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. 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. 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.

In [1]:

```python
# Import libraries necessary for this project
import numpy as np
import pandas as pd
from time import time
from IPython.display import display # Allows the use of display() for DataFrames

# Import supplementary visualization code visuals.py
import visuals as vs

# Pretty display for notebooks
%matplotlib inline

# Load the Census dataset
data = pd.read_csv("census.csv")
```

```
# Success - Display the first record
display(data.head(n=1))
```

	age	workclass	education_level	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours-per-week	na cou
0	39	State-gov	Bachelors	13.0	Never-married	Adm-clerical	Not-in-family	White	Male	2174.0	0.0	40.0	Unit Stai

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.

In [2]:

```
# TODO: Total number of records
n_records = data.shape[0]
#print(n_records)
# TODO:
#checking for null values
print("Number of missing values in income columnn:
{}".format(data['income'].isnull().sum().sum()))
#checking unique income
print("Number of unique income: {}".format(data['income'].unique()))
#Number of records where individual's income is more than $50,000
n_greater_50k = (data['income'] == '>50K').sum()
#Space
print()

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

# TODO: Percentage of individuals whose income is more than $50,000
#percentage to two decimal point
greater_percent = round((n_greater_50k/(n_greater_50k+n_at_most_50k))*100,2)

# 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: {}%".format(greater_percent))
```

```
Number of missing values in income columnn: 0
Number of unique income: ['<=50K' '>50K']

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%
```

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 [3]:

```
# Split the data into features and target label
income_raw = data['income']
features_raw = data.drop('income', axis = 1)
# Visualize skewed continuous features of original data
print(type(features_raw))
vs.distribution(data)
```

```
<class 'pandas.core.frame.DataFrame'>
```
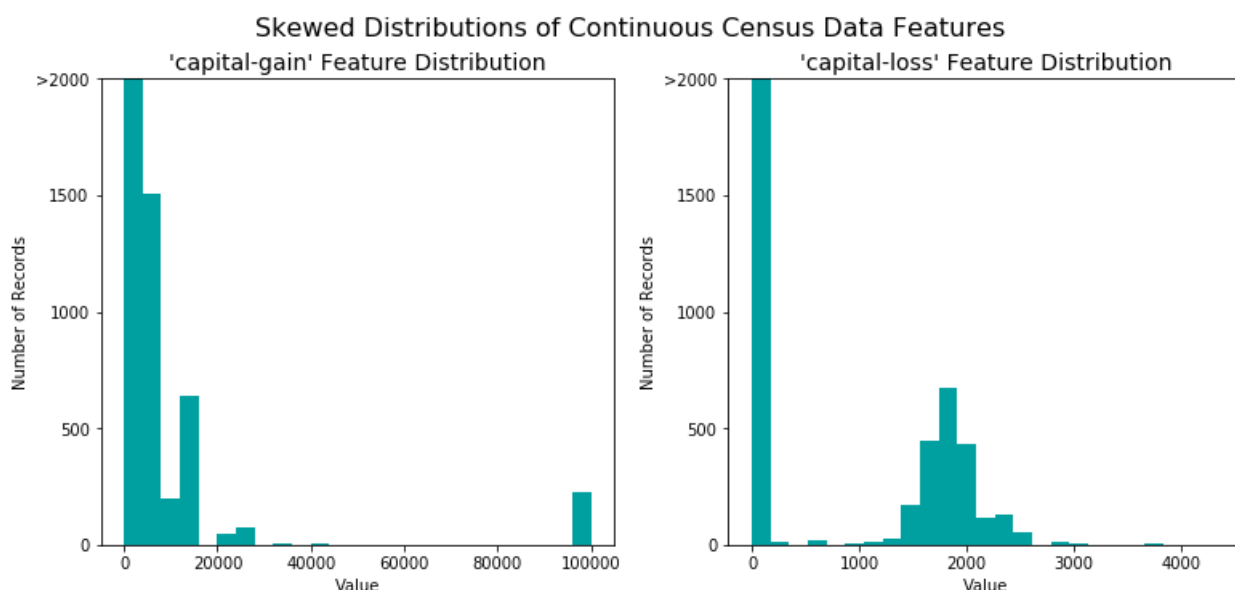

For highly-skewed feature distributions such as `'capital-gain'` and `'capital-loss'`, it is common practice to apply a logarithmic transformation on the data so that the very large and very small values do not negatively affect the performance of a learning algorithm. Using a logarithmic transformation 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.

In [4]:

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

features_log_transformed[skewed] = features_raw[skewed].apply(lambda x: np.log(x + 1))
# Visualize the new log distributions
vs.distribution(features_log_transformed, transformed = True)
```


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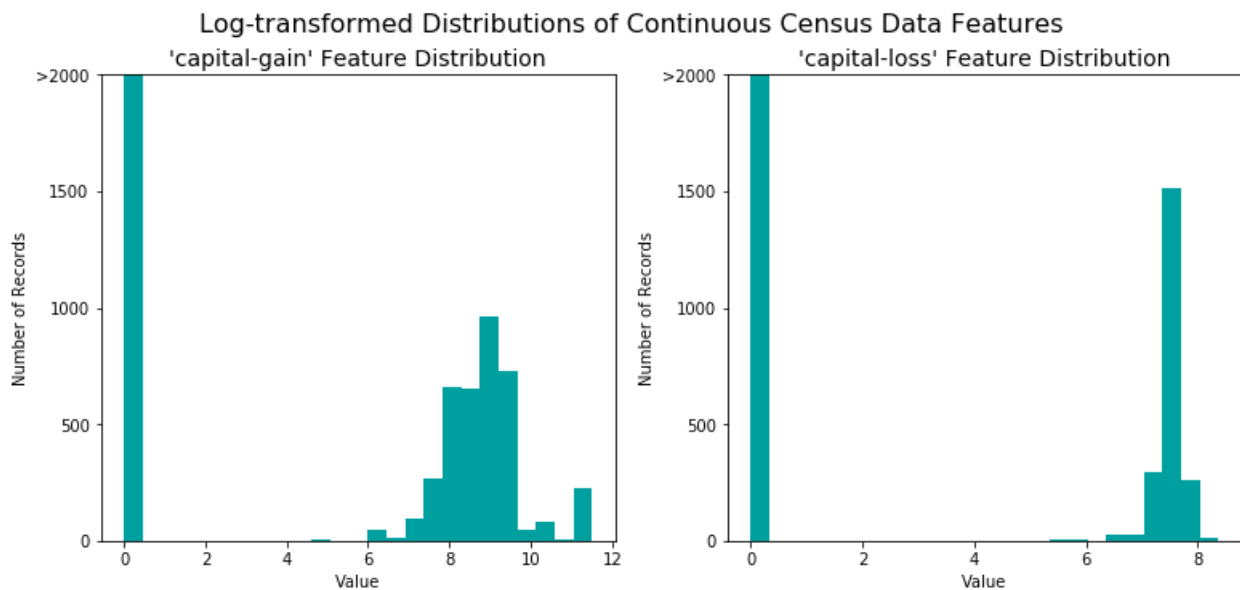
Normalizing Numerical Features

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 for this.

In [5]:

```python
# Import sklearn.preprocessing.StandardScaler
from sklearn.preprocessing import MinMaxScaler

# Initialize a scaler, then apply it to the features
scaler = MinMaxScaler() # default=(0, 1)
numerical = ['age', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week']

features_log_minmax_transform = pd.DataFrame(data = features_log_transformed)
features_log_minmax_transform[numerical] = scaler.fit_transform(features_log_transformed[numerical])

# Show an example of a record with scaling applied
display(features_log_minmax_transform.head(n = 5))
```

	age	workclass	education_level	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss
0	0.301370	State-gov	Bachelors	0.800000	Never-married	Adm-clerical	Not-in-family	White	Male	0.667492	0.0	0.

	age	workclass	education_level	education-num	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	
1	0.452055	Self-emp-not-inc	Bachelors	0.800000	Married-civ-spouse	Exec-managerial	Husband	White	Male	0.000000	0.0	0.
2	0.287671	Private	HS-grad	0.533333	Divorced	Handlers-cleaners	Not-in-family	White	Male	0.000000	0.0	0.
3	0.493151	Private	11th	0.400000	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0.000000	0.0	0.
4	0.150685	Private	Bachelors	0.800000	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0.000000	0.0	0.

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 ---->	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()](#) 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`.

In [6]:

```python
# TODO: One-hot encode the 'features_log_minmax_transform' data using pandas.get_dummies()
print()
features_final = pd.get_dummies(features_log_minmax_transform)
#print(features_final)

# TODO: Encode the 'income_raw' data to numerical values
income = income_raw.replace(['<=50K','>50K'], [0,1])

# Print the number of features after one-hot encoding
encoded = list(features_final.columns)
print("{} total features after one-hot encoding.".format(len(encoded)))

# Uncomment the following line to see the encoded feature names
print(encoded)
#print(features_final)
```

```
103 total features after one-hot encoding.
['age', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week', 'workclass_ Federal-gov
', 'workclass_ Local-gov', 'workclass_ Private', 'workclass_ Self-emp-inc', 'workclass_ Self-emp-n
ot-inc', 'workclass_ State-gov', 'workclass_ Without-pay', 'education_level_ 10th',
'education_level_ 11th', 'education_level_ 12th', 'education_level_ 1st-4th', 'education_level_ 5t
h-6th', 'education_level_ 7th-8th', 'education_level_ 9th', 'education_level_ Assoc-acdm',
'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_ Some-college', 'marital-status_ Divorced',
'marital-status_ Married-AF-spouse', 'marital-status_ Married-civ-spouse', 'marital-status_ Marrie
d-spouse-absent', 'marital-status_ Never-married', 'marital-status_ Separated', 'marital-status_ W
idowed', 'occupation_ Adm-clerical', 'occupation_ Armed-Forces', 'occupation_ Craft-repair',
'occupation_ Exec-managerial', 'occupation_ Farming-fishing', 'occupation_ Handlers-cleaners', 'oc
```

Occupation_ Exec-managerial', 'Occupation_ Farming-fishing', 'Occupation_ Handlers-cleaners', 'Oc
cupation_ Machine-op-inspct', 'occupation_ Other-service', 'occupation_ Priv-house-serv',
'occupation_ Prof-specialty', 'occupation_ Protective-serv', 'occupation_ Sales', 'occupation_
Tech-support', 'occupation_ Transport-moving', 'relationship_ Husband', 'relationship_ Not-in-fami
ly', 'relationship_ Other-relative', 'relationship_ Own-child', 'relationship_ Unmarried', 'relati
onship_ Wife', 'race_ Amer-Indian-Eskimo', 'race_ Asian-Pac-Islander', 'race_ Black', 'race_ Other
', 'race_ White', 'sex_ Female', 'sex_ Male', 'native-country_ Cambodia', 'native-country_
Canada', 'native-country_ China', 'native-country_ Columbia', 'native-country_ Cuba', 'native-coun
try_ Dominican-Republic', 'native-country_ Ecuador', 'native-country_ El-Salvador', 'native-countr
y_ England', 'native-country_ France', 'native-country_ Germany', 'native-country_ Greece', 'nativ
e-country_ Guatemala', 'native-country_ Haiti', 'native-country_ Holand-Netherlands', 'native-coun
try_ Honduras', 'native-country_ Hong', 'native-country_ Hungary', 'native-country_ India', 'nativ
e-country_ Iran', 'native-country_ Ireland', 'native-country_ Italy', 'native-country_ Jamaica', '
native-country_ Japan', 'native-country_ Laos', 'native-country_ Mexico', 'native-country_ Nicarag
ua', 'native-country_ Outlying-US(Guam-USVI-etc)', 'native-country_ Peru', 'native-country_ Philip
pines', 'native-country_ Poland', 'native-country_ Portugal', 'native-country_ Puerto-Rico', 'nati
ve-country_ Scotland', 'native-country_ South', 'native-country_ Taiwan', 'native-country_
Thailand', 'native-country_ Trinadad&Tobago', 'native-country_ United-States', 'native-country_ Vi
etnam', 'native-country_ Yugoslavia']

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 [7]:

```python
# Import train_test_split
from sklearn.cross_validation import train_test_split

# Split the 'features' and 'income' data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features_final,
                                                    income,
                                                    test_size = 0.2,
                                                    random_state = 0)

# Show the results of the split
print("Training set has {} samples.".format(X_train.shape[0]))
print("Testing set has {} samples.".format(X_test.shape[0]))
```

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

```
C:\Users\Shreyas\Anaconda3\lib\site-packages\sklearn\cross_validation.py:41: DeprecationWarning: T
his module was deprecated in version 0.18 in favor of the model_selection module into which all th
e refactored classes and functions are moved. Also note that the interface of the new CV iterators
are different from that of this module. This module will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
```

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 **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.

Please note that the the purpose of generating a naive predictor is simply to show what a base model without any intelligence would look like. In the real world, ideally your base model would be either the results of a previous model or could be based on a research paper upon which you are looking to improve. When there is no benchmark model set, getting a result better than random choice is a place you could start from.

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 [8]:

```
'''
TP = np.sum(income) # Counting the ones as this is the naive case. Note that 'income' is the 'income_raw' data
encoded to numerical values done in the data preprocessing step.
FP = income.count() - TP # Specific to the naive case

TN = 0 # No predicted negatives in the naive case
FN = 0 # No predicted negatives in the naive case
'''
#print(income.count() - np.sum(income))
# TODO: Calculate accuracy, precision and recall
accuracy =   (np.sum(income) + 0)/income.count()
```

```
recall = np.sum(income)/(np.sum(income) + 0)
precision = np.sum(income)/(np.sum(income) + income.count() - np.sum(income) )

# TODO: Calculate F-score using the formula above for beta = 0.5 and correct values for precision
and recall.
fscore = ((1+ .5*.5)*precision*recall)/(.5*.5*precision + recall)

# Print the results
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` **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:
1. LOGISTIC REGRESSION
resources: https://machinelearning-blog.com/2018/04/23/logistic-regression-101/ https://towardsdatascience.com/real-world-implementation-of-logistic-regression-5136cefb8125 https://www.analyticsinsight.net/introduction-to-logistic-regression/ https://towardsdatascience.com/real-world-implementation-of-logistic-regression-5136cefb8125

One Real World Application -
Banks use CIBIL score, Income and Collateral to predict if the customer will default the loan or not. The outcocme here is whether the loan will be defaulted or not, hence having only two possible outcomes. Resource: Discussion with father.

Strength of the model

1. the model works best when the data can be seprated linearly into two using a line, plane or polynomial lines.
2. The ouput is expected to have two states yes/no, true/false, 1/0,
3. The independent variables are removed which are not related to the output.

Weakness of the model

1. It does not work on data which can not be linearly seprable
2. Does not work well when independent variables are interconnected
3. does not work well when output has no relation to the indepednednt variable.

Why it is a good candidate for this model ? Since the predicion needs to be binary that is the person makes more than 50K or not.

2. DESICION TREE

2. DESICION TREE

Resorces : https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4466856/ https://medium.com/@sagar.rawale3/understanding-decision-tree-algorithm-drawbacks-and-advantages-4486efa6b8c3 **One Real World Application** -

Popular websites such as YouTube, Facebook, Amazon use desicion tree to recommand advertisements by considering data such as age, location, hours spent online and more. Phone apps can also be suggested based on user details(age,sex, location) which the user agrees to share for suggestions.

Strength of the model

1. Desicion tree can be read by humans.
2. It can handle missing data by classifying it as another category.
3. the desicion is either yes or no, data can be read when the rules are formed.

Weakness of the model

1. High chances of over fitting training data.
2. calcualtion becomes complex when there are many features.
3. it is important to identifing relevant features.

Why it is a good candidate for this model? There are many columns such as age, workclass, marital-status, capital-gain, capital-los, etc which can be used to form desicion tree to form predictions. Desicion tree on these can be formed to predict the model.

3. ADABOOST CLASSIFICAION

Resources: https://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning/
http://www.cs.man.ac.uk/~nikolaon/~nikolaon_files/Introduction_to_AdaBoost.pdf
http://users.cecs.anu.edu.au/~wanglei/SPR_course/boosting.pdf

Real world Application
Face detection Applications

Strengths of the model

1. It continously adapts to the previous mistakes and works on new models.
2. weighted majority vote hence there is a confident prediction
3. can be added to other supervised classifier for training usch as discicion tree, random forest.

Weakness of the model

1. Data with noice(in correct information) leads to wrong results
2. Weights are attachedd to the data, outliers can lead to wrong weights assigned to the data and the program may try hard to find the correct weight due to the outlier.
3. the Training data needs to be perfectly clean

Why is this a good candidate for this model?
Since the output is binary and there are no missing data this can be a good candidate. The independent data are interconnected.

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.
- 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!

In [9]:

```
# TODO: Import two metrics from sklearn - fbeta_score and accuracy_score
from sklearn.metrics import fbeta_score, accuracy_score
```

```python
from sklearn.metrics import fbeta_score,accuracy_score

def train_predict(learner, sample_size, X_train, y_train, X_test, y_test):
    '''
    inputs:
       - learner: the learning algorithm to be trained and predicted on
       - sample_size: the size of samples (number) to be drawn from training set
       - X_train: features training set
       - y_train: income training set
       - X_test: features testing set
       - y_test: income testing set
    '''

    results = {}

    # TODO: Fit the learner to the training data using slicing with 'sample_size' using
    # .fit(training_features[:], training_labels[:])
    start = time() # Get start time
    learner = learner.fit(X_train[:sample_size],y_train[:sample_size])
    end = time() # Get end time

    # TODO: Calculate the training time
    results['train_time'] = end - start

    # TODO: Get the predictions on the test set(X_test),
    #       then get predictions on the first 300 training samples(X_train) using .predict()
    start = time() # Get start time
    predictions_test = learner.predict(X_test)
    predictions_train = learner.predict(X_train[:300])
    end = time() # Get end time

    # TODO: Calculate the total prediction time
    results['pred_time'] = end - start

    # TODO: Compute accuracy on the first 300 training samples which is y_train[:300]
    results['acc_train'] = accuracy_score(y_train[:300], predictions_train)


    # TODO: Compute accuracy on test set using accuracy_score()
    results['acc_test'] = accuracy_score(y_test, predictions_test)
    print('The acc_train:',results['acc_train'],'  The acc_test:',results['acc_test'])

    # TODO: Compute F-score on the the first 300 training samples using fbeta_score()
    results['f_train'] = fbeta_score(y_train[:300], predictions_train, beta=0.5)

    # TODO: Compute F-score on the test set which is y_test
    results['f_test'] = fbeta_score(y_test, predictions_test, beta=0.5)
    print('The f_train:',results['f_train'],'  The f_test:',results['f_test'])
    # Success
    print("{} trained on {} samples.".format(learner.__class__.__name__, sample_size))
    print()
    # Return the results
    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 [10]:

```python
# TODO: Import the three supervised learning models from sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC
```

```python
# TODO: Initialize the three models
clf_A = LogisticRegression(random_state = 0)
clf_B = DecisionTreeClassifier(random_state =0)
clf_C = AdaBoostClassifier(random_state = 0)

# TODO: Calculate the number of samples for 1%, 10%, and 100% of the training data
# HINT: samples_100 is the entire training set i.e. len(y_train)
# HINT: samples_10 is 10% of samples_100 (ensure to set the count of the values to be `int` and not `float`)
# HINT: samples_1 is 1% of samples_100 (ensure to set the count of the values to be `int` and not `float`)
samples_100 = len(y_train)
samples_10 = int(samples_100*.1)
samples_1 = int(samples_100*0.01)

# Collect results on the learners
results = {}
for clf in [clf_A, clf_B, clf_C]:
    clf_name = clf.__class__.__name__
    results[clf_name] = {}
    for i, samples in enumerate([samples_1, samples_10, samples_100]):
        results[clf_name][i] = \
        train_predict(clf, samples, X_train, y_train, X_test, y_test)

# Run metrics visualization for the three supervised learning models chosen
vs.evaluate(results, accuracy, fscore)
```

```
The acc_train: 0.86    The acc_test: 0.8186843559977889
The f_train: 0.7386363636363635    The f_test: 0.6288601909039865
LogisticRegression trained on 361 samples.

The acc_train: 0.8466666666666667    The acc_test: 0.8385848535102266
The f_train: 0.703125    The f_test: 0.6775067750677506
LogisticRegression trained on 3617 samples.

The acc_train: 0.8466666666666667    The acc_test: 0.841901603095633
The f_train: 0.6985294117647058    The f_test: 0.6831652282416554
LogisticRegression trained on 36177 samples.

The acc_train: 1.0    The acc_test: 0.77191818684356
The f_train: 1.0    The f_test: 0.5359784216479596
DecisionTreeClassifier trained on 361 samples.

The acc_train: 0.9966666666666667    The acc_test: 0.8016583747927032
The f_train: 0.997191011235955    The f_test: 0.5938748335552595
DecisionTreeClassifier trained on 3617 samples.

The acc_train: 0.97    The acc_test: 0.8185737976782753
The f_train: 0.9638554216867471    The f_test: 0.627939142461964
DecisionTreeClassifier trained on 36177 samples.

The acc_train: 0.8933333333333333    The acc_test: 0.8206744057490326
The f_train: 0.8012820512820513    The f_test: 0.6327567673214111
AdaBoostClassifier trained on 361 samples.

The acc_train: 0.84    The acc_test: 0.849861802100608
The f_train: 0.6801470588235293    The f_test: 0.7018820838099199
AdaBoostClassifier trained on 3617 samples.

The acc_train: 0.85    The acc_test: 0.8576008844665561
The f_train: 0.7115384615384616    The f_test: 0.7245508982035928
AdaBoostClassifier trained on 36177 samples.
```
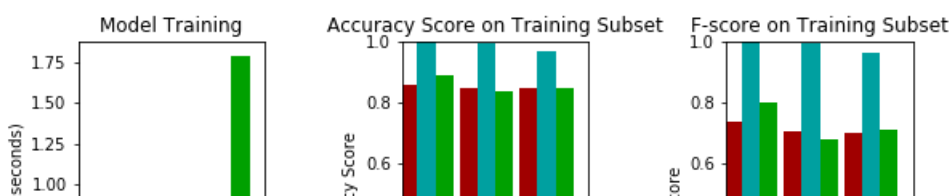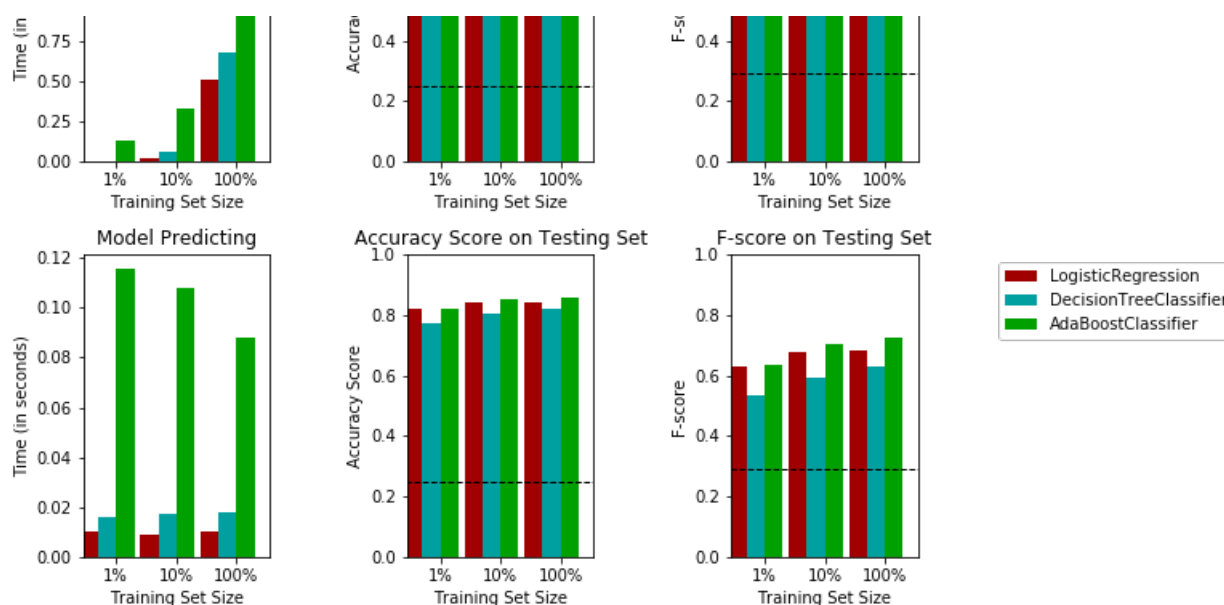

Performance Metrics for Three Supervised Learning Models

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:
F score
F score of AdaBoost is the above .7 for both training and test data for 100% taining data.
Accuracy, F-score of traing and test data for desicion tree shows that the model is overfitting.

Prediction/Training time
the Taining and prediticting time is more than 3 to 4 times more times more than the other models.
The algorithm's suitability for the data
Since the output is dependent on many columns, simple classifiers such as logistic, linear and other models can not classify the data. Desicion tree will only overfit the data during training the model.

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 data points in 2D space/graph, belonging to two categories can be tried to split into two by using a line. All the data points on one

side of the line can be separated from the other side of the line. This line forms the basis of desicion, hence a desicion tree can be formed using the line.

The line may not be able to separate all the data points in two categories correctly hence in Adaboost it is called a weak learner/classifier. The line tries to best separate the points to get minimum errors. The error for the first learner/line is noted and size/weight of the errors is increased by a factor so that the sum of sizes/weight of errors and correct values are 50:50.

The process is repeated for the next weak learner. The line for the second weak learner is formed so that there is minimum number of errors based on the sum of the sizes/weights. This forms a sequential process where the sizes/weights are adjusted for the next weak learner and there are multiple weak learners.

All the weak learners results are combined to form a complex line separating the two categories which best separates the categories in 2D space.

References:
https://towardsdatascience.com/boosting-and-adaboost-clearly-explained-856e21152d3e

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` and `sklearn.metrics.make_scorer`.
- 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 [11]:

```python
# TODO: Import 'GridSearchCV', 'make_scorer', and any other necessary libraries
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer,fbeta_score
# TODO: Initialize the classifier
clf = AdaBoostClassifier(random_state = 0)

# TODO: Create the parameters list you wish to tune, using a dictionary if needed.
# HINT: parameters = {'parameter_1': [value1, value2], 'parameter_2': [value1, value2]}
parameters = { 'n_estimators': [10,50,100,1000,1],'learning_rate': [1,3,5]}

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

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

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

# Get the estimator
best_clf = grid_fit.best_estimator_

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

# Report the before-and-afterscores
print("Unoptimized model\n------")
print("Accuracy score on testing data: {:.4f}".format(accuracy_score(y_test, predictions)))
print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, predictions, beta = 0.5)))
print("\nOptimized Model\n------")
print("Final accuracy score on the testing data: {:.4f}".format(accuracy_score(y_test, best_predict
ions)))
print("Final F-score on the testing data: {:.4f}".format(fbeta_score(y_test, best_predictions, beta
```

```
= 0.5)))
```

```
C:\Users\Shreyas\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\Shreyas\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\Shreyas\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\Shreyas\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\Shreyas\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\Shreyas\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\Shreyas\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\Shreyas\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
C:\Users\Shreyas\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1135:
UndefinedMetricWarning: F-score is ill-defined and being set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)
```

```
Unoptimized model
------
Accuracy score on testing data: 0.8576
F-score on testing data: 0.7246

Optimized Model
------
Final accuracy score on the testing data: 0.8676
Final F-score on the testing data: 0.7448
```

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	Unoptimized Model	Optimized Model
Accuracy Score	.8576	.8676
F-score	.7246	.7448

Answer:
1. The accuracy of the optimized model is .8676 and the F-score(Beta = 0.5) is .7448.
2. The scores are better than the unoptimized model. By changing the learning rate and estimators the scores improved.
3. when the model using Naive predictor always predicted '1'/ individual making more than 50K the accuracy was 0.2478 and F-score: 0.2917 which is very bad model. The optimized model is very good compared to the Naive Predictor.

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

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:

1. Capital - Loss: the data in excel shows that, for income more thn 50k the capital loss is greater than 1500. This is valid for more than 99% of the data with capital-loss not equal to zero.
2. Capital - gain: while viewing the data in excel with filter, most of the incomes are above $50k for capital gains > 5000 and less than $50k for capital gains <5000.
3. Age : For people whose age is below 19 might not be earning and for earderly above 60 might have retired, So I am guessing age is the most importatnt feature for determining income.
4. Education_num: the number of years they have spent on education. This may tell us if they have done schooling, graduation, post graduation. The more degrees they have the higher chances of more income.
5. Hours per week: The more number of hours spent per week meaning more money. If the hour spent is below 10 Hrs which means it is a student working on weekends(assuming)

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 [12]:

```
# TODO: Import a supervised learning model that has 'feature_importances_'
#Adaboost already imported in previous section

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

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

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


Normalized Weights for First Five Most Predictive Features
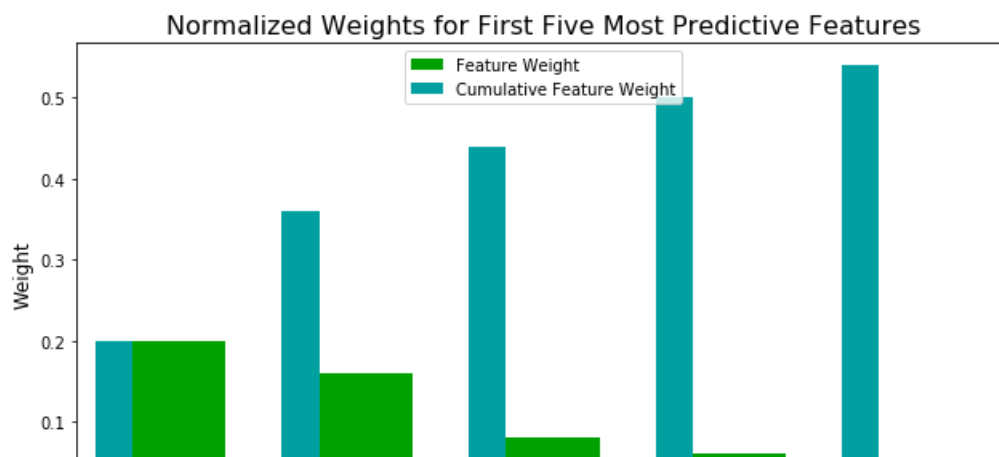
Question 7 - Extracting Feature Importance

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:
The visualization helps to understand different weights attached to different models in Adaboost. the cumulative feature weight shows the sum of different weights attached to features. Feature weight being the average of the weight attached to the feature. The visualization helps to understand that diferent weights needs to be attached to features in different models.

Yes the visualization confirms the thoughts because, marital status, workclass, occupation, relation and native-country are not very relevant to make the conclusion for income.

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 [13]:

```python
# Import functionality for cloning a model
from sklearn.base import clone

# Reduce the feature space
X_train_reduced = X_train[X_train.columns.values[(np.argsort(importances)[::-1])[:5]]]
X_test_reduced = X_test[X_test.columns.values[(np.argsort(importances)[::-1])[:5]]]

# Train on the "best" model found from grid search earlier
clf = (clone(best_clf)).fit(X_train_reduced, y_train)

# Make new predictions
reduced_predictions = clf.predict(X_test_reduced)

# Report scores from the final model using both versions of data
print("Final Model trained on full data\n------")
print("Accuracy on testing data: {:.4f}".format(accuracy_score(y_test, best_predictions)))
print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, best_predictions, beta = 0.5)))
print("\nFinal Model trained on reduced data\n------")
print("Accuracy on testing data: {:.4f}".format(accuracy_score(y_test, reduced_predictions)))
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: 0.8676
F-score on testing data: 0.7448

Final Model trained on reduced data
------
Accuracy on testing data: 0.8433
F-score on testing data: 0.7053
```

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?

- If training time was a factor, would you consider using the reduced data as your training set?

Answer:
The F-Score reduces considerably more than accuracy wehn only five features are used.
If Training time was a factor, the reduced data would not have been considered. Other features also play role in predicting the income. As the f- score reduces it can be stated that all the data is important for the prediction.

> **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.