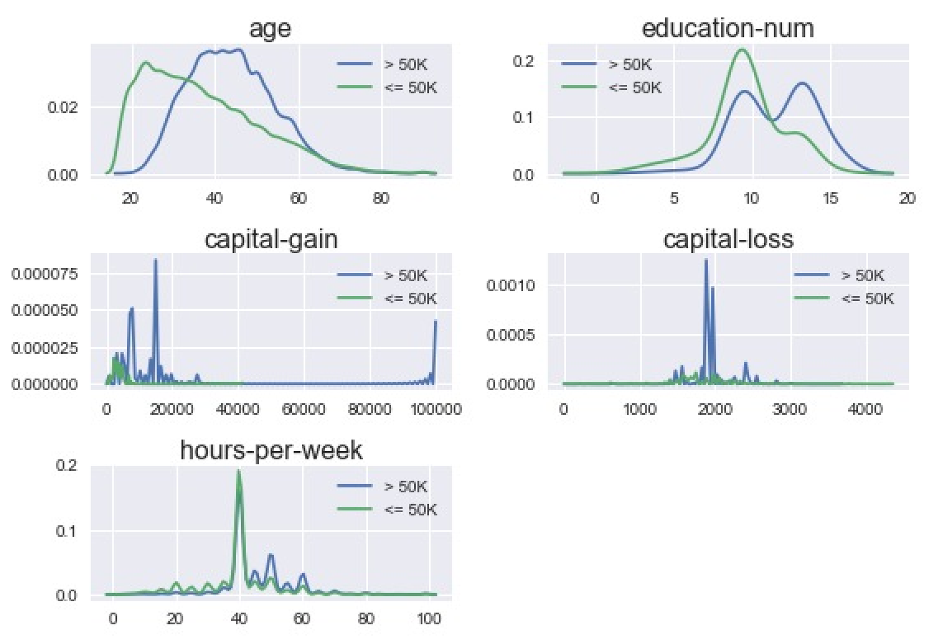
- **Number of records**
- **Number of individuals with income >$50,000**
- **Number of individuals with income <=$50,000**
- **Percentage of individuals with income > $50,000**

Another we can do to visualize the data is to plot the distributions of our numerical features with kdeplot...

```
import matplotlib.pyplot as plt
import seaborn as sns

num = ['age', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week']

for i, feat in enumerate(num):
    plt.subplot(3, 2, i+1)
    df = data[[feat, 'income']]
    df_greater = df[df.income == '>50K']
    df_less = df[df.income == '<=50K']
    sns.kdeplot(df_greater[feat], bw=1, label = "> 50K")
    sns.kdeplot(df_less[feat], bw=1, label = "<= 50K")
    plt.title(feat, size=16)
plt.tight_layout()
```



Preparing the Data

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

The best way to write clean concise ML code is just to keep practicing and look at examples of other people's work for inspiration. For example, to see another look at how you could approach a Python data analysis, you can step through this example ML notebook.

And for other ML resources that are worth checking out see this list from the kaggle blog.

Evaluating Model Performance

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

DONE!

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

I'm not aware of free online resources for working with large datasets, but in addition to AWS you could try looking at Google cloud and Microsoft Azure. Here's a microsoft azure guide on choosing an algorithm for your analysis.

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

Excellent work implementing the pipeline!

- With this pipeline you can see how the performance of the 3 models changes when using different training sizes passed to the `sample_size` parameter.
- You've already done this, but you can also experiment on your own with making predictions on the training set using `sample_size` — for the sake of speed the project guidelines call for only using the first 300 training points.

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

DONE!

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.

In addition to a confusion matrix, you can see how the models compare on precision, recall, and F1 scores by using classification_report...

```
from sklearn.metrics import classification_report
for clf in [clf_A, clf_B, clf_C]:
    print '\nReport for {}:\n'.format(clf.__class__.__name__)
    print classification_report(y_test, clf.predict(X_test))
    print '-'*52
```

```
Report for LinearSVC:

             precision    recall  f1-score   support

          0       0.88      0.93      0.90      6840
          1       0.74      0.60      0.66      2205

avg / total       0.84      0.85      0.85      9045

...
```

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.

DONE!

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.

For another look at performing logistic regression using `statsmodels` instead of sklearn, you can check out this post on the yhat blog.

```
import statsmodels.api as sm

# fit the model
result = sm.Logit(y_train, X_train).fit()

# look at the results
print result.summary()

# odds ratios only
print np.exp(result.params)
```

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.

DONE!

Feature Importance

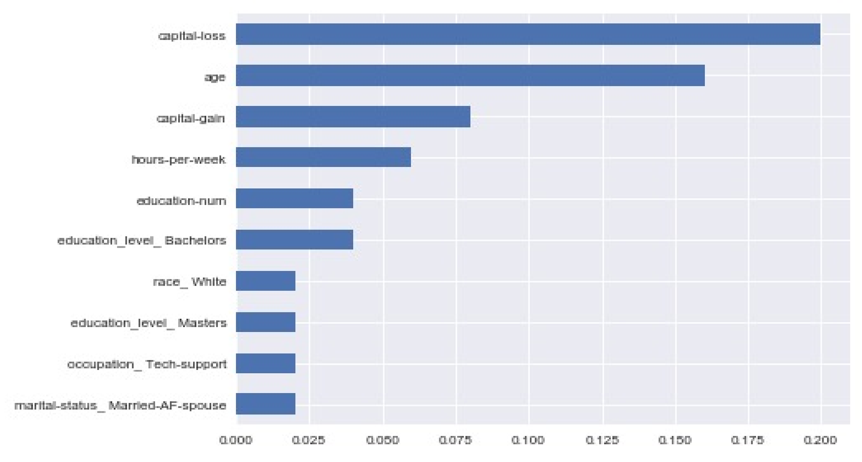
Student ranks five features which they believe to be the most relevant for predicting an individual's' income. Discussion is provided for why these features were chosen.

DONE!

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.

To explore beyond the 5 most important features, you can try using pandas plotting to look at some of the other feature importances as well...

```
# look at top "n" feature importances
n = 10
fi = model.feature_importances_
pd.Series(fi, index=X_train.columns).sort_values()[-n:].plot(kind='barh');
```



Read more on feature selection techniques here:
http://machinelearningmastery.com/feature-selection-machine-learning-python/

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.

DONE!

DOWNLOAD PROJECT

Have a question about your review? Email us at review-support@udacity.com and include the link to this review.

RETURN TO PATH