Finding Donors for CharityML

REVIEW | CODE REVIEW | HISTORY

Requires Changes

1 SPECIFICATION REQUIRES CHANGES

Very impressive submission here, as you have good understanding of these techniques and you now have a solid understanding of the machine learning pipeline. Just need one more simple adjustment and you will be good to go, but shouldn't be too difficult. Really enjoyed your analysis. Keep up the great 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**

All correct here and great use of pandas column slicing to receive these statistics!! Always a good idea to get a basic understanding of the dataset before doing more complex computations. As we now know that we have an unbalanced dataset, so we can plan accordingly.

Another great idea would be to preform some extra exploratory data analysis for the features. Could check out the library Seaborn. For example

```
import seaborn as sns
sns.factorplot('income', 'capital-gain', hue='sex', data=data, kind='bar'
)
```

Preparing the Data

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

Great work with the map method, very pythonic!

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

Another way we could do this is with LabelEncoder from sklearn. As this would be suitable with a larger categorical range of values

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
income = le.fit_transform(income_raw)
# print one hot
print income
# then we can reverse it with
print le.inverse_transform(income)
```

Evaluating Model Performance

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

Impressive calculation. Another idea would be to use the `np.square()` method.

```
fscore = (1+np.square(beta))*(precision * recall)/((np.square(beta) * pre
cision) + recall)
```

It is always a great idea to establish a benchmark in any type of problem. As these are now considered our "dumb" classifier results, as any real model should be able to beat these scores, and if they don't we may have some model issues.

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 nice job mentioning some real-world application, strengths / weakness and reasoning for your choice! Great to know even outside of this project!

Gaussian Naive Bayes

- Often used in spam detection as this is great for word processing.
- Correct. As a limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of features which are completely independent.
- Often very quick, but typically not the best predictive algorithm
- High bias algorithm

Decision Tree

- Typically very fast!
- Can handle both categorical and numerical features
- As we can definitely see here that our Decision Tree has an overfitting issue. This is typical with Decision Trees and Random Forests.
- They are are easy to visualize. Very interpretable model. Check out export_graphviz
- Another great thing that a Decision Tree model and tree methods in sklearn gives us is feature importances. Which we use later on.

Support Vector Machine

- Typically much slower, but the kernel trick makes it very nice to fit non-linear data.
- They are memory efficient, as they only have to remember the support vectors.
- Also note that the SVM output parameters are not really interpretable.
- We can use `probability = True` to calculate probabilities, however very slow as it used five fold CV

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

Very nice implementation of the `train_predict()` function!

One thing to be aware of in the future, in `fbeta_score()`, you can't switch the `y_true` and `y_pred` parameter arguments. `y_true` has to come first. Check out this example to demonstrate this

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, fbeta_score
clf = DecisionTreeClassifier(max_depth=5, random_state=1)
clf.fit(X_train, y_train)
# the correct way
print fbeta_score(y_test, clf.predict(X_test), 0.5)
# the incorrect way
print fbeta_score(clf.predict(X_test), y_test, 0.5)
```

It actually doesn't matter for `accuracy_score()` sorry that you had to redo all this.

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

Nice work setting random states in these models and subsetting with the appropriate training set size! Could also check out the variable `results` as this gives numeric output. Maybe put it in a dataframe.

```
for i in results.items():
    print(i[0])
    display(pd.DataFrame(i[1]).rename(columns={0:'1% of train', 1:'10% of train', 2:'100% of train'}))
```

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.

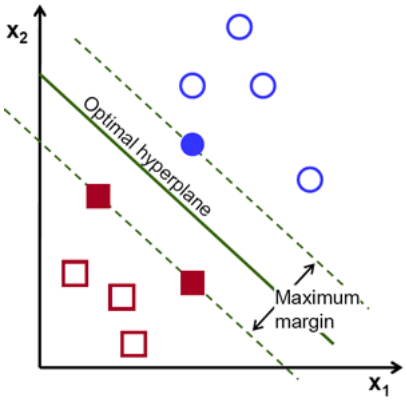
Good justification for your choice in your SVM model, you have done a great job comparing these all in terms of predictive power and computational expense. Glad that you mention the overfitting issue with your DT.

As in practice we always need to think about multiple things

- the predictive power of the model
- the runtime of the model and how it will scale to much more data
- the interpretability of the model
- how often we will need to run the model and/or if it support online learning.

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.

You have done a good job mentioning the 'kernel' trick for an SVM, however there is one more big thing that should also be addressed in more detail when describing this algorithm. How is the actual data "cut"? What does a SVM try to 'maximize'?



(http://scikit-learn.org/stable/auto_examples/svm/plot_separating_hyperplane.html#sphx-glr-auto-examples-svm-plot-separating-hyperplane-py)
(https://en.wikipedia.org/wiki/Support_vector_machine#Motivation)
(https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72)

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 use of GridSearch here and the hyper-parameters of `C` and `gamma` are definitely the most tuned hyper-parameters for an SVM.

- A large C gives you low bias and high variance. Low bias because you penalize the cost of misclassification a lot. A small C gives you higher bias and lower variance.
- A small gamma gives you a pointed bump in the higher dimensions, a large gamma gives you a softer, broader bump. So a small gamma will give you low bias and high variance while a large gamma will give you higher bias and low variance.

Pro Tip: With an unbalanced dataset like this one, one idea to make sure the labels are evenly split between the validation sets a great idea would be to use sklearn's StratifiedShuffleSplit

```
from sklearn.model_selection import StratifiedShuffleSplit
cv = StratifiedShuffleSplit(...)
grid_obj = GridSearchCV(clf, parameters,scoring=scorer, cv=cv)
```

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.

Good work comparing your tuned model to the untuned one.

Pro Tip: We could also examine the final confusion matrix. A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
%matplotlib inline

pred = best_clf.predict(X_test)
sns.heatmap(confusion_matrix(y_test, pred), annot = True, fmt = '')
```

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 some great features to check out. Very intuitive.

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.

Really enjoyed reading your analysis! As `feature_importances_` are always a good idea to check out for tree based algorithms. Some other ideas

- As tree based methods are good for this, if you use something like linear regression or logistic regression, the coefficients would be great to check out.
- Another idea would be to check out the feature_selection module in sklearn. As SelectKBest and SelectPercentile could also give you important 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.

This is a huge dip in performance. Maybe try a different algorithm instead?

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_predictions, beta = 0.5))
```

RESUBMIT PROJECT

DOWNLOAD PROJECT

Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

RETURN TO PATH

Student FAQ