Finding Donors for Charity

OVERVIEW

In this project, you will employ several supervised algorithms of my choice to accurately model individuals' income using data collected from the 1994 U.S. Census. I will then choose the best candidate algorithm from preliminary results and further optimize this algorithm to best model the data. The goal with this implementation is to construct a model that accurately predicts whether an individual makes more than $50,000. This sort of task can arise in a non-profit setting, where organizations survive on donations. Understanding an individual's income can help a non-profit better understand how large of a donation to request, or whether they should reach out to begin with. While it can be difficult to determine an individual's general income bracket directly from public sources, we can infer this value from other publically available features.

DATA EXPLORATION

A cursory investigation of the dataset will determine how many individuals fit into either group and will tell us about the percentage of these individuals making more than $50,000.

```
Total number of records: 45222
Individuals making more than $50,000: 11208
Individuals making at most $50,000: 34014
Percentage of individuals making more than $50,000: 24.78%
```

Feature Exploration:

- **age**: continuous.
- **workclass**: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
- **education**: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-vocal, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.
- **education-num**: continuous.
- **marital-status**: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
- **occupation**: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
- **relationship**: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
- **race**: Black, White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other.
- **sex**: Female, Male.
- **capital-gain**: continuous.
- **capital-loss**: continuous.
- **hours-per-week**: continuous.
- **native-country**: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc.), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador,

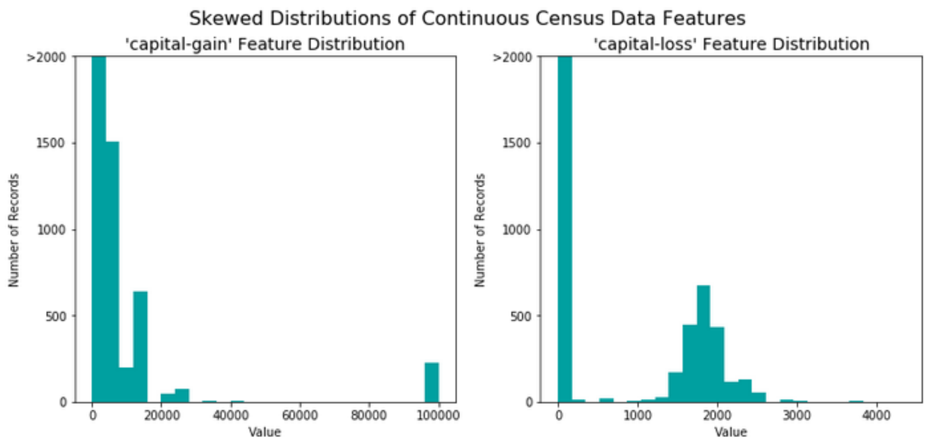
Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinidad & Tobago, Peru, Hong, Holland-Netherlands.

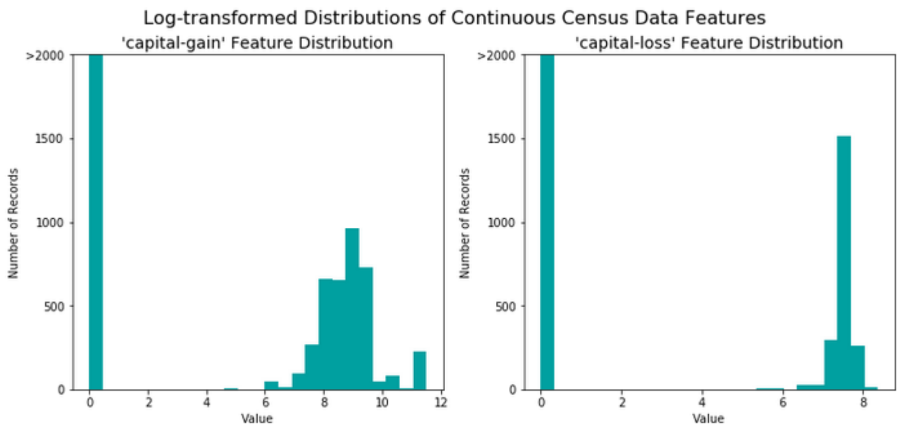
DATA PREPROCESSING & FEATURE ENGINEERING

Fortunately, for this dataset, there are no invalid or missing entries however, there are some qualities about certain features that must be adjusted.

Transforming Skewed Continuous Features

Algorithms can be sensitive to skewed distributions of values and can underperform if the range is not properly normalized. With the census dataset two features fit this description: 'capital-gain' and 'capital-loss'. I will apply a logarithmic transformation on the two features so that the very large and very small values do not negatively affect the performance of a learning algorithm. Using a logarithmic transformation significantly reduces the range of values caused by outliers. Care must be taken when applying this transformation because the logarithm of 0 is undefined, so I must translate the values by a small amount above 0 to apply the logarithm successfully.



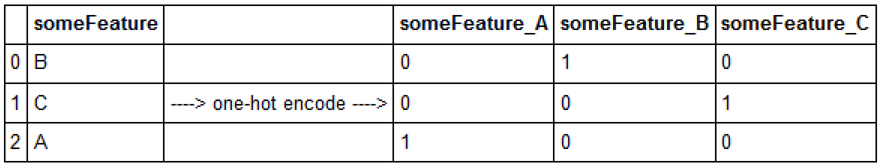


Normalizing Numerical Features

In addition to performing transformations on features that are highly skewed, I must perform some type of scaling on numerical features. Applying scaling to the data does not change the shape of each feature's distribution (such as 'capital-gain' or 'capital-loss' above); however, normalization ensures that each feature is treated equally when applying supervised learners. In my case I used the MinMaxScaler().

Implementation: Data Preprocessing

Machine learning algorithms expect input to be numeric, which requires that non-numeric features (called categorical variables) be converted. One popular way to convert categorical variables is by using the one-hot encoding scheme. One-hot encoding creates a "dummy" variable for each possible category of each non-numeric feature. For example, assume someFeature has three possible entries: A, B, or C. We then encode this feature into someFeature_A, someFeature_B and someFeature_C.



	someFeature		someFeature_A	someFeature_B	someFeature_C
0	B		0	1	0
1	C	----> one-hot encode ---->	0	0	1
2	A		1	0	0

Additionally, as with the non-numeric features, I need to convert the non-numeric target label, 'income' to numerical values for the learning algorithm to work. Since there are only two possible categories for this label ("<=50K" and ">50K"), we can avoid using one-hot encoding and simply encode these two categories as 0 and 1, respectively. In code cell below, you will need to implement the following:

- Use pandas.get_dummies() to perform one-hot encoding on the 'features_log_minmax_transform' data.
- Convert the target label 'income_raw' to numerical entries.
- Set records with "<=50K" to 0 and records with ">50K" to 1.
- `Code example: income = income_raw.apply(lambda x:1 if x == '>50K'else 0)`

Shuffle and Split Data

Now all categorical variables have been converted into numerical features, and all numerical features have been normalized. Now I can split the data (both features and their labels) into training and test sets. 80% of the data will be used for training and 20% for testing.

```
Training set has 36177 samples.
Testing set has 9045 samples.
```

BENCHMARK & EVALUATION MODEL PERFORMANCE

Metrics and the Naive Predictor

CharityML, knows individuals that make more than $50,000 are most likely to donate to their charity. Because of this, *CharityML* is particularly interested in predicting who makes more than $50,000 accurately. Using the **accuracy** as a metric for evaluating a particular model's performance would be appropriate. Additionally, identifying someone that does not make more than $50,000 as someone who does would be detrimental to *CharityML*, since they are looking to find individuals willing to donate. Therefore, a model's ability to precisely predict those that make more than $50,000 is more important than the model's ability to **recall** those individuals. I can use **F-beta score** as a metric that considers both precision and recall:

$$F_{\beta} = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

When beta = 0.5, more emphasis is placed on precision. This is called the **F 0.5 score** (or F-score for simplicity).

Looking at the distribution of classes (those who make at most `$50,000`, and those who make more), it's clear most individuals do not make more than $50,000. This can greatly affect **accuracy**, since we could simply say *"this person does not make more than $50,000"* and generally be right, without ever looking at the data! Making such a statement would be called **naive**, since we have not considered any information to substantiate the claim. It is always important to consider the *naive prediction* in order to help establish a benchmark for whether a model is performing well. Using that prediction would be pointless: If we predicted all people made less than $50,000, *CharityML* would identify no one as donors.

Accuracy: `[(True Positives + False Positives)/Total]`

Precision: `[True Positives/ (True Positives + False Positives)]`

Recall: `[True Positives/ (True Positives + False Negatives)]`

When we have a model that always predicts '1' (i.e. the individual makes more than 50k) then our model will have no True Negatives (TN) or False Negatives (FN) as we are not making any negative ('0' value) predictions. Therefore our Accuracy in this case becomes the same as our Precision(True Positives/(True Positives + False Positives)) as every prediction that we have made with value '1' that should have '0' becomes a False Positive; therefore our denominator in this case is the total number of records we have in total.

When we have a model that always predicts '1' (i.e. the individual makes more than 50k) then our model will have no True Negatives (TN) or False Negatives (FN) as we are not making any negative ('0' value) predictions. Therefore our Accuracy in this case becomes the same as our Precision(True Positives/(True

Positives + False Positives)) as every prediction that we have made with value '1' that should have '0' becomes a False Positive; therefore our denominator in this case is the total number of records we have in total.

Naïve Predictor Outcome: [Accuracy score: 0.2478, F-score: 0.2917]

<u>Model Application</u>

Bagged Trees	Random Forest	Logistic Regression
• Strengths: reduces variance in comparison to regular decision trees. It can provide variable importance measures, classification (Gini Index) and Regression (RSS). Can easily handle qualitative (categorical) features • Weaknesses: Not easy to visually interpret, does not reduce variance if the features are correlated • Reason for selection: I would like to compare the performance difference between this and Random Forrest.	• Strengths: can be used for both regression and classification tasks • Weaknesses: by creating more trees in order to avoid overfitting the algorithm becomes computationally inefficient • Reason for selection: It is a relatively fast algorithm especially compared to Support Vector Machines and it works great with High dimensionality. Random forest is also great at handling outliers by binning them.	• Strengths: can avoid overfitting through regularization and allows for fast iterations. • Weaknesses: can underperform when there are multiple boundaries, not robust enough for complex data. High Bias • Reason for selection: Since the selected models tend to overfit and there is a lack of lack of linearity amongst the features in the data set. It's also a great model for binary classification.

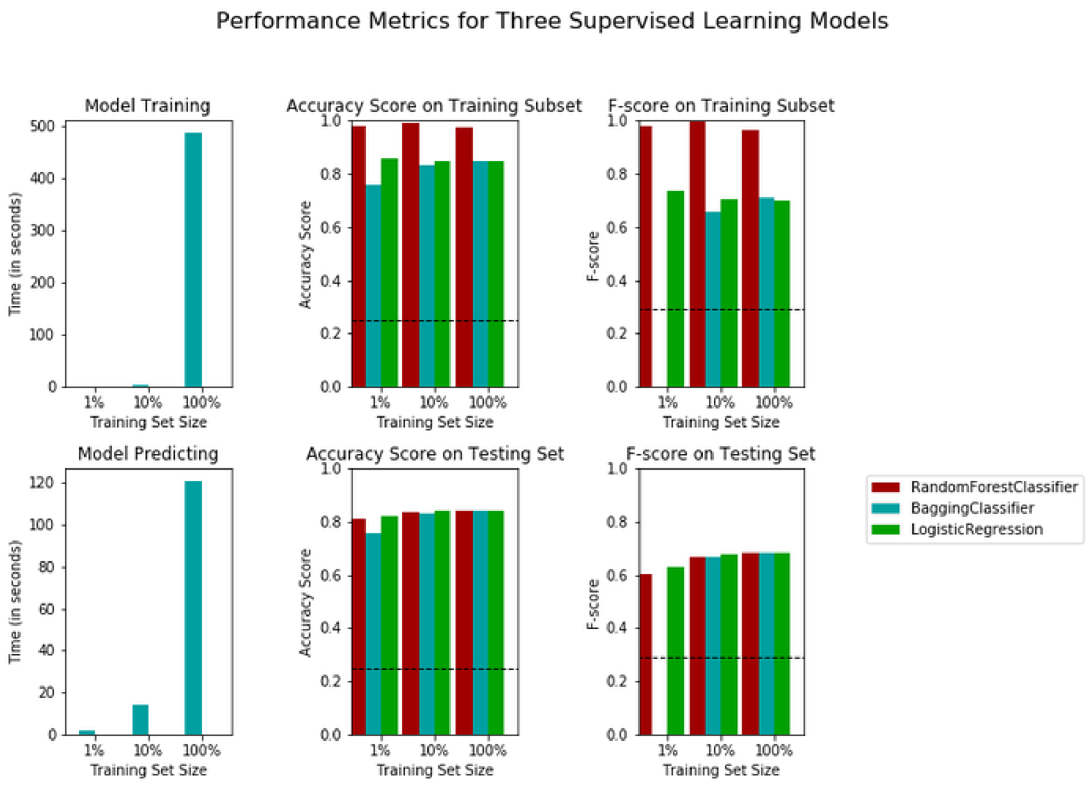
<u>Implementation - Creating a Training and Predicting Pipeline</u>

To properly evaluate the performance of each model it's important to create a training and predicting pipeline that allows for a quick and effective away to train models using various sizes of training data and perform predictions on the testing data. The implementations here will be the following

- Import fbeta_score and accuracy_score
- Fit the learner to the sampled training data and record the training time.
- Perform predictions on the test data X_test, and on the first 300 training points X_train [:300].
- Record the total prediction time.
- Calculate the accuracy score for both the training subset and testing set.
- Calculate the F-score for both the training subset and testing set.

Initial Model Evaluation

- Import the three supervised learning models discussed in the previous section.
- Initialize the three models and store them in 'clf_A', 'clf_B', and 'clf_C'.
- Use a 'random_state' for each model used, if provided.
- Calculate the number of records equal to 1%, 10%, and 100% of the training data.
- Store those values in 'samples_1', 'samples_10', and 'samples_100' respectively.



MODEL OPTIMIZATION

After selecting the top performing model, in this case Random Forest; It's time to perform a grid search optimization for the model over the entire training set (`X_train` and `y_train`) by tuning some of the parameters to improve upon the untuned model's F-score.

```
# Initialize the classifier
clf = RandomForestClassifier(bootstrap=True, random_state=42)

# Create the parameters list using a dictionary
parameters = {'n_estimators':[100,300,500], 'max_leaf_nodes': [60, 80, 100]}

# Make an fbeta_score scoring object using make_scorer()
scorer = make_scorer(fbeta_score, beta=0.5)

# Perform grid search on the classifier using 'scorer' as the scoring method
using GridSearchCV()
grid_obj = GridSearchCV(clf, parameters, scoring=scorer)
```

```
# Fit the grid search object to the training data and find the optimal
parameters
grid_fit = grid_obj.fit(X_train, y_train)

# Get the estimator
best_clf = grid_fit.best_estimator_

# Make predictions using the unoptimized and optimized model
predictions = (clf.fit(X_train, y_train)).predict(X_test)
best_predictions = best_clf.predict(X_test)
```

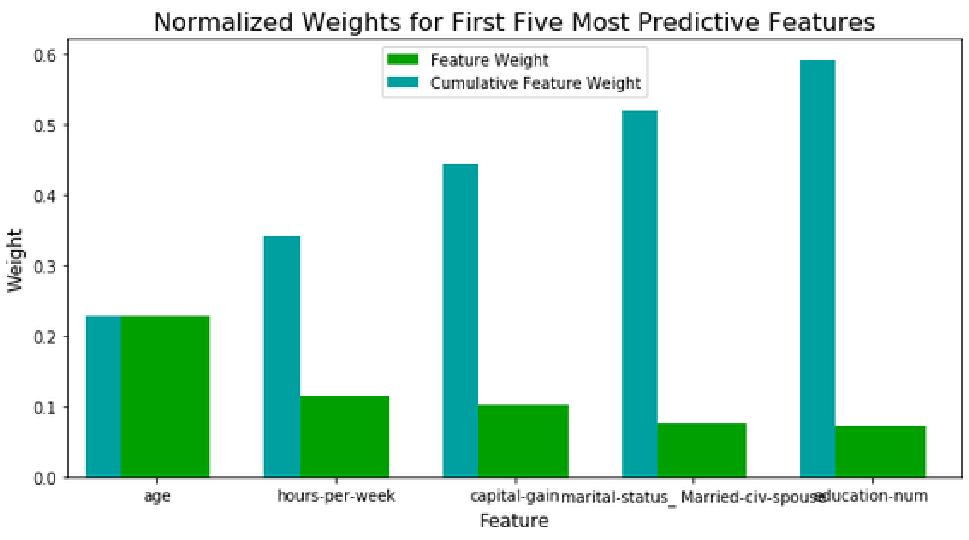
```
Unoptimized model
------
Accuracy score on testing data: 0.8433
F-score on testing data: 0.6848

Optimized Model
------
Final accuracy score on the testing data: 0.8542
Final F-score on the testing data: 0.7214
```

FEATURE IMPORTANCE & SELECTION

When performing supervised learning on a dataset like the census data, It's important to determine which features provide the most predictive power and by focusing on the relationship between only a few crucial features and the target label, I simplify our understanding of the phenomenon, as well as becoming more efficient. In the case of this project, that means I wish to identify a small number of features that most strongly predict whether an individual makes at most or more than $50,000.

Here is a chart built by utilizing the feature_importance_ attribute, which is a function that ranks the importance of features according to the chosen classifier.



Feature Selection

Having less features to train, the expectation is that training and prediction time is much lower — at the cost of performance metrics. From the visualization above, we see that the top five most important features contribute more than half of the importance of **all** features present in the data. This hints that I can attempt to *reduce the feature space* and simplify the information required for the model to learn. The results below showcase the same optimized model from earlier, trained on the same training set with only the top five important features.

```
Final Model trained on full data
------
Accuracy on testing data: 0.8542
F-score on testing data: 0.7214

Final Model trained on reduced data
------
Accuracy on testing data: 0.8479
F-score on testing data: 0.7017
```

RESULTS

The F-score and accuracy have decreased by 1-2%. Depending on the project domain, if training time was a crucial factor, I would consider using the reduced data as my training set as I do not find the difference in performance to be significant.