Machine Learning Engineer Nanodegree

Supervised Learning

Project: Finding Donors for *CharityML*

Welcome to the second project of the Machine Learning Engineer Nanodegree! In this notebook, some template code has already been provided for you, and it will be your job to implement the additional functionality necessary to successfully complete this project. Sections that begin with **'Implementation'** in the header indicate that the following block of code will require additional functionality which you must provide. Instructions will be provided for each section and the specifics of the implementation are marked in the code block with a `'TODO'` statement. Please be sure to read the instructions carefully!

In addition to implementing code, there will be questions that you must answer which relate to the project and your implementation. Each section where you will answer a question is preceded by a **'Question X'** header. Carefully read each question and provide thorough answers in the following text boxes that begin with **'Answer:'**. Your project submission will be evaluated based on your answers to each of the questions and the implementation you provide.

> **Note:** Please specify WHICH VERSION OF PYTHON you are using when submitting this notebook. Code and Markdown cells can be executed using the **Shift + Enter** keyboard shortcut. In addition, Markdown cells can be edited by typically double-clicking the cell to enter edit mode.

Getting Started

In this project, you will employ several supervised algorithms of your choice to accurately model individuals' income using data collected from the 1994 U.S. Census. You will then choose the best candidate algorithm from preliminary results and further optimize this algorithm to best model the data. Your 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 or not 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 (as we will see) infer this value from other publically available features.

The dataset for this project originates from the [UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Census+Income)](https://archive.ics.uci.edu/ml/datasets/Census+Income). The datset was donated by Ron Kohavi and Barry Becker, after being published in the article *"Scaling Up the Accuracy of Naive-Bayes Classifiers: A Decision-Tree Hybrid"*. You can find the article by Ron Kohavi [online (https://www.aaai.org/Papers/KDD/1996/KDD96-033.pdf)](https://www.aaai.org/Papers/KDD/1996/KDD96-033.pdf). The data we investigate here consists of small changes to the original dataset, such as removing the `'fnlwgt'` feature and records with missing or ill-formatted entries.

Exploring the Data

Run the code cell below to load necessary Python libraries and load the census data. Note that the last column from this dataset, `'income'`, will be our target label (whether an individual makes more than, or at most, $50,000 annually). All other columns are features about each individual in the census database.

```python
In [1]:
1  # Import libraries necessary for this project
2  import numpy as np
3  import pandas as pd
4  from time import time
5  from IPython.display import display # Allows the use of display() for DataFrames
6
7  # Import supplementary visualization code visuals.py
8  import visuals as vs
9
10 # Pretty display for notebooks
11 %matplotlib inline
12
13 # Load the Census dataset
14 data = pd.read_csv("census.csv")
15
16 # Success - Display the first record
17 display(data.head(n=1))
```

	age	workclass	education_level	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country	income
0	39	State-gov	Bachelors	13.0	Never-married	Adm-clerical	Not-in-family	White	Male	2174.0	0.0	40.0	United-States	<=50K

Implementation: 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. In the code cell below, you will need to compute the following:

- The total number of records, `'n_records'`
- The number of individuals making more than $50,000 annually, `'n_greater_50k'`.
- The number of individuals making at most $50,000 annually, `'n_at_most_50k'`.
- The percentage of individuals making more than $50,000 annually, `'greater_percent'`.

** HINT: ** You may need to look at the table above to understand how the `'income'` entries are formatted.

```python
# TODO: Total number of records
n_records = len(data)
# data.shape[0] also does the trick

# TODO: Number of records where individual's income is more than $50,000
n_greater_50k = sum(data['income'] == '>50K')

# TODO: Number of records where individual's income is at most $50,000
n_at_most_50k = sum(data['income'] == '<=50K')

# TODO: Percentage of individuals whose income is more than $50,000
greater_percent = 1.0 * n_greater_50k / n_records * 100

# Print the results
print "Total number of records: {}".format(n_records)
print "Individuals making more than $50,000: {}".format(n_greater_50k)
print "Individuals making at most $50,000: {}".format(n_at_most_50k)
print "Percentage of individuals making more than $50,000: {:.2f}%".format(greater_percent)
```

```
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%
```

Some more exploratory data analysis, using factor plots in seaborn.

```python
import seaborn as sns
sns.catplot('sex', col='marital-status', data=data, hue='income', kind='count', col_wrap=4)
```

Out[3]: `<seaborn.axisgrid.FacetGrid at 0xd23cc88>`
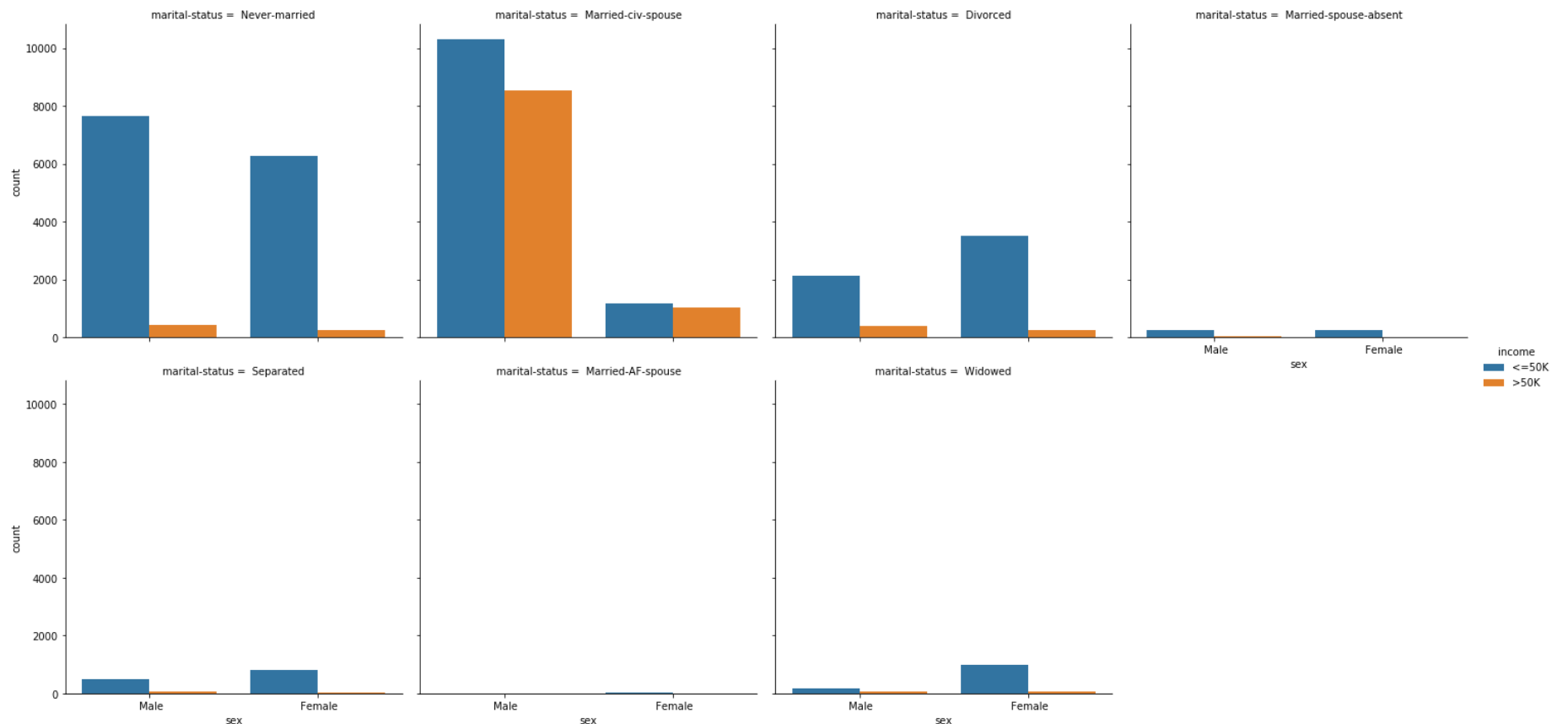
** Featureset Exploration **

- **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-voc, 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, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.


Preparing the Data

Before data can be used as input for machine learning algorithms, it often must be cleaned, formatted, and restructured — this is typically known as **preprocessing**. Fortunately, for this dataset, there are no invalid or missing entries we must deal with, however, there are some qualities about certain features that must be adjusted. This preprocessing can help tremendously with the outcome and predictive power of nearly all learning algorithms.

Transforming Skewed Continuous Features

A dataset may sometimes contain at least one feature whose values tend to lie near a single number, but will also have a non-trivial number of vastly larger or smaller values than that single number. Algorithms can be sensitive to such 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'`.

Run the code cell below to plot a histogram of these two features. Note the range of the values present and how they are distributed.

```
In [4]:   1  # Split the data into features and target label
          2  income_raw = data['income']
          3  features_raw = data.drop('income', axis = 1)
          4
          5  # Visualize skewed continuous features of original data
          6  vs.distribution(data)
```
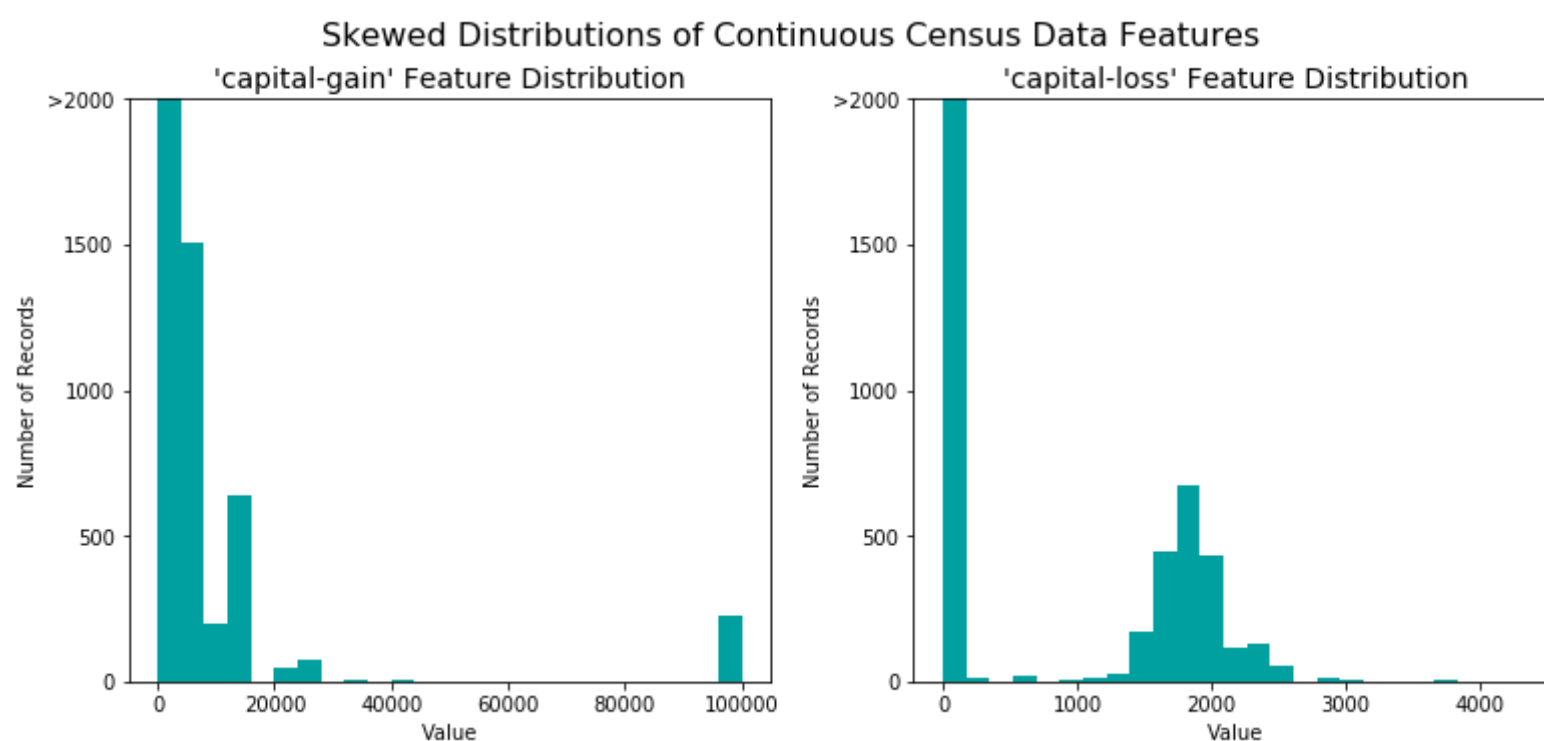

For highly-skewed feature distributions such as `'capital-gain'` and `'capital-loss'`, it is common practice to apply a [logarithmic transformation (https://en.wikipedia.org/wiki/Data_transformation_(statistics))](https://en.wikipedia.org/wiki/Data_transformation_(statistics)) on the data so that the 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 however: The logarithm of `0` is undefined, so we must translate the values by a small amount above `0` to apply the the logarithm successfully.

Run the code cell below to perform a transformation on the data and visualize the results. Again, note the range of values and how they are distributed.

```python
# Log-transform the skewed features
skewed = ['capital-gain', 'capital-loss']
features_log_transformed = pd.DataFrame(data = features_raw).copy()

print 'Before log transformation'
display(features_log_transformed.head())
#display(features_raw.head()[skewed])
features_log_transformed[skewed] = features_raw[skewed].apply(lambda x: np.log(x + 1))
#display(features_raw.head()[skewed])

print 'After log transformation'
display(features_log_transformed.head())

# Visualize the new log distributions
vs.distribution(features_log_transformed, transformed = True)
```

Before log transformation

	age	workclass	education_level	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country
0	39	State-gov	Bachelors	13.0	Never-married	Adm-clerical	Not-in-family	White	Male	2174.0	0.0	40.0	United-States
1	50	Self-emp-not-inc	Bachelors	13.0	Married-civ-spouse	Exec-managerial	Husband	White	Male	0.0	0.0	13.0	United-States
2	38	Private	HS-grad	9.0	Divorced	Handlers-cleaners	Not-in-family	White	Male	0.0	0.0	40.0	United-States
3	53	Private	11th	7.0	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0.0	0.0	40.0	United-States
4	28	Private	Bachelors	13.0	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0.0	0.0	40.0	Cuba

After log transformation

	age	workclass	education_level	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country
0	39	State-gov	Bachelors	13.0	Never-married	Adm-clerical	Not-in-family	White	Male	7.684784	0.0	40.0	United-States
1	50	Self-emp-not-inc	Bachelors	13.0	Married-civ-spouse	Exec-managerial	Husband	White	Male	0.000000	0.0	13.0	United-States
2	38	Private	HS-grad	9.0	Divorced	Handlers-cleaners	Not-in-family	White	Male	0.000000	0.0	40.0	United-States
3	53	Private	11th	7.0	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0.000000	0.0	40.0	United-States
4	28	Private	Bachelors	13.0	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0.000000	0.0	40.0	Cuba

Log-transformed Distributions of Continuous Census Data Features
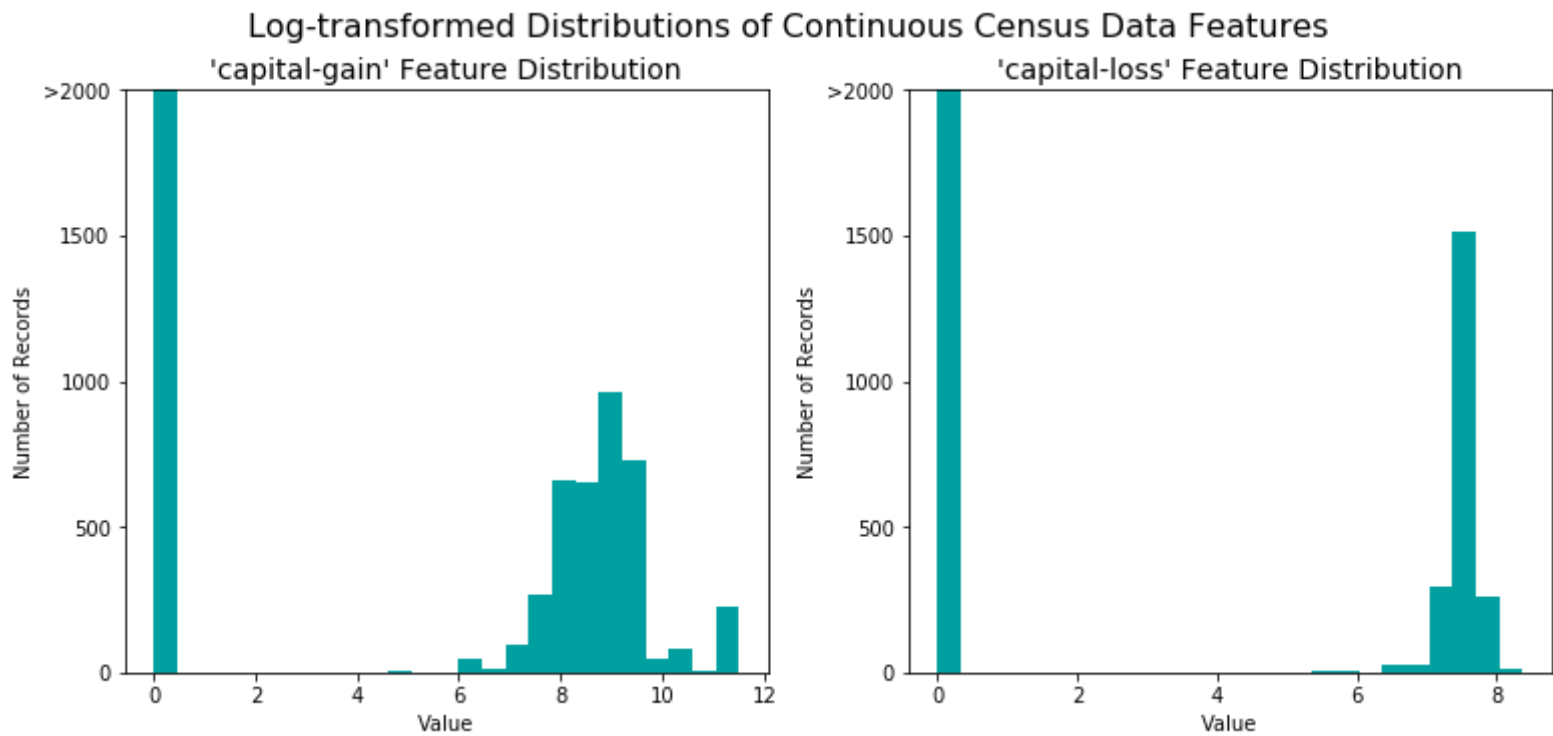
Normalizing Numerical Features

In addition to performing transformations on features that are highly skewed, it is often good practice to perform some type of scaling on numerical features. Applying a 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. Note that once scaling is applied, observing the data in its raw form will no longer have the same original meaning, as exampled below.

Run the code cell below to normalize each numerical feature. We will use [sklearn.preprocessing.MinMaxScaler](http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html) for this.

```
In [6]:   1  # Import sklearn.preprocessing.StandardScaler
          2  from sklearn.preprocessing import MinMaxScaler
          3
          4  # Initialize a scaler, then apply it to the features
          5  scaler = MinMaxScaler() # default=(0, 1)
          6  numerical = ['age', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week']
          7
          8  #display(features_log_transformed.head())
          9
         10  features_log_minmax_transform = pd.DataFrame(data = features_log_transformed)
         11  features_log_minmax_transform[numerical] = scaler.fit_transform(features_log_transformed[numerical])
         12
         13  # Show an example of a record with scaling applied
         14  display(features_log_minmax_transform.head(n = 5))
```

C:\Users\Xiaowei\Anaconda3\envs\mlnd\lib\site-packages\sklearn\preprocessing\data.py:334: DataConversionWarning: Data with input dtype int64, float64 were all converted to float64 by MinMaxScaler.
  return self.partial_fit(X, y)

	age	workclass	education_level	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	native-country
0	0.301370	State-gov	Bachelors	0.800000	Never-married	Adm-clerical	Not-in-family	White	Male	0.667492	0.0	0.397959	United-States
1	0.452055	Self-emp-not-inc	Bachelors	0.800000	Married-civ-spouse	Exec-managerial	Husband	White	Male	0.000000	0.0	0.122449	United-States
2	0.287671	Private	HS-grad	0.533333	Divorced	Handlers-cleaners	Not-in-family	White	Male	0.000000	0.0	0.397959	United-States
3	0.493151	Private	11th	0.400000	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0.000000	0.0	0.397959	United-States
4	0.150685	Private	Bachelors	0.800000	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0.000000	0.0	0.397959	Cuba

Implementation: Data Preprocessing

From the table in **Exploring the Data** above, we can see there are several features for each record that are non-numeric. Typically, 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 ---->	1	0	0	1
2	A		1	0	0

Additionally, as with the non-numeric features, we 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()](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html?highlight=get_dummies#pandas.get_dummies) to perform one-hot encoding on the `'features_raw'` data.
- Convert the target label `'income_raw'` to numerical entries.
  - Set records with "<=50K" to `0` and records with ">50K" to `1`.

```
In [7]:     1   # TODO: One-hot encode the 'features_log_minmax_transform' data using pandas.get_dummies()
            2   features_final = pd.get_dummies(features_log_minmax_transform)
            3
            4   # TODO: Encode the 'income_raw' data to numerical values
            5   f = lambda x: 0 if x=='<=50K' else 1
            6   income = income_raw.apply(f)
            7
            8   ## Another way of encoding the income_raw is by using fit_transform()
            9   #from sklearn.preprocessing import LabelEncoder
           10   #le = LabelEncoder()
           11   #income = le.fit_transform(income_raw)
           12
           13   # if needed we can reverse the transform
           14   #print le.inverse_transform(income)
           15
           16   # Print the number of features after one-hot encoding
           17   encoded = list(features_final.columns)
           18   print "{} total features after one-hot encoding.".format(len(encoded))
           19
           20   # Uncomment the following line to see the encoded feature names
           21   print encoded
```

1st-4th', 'education_level_ 5th-6th', 'education_level_ 7th-8th', 'education_level_ 9th', 'education_level_ Assoc-ac
dm', 'education_level_ Assoc-voc', 'education_level_ Bachelors', 'education_level_ Doctorate', 'education_level_ HS-
grad', 'education_level_ Masters', 'education_level_ Preschool', 'education_level_ Prof-school', 'education_level_ S
ome-college', 'marital-status_ Divorced', 'marital-status_ Married-AF-spouse', 'marital-status_ Married-civ-spouse',
'marital-status_ Married-spouse-absent', 'marital-status_ Never-married', 'marital-status_ Separated', 'marital-stat
us_ Widowed', 'occupation_ Adm-clerical', 'occupation_ Armed-Forces', 'occupation_ Craft-repair', 'occupation_ Exec-
managerial', 'occupation_ Farming-fishing', 'occupation_ Handlers-cleaners', 'occupation_ Machine-op-inspct', 'occup
ation_ Other-service', 'occupation_ Priv-house-serv', 'occupation_ Prof-specialty', 'occupation_ Protective-serv',
'occupation_ Sales', 'occupation_ Tech-support', 'occupation_ Transport-moving', 'relationship_ Husband', 'relations
hip_ Not-in-family', 'relationship_ Other-relative', 'relationship_ Own-child', 'relationship_ Unmarried', 'relation
ship_ Wife', 'race_ Amer-Indian-Eskimo', 'race_ Asian-Pac-Islander', 'race_ Black', 'race_ Other', 'race_ White', 's
ex_ Female', 'sex_ Male', 'native-country_ Cambodia', 'native-country_ Canada', 'native-country_ China', 'native-cou
ntry_ Columbia', 'native-country_ Cuba', 'native-country_ Dominican-Republic', 'native-country_ Ecuador', 'native-co
untry_ El-Salvador', 'native-country_ England', 'native-country_ France', 'native-country_ Germany', 'native-country
_ Greece', 'native-country_ Guatemala', 'native-country_ Haiti', 'native-country_ Holand-Netherlands', 'native-count
ry_ Honduras', 'native-country_ Hong', 'native-country_ Hungary', 'native-country_ India', 'native-country_ Iran',
'native-country_ Ireland', 'native-country_ Italy', 'native-country_ Jamaica', 'native-country_ Japan', 'native-coun
try_ Laos', 'native-country_ Mexico', 'native-country_ Nicaragua', 'native-country_ Outlying-US(Guam-USVI-etc)', 'na
tive-country_ Peru', 'native-country_ Philippines', 'native-country_ Poland', 'native-country_ Portugal', 'native-co
untry_ Puerto-Rico', 'native-country_ Scotland', 'native-country_ South', 'native-country_ Taiwan', 'native-country_

Shuffle and Split Data

Now all *categorical variables* have been converted into numerical features, and all numerical features have been normalized. As always, we will now 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.

Run the code cell below to perform this split.

```
In [8]:     1   # Import train_test_split
            2   from sklearn.model_selection import train_test_split
            3
            4   # Split the 'features' and 'income' data into training and testing sets
            5   X_train, X_test, y_train, y_test = train_test_split(features_final,
            6                                                       income,
            7                                                       test_size = 0.2,
            8                                                       random_state = 0)
            9
           10   # Show the results of the split
           11   print "Training set has {} samples.".format(X_train.shape[0])
           12   print "Testing set has {} samples.".format(X_test.shape[0])
```

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

Evaluating Model Performance

In this section, we will investigate four different algorithms, and determine which is best at modeling the data. Three of these algorithms will be supervised learners of your choice, and the fourth algorithm is known as a *naive predictor*.

Metrics and the Naive Predictor

CharityML, equipped with their research, 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. It would seem that using **accuracy** as a metric for evaluating a particular model's performace 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. We can use **F-beta score** as a metric that considers both precision and recall:

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

In particular, when $\beta = 0.5$, more emphasis is placed on precision. This is called the $\mathbf{F_{0.5}}$ **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* for your data, to help establish a benchmark for whether a model is performing well. That been said, using that prediction would be pointless: If we predicted all people made less than $50,000, *CharityML* would identify no one as donors.

Note: Recap of accuracy, precision, recall

** Accuracy ** measures how often the classifier makes the correct prediction. It's the ratio of the number of correct predictions to the total number of predictions (the number of test data points).

** Precision ** tells us what proportion of messages we classified as spam, actually were spam. It is a ratio of true positives(words classified as spam, and which are actually spam) to all positives(all words classified as spam, irrespective of whether that was the correct classificatio), in other words it is the ratio of

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

** Recall(sensitivity)** tells us what proportion of messages that actually were spam were classified by us as spam. It is a ratio of true positives(words classified as spam, and which are actually spam) to all the words that were actually spam, in other words it is the ratio of

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

For classification problems that are skewed in their classification distributions like in our case, for example if we had a 100 text messages and only 2 were spam and the rest 98 weren't, accuracy by itself is not a very good metric. We could classify 90 messages as not spam(including the 2 that were spam but we classify them as not spam, hence they would be false negatives) and 10 as spam(all 10 false positives) and still get a reasonably good accuracy score. For such cases, precision and recall come in very handy. These two metrics can be combined to get the F1 score, which is weighted average(harmonic mean) of the precision and recall scores. This score can range from 0 to 1, with 1 being the best possible F1 score(we take the harmonic mean as we are dealing with ratios).

Question 1 - Naive Predictor Performace

- If we chose a model that always predicted an individual made more than $50,000, what would that model's accuracy and F-score be on this dataset? You must use the code cell below and assign your results to `'accuracy'` and `'fscore'` to be used later.

** HINT: **

- 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.
- Our Recall score(True Positives/(True Positives + False Negatives)) in this setting becomes 1 as we have no False Negatives.

```
In [9]:
1
2   TP = np.sum(income) # Counting the ones as this is the naive case. Note that 'income' is the 'income_raw' data
3   #encoded to numerical values done in the data preprocessing step.
4   FP = income.count() - TP # Specific to the naive case
5
6   TN = 0 # No predicted negatives in the naive case
7   FN = 0 # No predicted negatives in the naive case
8
9   # TODO: Calculate accuracy, precision and recall
10  accuracy = 1.0 * (TP + TN) / (TP + TN + FP + FN)
11  recall = 1.0 * TP/(TP + FN)
12  precision = 1.0 * TP/(TP + FP)
13
14  # TODO: Calculate F-score using the formula above for beta = 0.5 and correct values for precision and recall.
15  # HINT: The formula above can be written as (1 + beta**2) * (precision * recall) / ((beta**2 * precision) + recall)
16  beta = 0.5
17  fscore = (1 + beta**2) * (precision * recall) / ((beta**2 * precision) + recall)
18
19  # Print the results
20  print "Naive Predictor: [Accuracy score: {:.4f}, F-score: {:.4f}]".format(accuracy, fscore)
```

```
Naive Predictor: [Accuracy score: 0.2478, F-score: 0.2917]
```

Supervised Learning Models

The following are some of the supervised learning models that are currently available in `scikit-learn` (http://scikit-learn.org/stable/supervised_learning.html) **that you may choose from:**

- Gaussian Naive Bayes (GaussianNB)
- Decision Trees
- Ensemble Methods (Bagging, AdaBoost, Random Forest, Gradient Boosting)

- K-Nearest Neighbors (KNeighbors)
- Stochastic Gradient Descent Classifier (SGDC)
- Support Vector Machines (SVM)
- Logistic Regression

Question 2 - Model Application

List three of the supervised learning models above that are appropriate for this problem that you will test on the census data. For each model chosen

- Describe one real-world application in industry where the model can be applied.
- What are the strengths of the model; when does it perform well?
- What are the weaknesses of the model; when does it perform poorly?
- What makes this model a good candidate for the problem, given what you know about the data?

** HINT: **

Structure your answer in the same format as above^, with 4 parts for each of the three models you pick. Please include references with your answer.

*Answer: * Instead of 3, I picked 6 models. It forced me to think and also do some research.

1. Gaussian Naive Bayes (GaussianNB)
   A. Determine whether an email is spam by checking the word frequency in the email.
   B. It's easy to compute as just need to count to get the probability distribution. It performes well when the attributes are discrete and conditionally independent.
   C. The model relies on the fact that the attributes are conditionally independent and discrete. It performs poorly when the attributes are not independent.
   D. In this problem, GaussianNB may not perform well as some of the features are correlated, by common sense.
2. Decision Trees
   A. Based on factors of season, geolocation, time to determine whehter it's cold or hot.
   B. Decision tree is extremely easy to model and explain. It performs well for almost all the classification problems.
   C. It easily gets into overfit and local minima. It performs poorly when such problems happen.
   D. Decision tree is capable to handle both numerical and categorical data, and that are the features we are dealing with.
3. Ensemble Methods
   A. Use the data from wearable devices to predict user's activity level (walk, run, sit or etcs).
   B. It solves the overfit issue from decision trees and performs well for almost all the classification problems.
   C. The ensemble methods leverage on the majority voting from different weak learners. If there are dependencies between the weak learners, it may not perform well.
   D. There are many features in the problem, they contribute different "weights" to the accuracy of prediction. A single model can have a "good" prediction accuracy, but may not be "good" enough. Ensemble methods will address this issue and push the accuracy as close to "perfect".
4. SGDC
   A. SGD is quite widely used in text mining or natural language processing.
   B. It's is very powerful and scalable. It perform extremely well when the training data and dimensions are big.
   C. It's sensitive to feature scaling. The data needs preprecessing before fitting into the model.
   D. In this project, the data was preprocessing by using MinMaxScaler, not StandardScaler as favored by SGD. But I think it will still perform well as we've still gone through a "normalization" to preprocess the data.
5. SVM
   A. Determine the sentiment from text.
   B. SVM is able to use both linear and non linear kernels. It performs well on both low or high dimension space.
   C. However, SVM can be inefficient if feature dimensions are high or there are lots of data.
   D. In this problem, we are only dealing with slightly more than 10 features, SVM should be able to perfom well.
6. Logistic Regression
   A. Weather forcasting.
   B. The logistic regression applies to almost all kinds of classification problems, regardless the feature space is linearly seperable or not.
   C. It is suitable for classification problems, not for regression problems need continous output.
   D. In this project, the predicated output is binary, logistic regression will be a good choice. Moreover, the predicted output is a probability, it tells more than just binary output.

Implementation - Creating a Training and Predicting Pipeline

To properly evaluate the performance of each model you've chosen, it's important that you create a training and predicting pipeline that allows you to quickly and effectively train models using various sizes of training data and perform predictions on the testing data. Your implementation here will be used in the following section. In the code block below, you will need to implement the following:

- Import `fbeta_score` and `accuracy_score` from `sklearn.metrics` (http://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics).
- Fit the learner to the sampled training data and record the training time.
- Perform predictions on the test data `X_test`, and also 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.
  - Make sure that you set the `beta` parameter!

```python
In [10]:   1  # TODO: Import two metrics from sklearn - fbeta_score and accuracy_score
           2  from sklearn.metrics import fbeta_score, accuracy_score
           3  def train_predict(learner, sample_size, X_train, y_train, X_test, y_test):
           4      '''
           5      inputs:
           6         - learner: the learning algorithm to be trained and predicted on
           7         - sample_size: the size of samples (number) to be drawn from training set
           8         - X_train: features training set
           9         - y_train: income training set
          10         - X_test: features testing set
          11         - y_test: income testing set
          12      '''
          13
          14      results = {}
          15
          16      # TODO: Fit the learner to the training data using slicing with 'sample_size' using .fit(training_features[:], t
          17      start = time() # Get start time
          18      learner = learner.fit(X_train[:sample_size], y_train[:sample_size])
          19      end = time() # Get end time
          20
          21      # TODO: Calculate the training time
          22      results['train_time'] = end - start
          23
          24      # TODO: Get the predictions on the test set(X_test),
          25      #       then get predictions on the first 300 training samples(X_train) using .predict()
          26      start = time() # Get start time
          27      predictions_test = learner.predict(X_test[:sample_size])
          28      predictions_train = learner.predict(X_train[:sample_size][:300])
          29      end = time() # Get end time
          30
          31      # TODO: Calculate the total prediction time
          32      results['pred_time'] = end - start
          33
          34      # TODO: Compute accuracy on the first 300 training samples which is y_train[:300]
          35      results['acc_train'] = accuracy_score(y_train[:sample_size][:300], predictions_train)
          36
          37      # TODO: Compute accuracy on test set using accuracy_score()
          38      results['acc_test'] = accuracy_score(y_test[:sample_size], predictions_test)
          39
          40      # TODO: Compute F-score on the the first 300 training samples using fbeta_score()
          41      results['f_train'] = fbeta_score(y_train[:sample_size][:300], predictions_train, beta)
          42
          43      # TODO: Compute F-score on the test set which is y_test
          44      results['f_test'] = fbeta_score(y_test[:sample_size], predictions_test, beta)
          45
          46      # Success
          47      print "{} trained on {} samples.".format(learner.__class__.__name__, sample_size)
          48
          49      # Return the results
          50      return results
```

Implementation: Initial Model Evaluation

In the code cell, you will need to implement the following:

- Import the three supervised learning models you've 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 you use, if provided.
  - **Note:** Use the default settings for each model — you will tune one specific model in a later section.
- 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.

Note: Depending on which algorithms you chose, the following implementation may take some time to run!

```
In [11]:  1  # TODO: Import the three supervised learning models from sklearn
          2  from sklearn.ensemble import RandomForestClassifier
          3  #from sklearn.svm import SVC
          4  from sklearn.linear_model import LogisticRegression, SGDClassifier
          5  #from sklearn.naive_bayes import GaussianNB
          6
          7  # TODO: Initialize the three models
          8  clf_A = RandomForestClassifier(random_state = 0)
          9  clf_B = SGDClassifier(max_iter=100, random_state = 0)
         10  clf_C = LogisticRegression(random_state = 0)
         11  #clf_D = GaussianNB()
         12
         13  # TODO: Calculate the number of samples for 1%, 10%, and 100% of the training data
         14  # HINT: samples_100 is the entire training set i.e. len(y_train)
         15  # HINT: samples_10 is 10% of samples_100
         16  # HINT: samples_1 is 1% of samples_100
         17  samples_100 = len(y_train)
         18  samples_10 = int(0.1 * len(y_train))
         19  samples_1 = int(0.01 * len(y_train))
         20
         21  # Collect results on the learners
         22  results = {}
         23  for clf in [clf_A, clf_B, clf_C]:
         24      clf_name = clf.__class__.__name__
         25      results[clf_name] = {}
         26      for i, samples in enumerate([samples_1, samples_10, samples_100]):
         27          results[clf_name][i] = \
         28              train_predict(clf, samples, X_train, y_train, X_test, y_test)
         29
         30  # Display the training results in data frame
         31  for i in results.items():
         32      print '\n' + i[0]
         33      display(pd.DataFrame(i[1]).round(4).rename(columns={0:'1%', 1:'10%', 2:'100%'}))
         34
         35  # Run metrics visualization for the three supervised learning models chosen
         36  vs.evaluate(results, accuracy, fscore)
```

C:\Users\Xiaowei\Anaconda3\envs\mlnd\lib\site-packages\sklearn\ensemble\forest.py:246: FutureWarning: The default value of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)

RandomForestClassifier trained on 361 samples.
RandomForestClassifier trained on 3617 samples.
RandomForestClassifier trained on 36177 samples.
SGDClassifier trained on 361 samples.
SGDClassifier trained on 3617 samples.

C:\Users\Xiaowei\Anaconda3\envs\mlnd\lib\site-packages\sklearn\linear_model\stochastic_gradient.py:183: FutureWarning: max_iter and tol parameters have been added in SGDClassifier in 0.19. If max_iter is set but tol is left unset, the default value for tol in 0.19 and 0.20 will be None (which is equivalent to -infinity, so it has no effect) but will change in 0.21 to 1e-3. Specify tol to silence this warning.
  FutureWarning)

SGDClassifier trained on 36177 samples.
LogisticRegression trained on 361 samples.
LogisticRegression trained on 3617 samples.

C:\Users\Xiaowei\Anaconda3\envs\mlnd\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)

LogisticRegression trained on 36177 samples.

LogisticRegression

	1%	10%	100%
acc_test	0.8338	0.8402	0.8419
acc_train	0.8600	0.8467	0.8467
f_test	0.6364	0.6655	0.6832
f_train	0.7386	0.7031	0.6985
pred_time	0.0020	0.0040	0.0070
train_time	0.0050	0.0150	0.4190

SGDClassifier

	1%	10%	100%
acc_test	0.7867	0.8377	0.8402
acc_train	0.8633	0.8533	0.8567
f_test	0.5478	0.6627	0.6822
f_train	0.7088	0.7292	0.7308

	1%	10%	100%
pred_time	0.0030	0.0040	0.0070
train_time	0.0090	0.0600	1.4450

```
RandomForestClassifier
```

	1%	10%	100%
acc_test	0.7812	0.8305	0.8377
acc_train	0.9767	0.9800	0.9567
f_test	0.4874	0.6402	0.6716
f_train	0.9789	0.9738	0.9337
pred_time	0.0050	0.0120	0.0330
train_time	0.0240	0.0480	0.5430

Performance Metrics for Three Supervised Learning Models
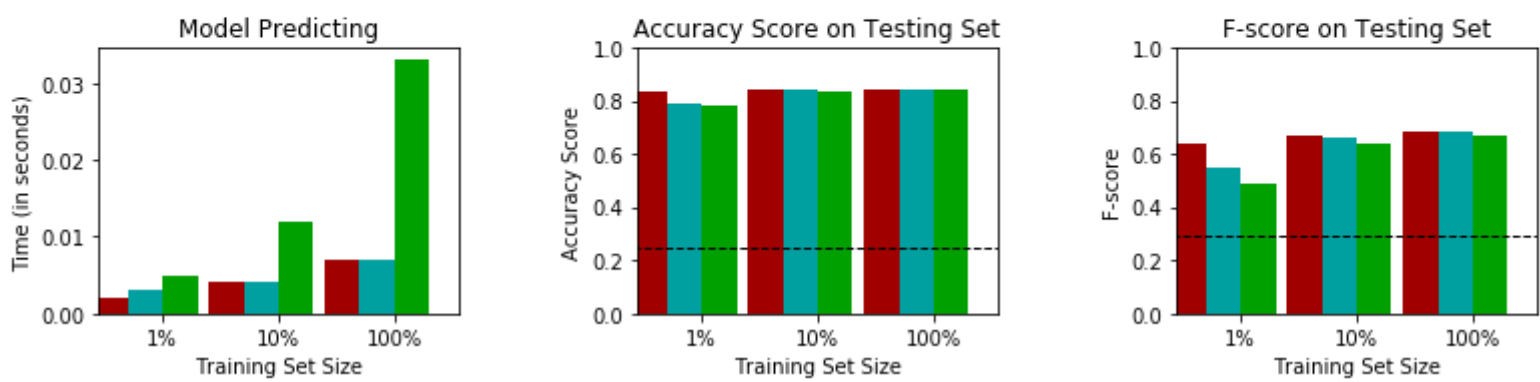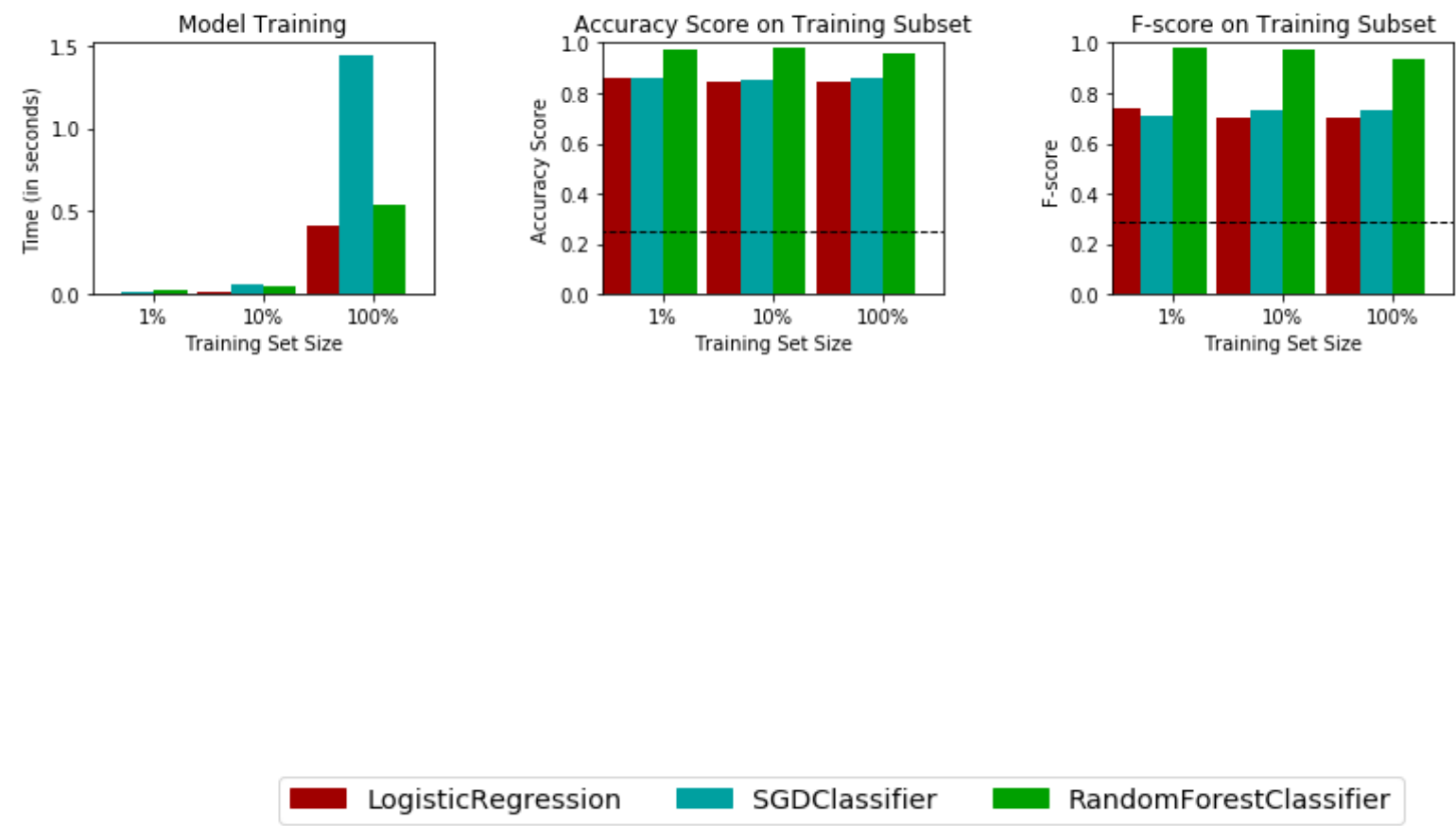
Improving Results

In this final section, you will choose from the three supervised learning models the *best* model to use on the student data. You will then perform a grid search optimization for the model over the entire training set (`X_train` and `y_train`) by tuning at least one parameter to improve upon the untuned model's F-score.

Question 3 - Choosing the Best Model

- Based on the evaluation you performed earlier, in one to two paragraphs, explain to *CharityML* which of the three models you believe to be most appropriate for the task of identifying individuals that make more than $50,000.

** HINT: ** Look at the graph at the bottom left from the cell above(the visualization created by `vs.evaluate(results, accuracy, fscore)`) and check the F score for the testing set when 100% of the training set is used. Which model has the highest score? Your answer should include discussion of the:

- metrics - F score on the testing when 100% of the training data is used,
- prediction/training time
- the algorithm's suitability for the data.

*Answer: * In this project I will choose logistic regression, reasons as follow.

1. Depite RandomForest produced the highest accuracy and F score in the training data set, but that's the testing against training set. It doesn't represent its performance in "unseen" data.
2. On the training data set, logistic regression always perform the best, regardless the sample size. This tells us even if we have limited amount of data, logistic regression still performs better than the others.
3. Logistic regression takes the least amount of time on training and predicting. This make it scalable and fast in real world.

Question 4 - Describing the Model in Layman's Terms

- In one to two paragraphs, explain to *CharityML*, in layman's terms, how the final model chosen is supposed to work. Be sure that you are describing the major qualities of the model, such as how the model is trained and how the model makes a prediction. Avoid using advanced mathematical jargon, such as describing equations.

** HINT: **

When explaining your model, if using external resources please include all citations.

*Answer: * The logistic regression starts with assigning random weights to the different features of the training data. A sum was performed against all the weight and feature pasirs. The value of sum will determine which class the training sample belongs to. If the predication differs from the truth, then the random weights will be fine tuned by either increasing or decreasing to make the predication "closer" to the truth. The value of the sum is continous but the expected output is binary. The nice thing about this logistic regression is the value of sum will be passed to the logistic function which outputs the result as a probability between 0 and 1. If the value of sum is 0, the output will be 0.5, which means half chance "<=50K" and half chance ">50k". If the value of sum is negative, the output will be smaller than 0.5, which means there is less chance "<=50K". So on and so forth when the value of sum is positive. When we know how to calculate the predication result, compare and fine tune the weights, we just doing it iteratively until the result converges, in which we cannot do any better if continuing the iteration.

Implementation: Model Tuning

Fine tune the chosen model. Use grid search (`GridSearchCV`) with at least one important parameter tuned with at least 3 different values. You will need to use the entire training set for this. In the code cell below, you will need to implement the following:

- Import `sklearn.grid_search.GridSearchCV` (http://scikit-learn.org/0.17/modules/generated/sklearn.grid_search.GridSearchCV.html) and `sklearn.metrics.make_scorer` (http://scikit-learn.org/stable/modules/generated/sklearn.metrics.make_scorer.html).
- Initialize the classifier you've chosen and store it in `clf` .
  - Set a `random_state` if one is available to the same state you set before.
- Create a dictionary of parameters you wish to tune for the chosen model.
  - Example: `parameters = {'parameter' : [list of values]}` .
  - **Note:** Avoid tuning the `max_features` parameter of your learner if that parameter is available!
- Use `make_scorer` to create an `fbeta_score` scoring object (with $\beta = 0.5$).
- Perform grid search on the classifier `clf` using the `'scorer'` , and store it in `grid_obj` .
- Fit the grid search object to the training data (`X_train` , `y_train`), and store it in `grid_fit` .

Note: Depending on the algorithm chosen and the parameter list, the following implementation may take some time to run!

```
In [12]:   1  %%time
           2  # TODO: Import 'GridSearchCV', 'make_scorer', and any other necessary libraries
           3  from sklearn.model_selection import GridSearchCV
           4  from sklearn.metrics import make_scorer
           5
           6  # TODO: Initialize the classifier
           7  clf = LogisticRegression(random_state = 0, max_iter = 100)
           8
           9  # TODO: Create the parameters list you wish to tune, using a dictionary if needed.
          10  # HINT: parameters = {'parameter_1': [value1, value2], 'parameter_2': [value1, value2]}
          11  parameters = {'penalty':['l1', 'l2'], 'max_iter':[50,200,500]}
          12
          13  # TODO: Make an fbeta_score scoring object using make_scorer()
          14  scorer = make_scorer(fbeta_score, beta = 0.5)
          15
          16  # TODO: Perform grid search on the classifier using 'scorer' as the scoring method using GridSearchCV()
          17  grid_obj = GridSearchCV(clf, parameters, scoring = scorer)
          18
          19  # TODO: Fit the grid search object to the training data and find the optimal parameters using fit()
          20  grid_fit = grid_obj.fit(X_train, y_train)
          21
          22  # Get the estimator
          23  best_clf = grid_fit.best_estimator_
          24  print best_clf
          25
          26  # Make predictions using the unoptimized and model
          27  predictions = (clf.fit(X_train, y_train)).predict(X_test)
          28  best_predictions = best_clf.predict(X_test)
          29
          30  # Report the before-and-afterscores
          31  print "Unoptimized model\n------"
          32  print "Accuracy score on testing data: {:.4f}".format(accuracy_score(y_test, predictions))
          33  print "F-score on testing data: {:.4f}".format(fbeta_score(y_test, predictions, beta = 0.5))
          34  print "\nOptimized Model\n------"
          35  print "Final accuracy score on the testing data: {:.4f}".format(accuracy_score(y_test, best_predictions))
          36  print "Final F-score on the testing data: {:.4f}".format(fbeta_score(y_test, best_predictions, beta = 0.5))
```

```
C:\Users\Xiaowei\Anaconda3\envs\mlnd\lib\site-packages\sklearn\model_selection\_split.py:2053: FutureWarning: You shoul
d specify a value for 'cv' instead of relying on the default value. The default value will change from 3 to 5 in versio
n 0.22.
  warnings.warn(CV_WARNING, FutureWarning)

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=50, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=0, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
Unoptimized model
------
Accuracy score on testing data: 0.8419
F-score on testing data: 0.6832

Optimized Model
------
Final accuracy score on the testing data: 0.8419
Final F-score on the testing data: 0.6832
Wall time: 1min 16s
```

Question 5 - Final Model Evaluation

- What is your optimized model's accuracy and F-score on the testing data?
- Are these scores better or worse than the unoptimized model?
- How do the results from your optimized model compare to the naive predictor benchmarks you found earlier in **Question 1**?_

Note: Fill in the table below with your results, and then provide discussion in the **Answer** box.

Results:

Metric	Benchmark Predictor	Unoptimized Model	Optimized Model
Accuracy Score	0.2478	0.8418	0.8419
F-score	0.2917	0.6829	0.6832

Answer: I have tried different combinations of parameters and found there is very very little improvment from unoptimized model to optimized model. Maybe it's because the default value of the model is already very close to the optimal one. Exploration on feature selection could be further explored.

Feature Importance

An important task when performing supervised learning on a dataset like the census data we study here is determining which features provide the most predictive power. By focusing on the relationship between only a few crucial features and the target label we simplify our understanding of the phenomenon, which is most always a useful thing to do. In the case of this project, that means we wish to identify a small number of features that most strongly predict whether an individual makes at most or more than $50,000.

Choose a scikit-learn classifier (e.g., adaboost, random forests) that has a `feature_importance_` attribute, which is a function that ranks the importance of features according to the chosen classifier. In the next python cell fit this classifier to training set and use this attribute to determine the top 5 most important features for the census dataset.

Question 6 - Feature Relevance Observation

When **Exploring the Data**, it was shown there are thirteen available features for each individual on record in the census data. Of these thirteen records, which five features do you believe to be most important for prediction, and in what order would you rank them and why?

Answer: I will order the five features as follows. Occupation, education_level, capital_gain, age, relationship. Certain occupations have higher income by its nature, like lawyer, doctor and etcs. Despite the data in this project doesn't have such categories, I still want to order it as the first one. I have difficuly to choose which one come next between "education_level" and "capital_gain". Just made a bet that "education_level" should be more significant. It's common sense that if one is more senior, he/she has a higher chance to earn more. So "age" comes then. The last one in the five features is "relationship". A good and healthy relationship between couples leads to high efficiency and productivity at work.
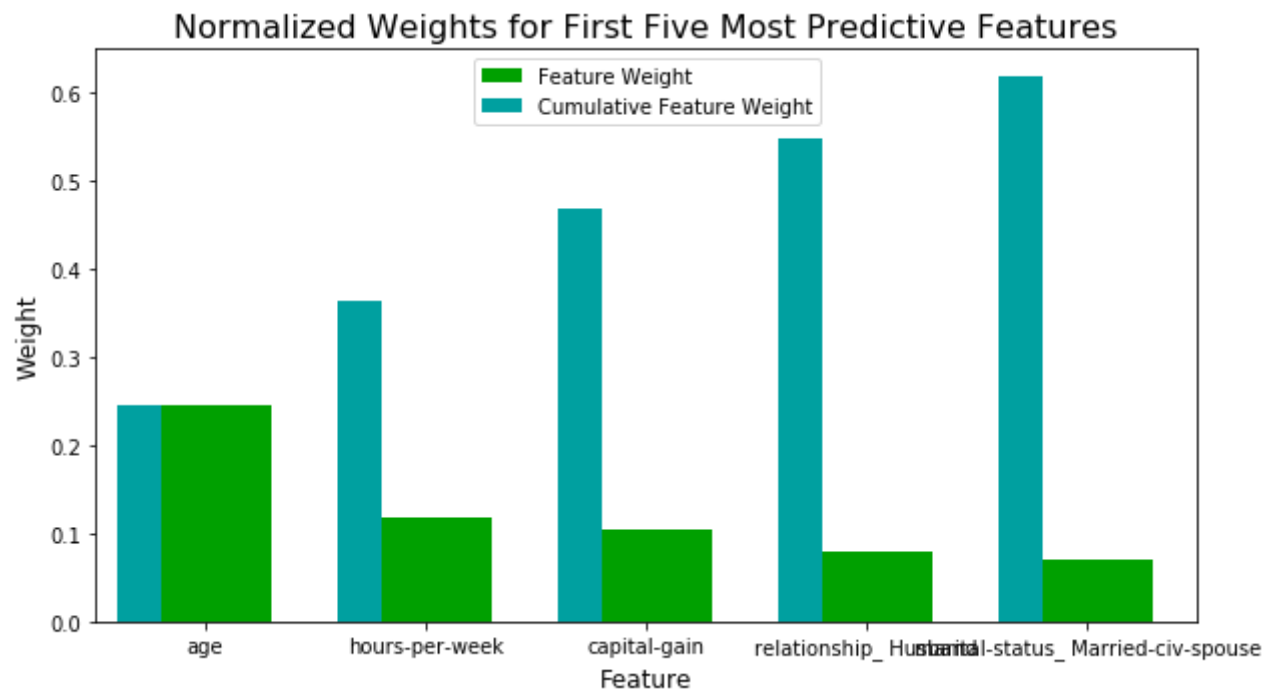
Implementation - Extracting Feature Importance

Choose a `scikit-learn` supervised learning algorithm that has a `feature_importance_` attribute availble for it. This attribute is a function that ranks the importance of each feature when making predictions based on the chosen algorithm.

In the code cell below, you will need to implement the following:

- Import a supervised learning model from sklearn if it is different from the three used earlier.
- Train the supervised model on the entire training set.
- Extract the feature importances using `'.feature_importances_'`.

```
In [13]:
# TODO: Import a supervised learning model that has 'feature_importances_'
from sklearn.ensemble import RandomForestClassifier

# TODO: Train the supervised model on the training set using .fit(X_train, y_train)
model = RandomForestClassifier(random_state = 0)
model.fit(X_train, y_train)

# TODO: Extract the feature importances using .feature_importances_
importances = model.feature_importances_

#print importances
#print X_train.head()
# Plot
vs.feature_plot(importances, X_train, y_train)
```


Question 7 - Extracting Feature Importance

Observe the visualization created above which displays the five most relevant features for predicting if an individual makes at most or above $50,000.

- How do these five features compare to the five features you discussed in **Question 6**?
- If you were close to the same answer, how does this visualization confirm your thoughts?
- If you were not close, why do you think these features are more relevant?

Answer: I got 3 correct, 1 partial correct (education_levcel vs education-num). The visualization showed the top 5 most important features. Age,

capital-gain and relationship are as expected. Education-num is chosen here instead of education_level. I guess the reason is education-num is an ordinal parameter while educatoin_level was performed against one hot encoding. So definitely education-num is more informative than a bunch of one hot encoders derived from education_level.

I thought occupation was important but it's not. Instead, hours-per-week is in the list of top 5. Maybe because there are many professions pay by per hour. It's a little different from Singapore here that many profession don't pay for over time.

Feature Selection

How does a model perform if we only use a subset of all the available features in the data? With less features required 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 we can attempt to *reduce the feature space* and simplify the information required for the model to learn. The code cell below will use the same optimized model you found earlier, and train it on the same training set *with only the top five important features*.

```
In [14]:  1  # Import functionality for cloning a model
          2  from sklearn.base import clone
          3
          4  #print importances
          5  #print np.argsort(importances)
          6
          7  # Reduce the feature space
          8  X_train_reduced = X_train[X_train.columns.values[(np.argsort(importances)[::-1])[:10]]]
          9  X_test_reduced = X_test[X_test.columns.values[(np.argsort(importances)[::-1])[:10]]]
         10
         11  # Train on the "best" model found from grid search earlier
         12  clf = (clone(best_clf)).fit(X_train_reduced, y_train)
         13
         14  # Make new predictions
         15  reduced_predictions = clf.predict(X_test_reduced)
         16
         17  # Report scores from the final model using both versions of data
         18  print "Final Model trained on full data\n------"
         19  print "Accuracy on testing data: {:.4f}".format(accuracy_score(y_test, best_predictions))
         20  print "F-score on testing data: {:.4f}".format(fbeta_score(y_test, best_predictions, beta = 0.5))
         21  print "\nFinal Model trained on reduced data\n------"
         22  print "Accuracy on testing data: {:.4f}".format(accuracy_score(y_test, reduced_predictions))
         23  print "F-score on testing data: {:.4f}".format(fbeta_score(y_test, reduced_predictions, beta = 0.5))
```

```
Final Model trained on full data
------
Accuracy on testing data: 0.8419
F-score on testing data: 0.6832

Final Model trained on reduced data
------
Accuracy on testing data: 0.8353
F-score on testing data: 0.6690
```

Question 8 - Effects of Feature Selection

- How does the final model's F-score and accuracy score on the reduced data using only five features compare to those same scores when all features are used?
- If training time was a factor, would you consider using the reduced data as your training set?

Answer: The accuracy scores are quite close while the F scores are a little far in between. However, if I set the number of features to 10, the scores are very close. I would definitely choose reduced features as it will save a lot of time on both training and predication. In this project, we used one hot encoding, there are more than 100 features, by choosing only 1/10 of features we almost achieved the same scores. That's a huge improvement.

> **Note**: Once you have completed all of the code implementations and successfully answered each question above, you may finalize your work by exporting the iPython Notebook as an HTML document. You can do this by using the menu above and navigating to **File -> Download as -> HTML (.html)**. Include the finished document along with this notebook as your submission.