finding_donors

June 15, 2021

0.1 Supervised Learning

0.2 Project: Finding Donors for *CharityML*

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.

0.3 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.

0.4 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 [27]: # 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  \
0   39   State-gov       Bachelors           13.0   Never-married

      occupation    relationship    race    sex  capital-gain  capital-loss  \
0   Adm-clerical   Not-in-family   White   Male        2174.0           0.0

   hours-per-week  native-country income
0            40.0   United-States  <=50K
```

0.4.1 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 [28]: data
```

Out[28]:

	age	workclass	education_level	education-num \
0	39	State-gov	Bachelors	13.0
1	50	Self-emp-not-inc	Bachelors	13.0
2	38	Private	HS-grad	9.0
3	53	Private	11th	7.0
4	28	Private	Bachelors	13.0
5	37	Private	Masters	14.0
6	49	Private	9th	5.0
7	52	Self-emp-not-inc	HS-grad	9.0
8	31	Private	Masters	14.0
9	42	Private	Bachelors	13.0
10	37	Private	Some-college	10.0
11	30	State-gov	Bachelors	13.0
12	23	Private	Bachelors	13.0
13	32	Private	Assoc-acdm	12.0
14	34	Private	7th-8th	4.0
15	25	Self-emp-not-inc	HS-grad	9.0
16	32	Private	HS-grad	9.0
17	38	Private	11th	7.0
18	43	Self-emp-not-inc	Masters	14.0
19	40	Private	Doctorate	16.0
20	54	Private	HS-grad	9.0
21	35	Federal-gov	9th	5.0
22	43	Private	11th	7.0
23	59	Private	HS-grad	9.0
24	56	Local-gov	Bachelors	13.0
25	19	Private	HS-grad	9.0
26	39	Private	HS-grad	9.0
27	49	Private	HS-grad	9.0
28	23	Local-gov	Assoc-acdm	12.0
29	20	Private	Some-college	10.0
...	...	...	...	...
45192	25	Private	HS-grad	9.0
45193	31	Private	HS-grad	9.0
45194	49	Self-emp-inc	HS-grad	9.0
45195	60	Private	Assoc-voc	11.0
45196	39	Private	Bachelors	13.0
45197	38	Private	Masters	14.0
45198	43	Local-gov	Masters	14.0
45199	23	Private	HS-grad	9.0
45200	73	Self-emp-inc	Some-college	10.0
45201	35	Private	Some-college	10.0
45202	66	Private	HS-grad	9.0
45203	27	Private	Some-college	10.0
45204	40	Private	Prof-school	15.0
45205	51	Private	HS-grad	9.0
45206	22	Private	Some-college	10.0
45207	64	Self-emp-not-inc	HS-grad	9.0

```
45208   55          Private      HS-grad          9.0
45209   38          Private    Assoc-voc         11.0
45210   58          Private   Assoc-acdm         12.0
45211   32          Private      HS-grad          9.0
45212   48          Private      HS-grad          9.0
45213   61          Private      HS-grad          9.0
45214   31          Private      HS-grad          9.0
45215   25          Private      HS-grad          9.0
45216   48        Local-gov      Masters         14.0
45217   33          Private    Bachelors         13.0
45218   39          Private    Bachelors         13.0
45219   38          Private    Bachelors         13.0
45220   44          Private    Bachelors         13.0
45221   35     Self-emp-inc    Bachelors         13.0

               marital-status          occupation    relationship  \
0               Never-married        Adm-clerical   Not-in-family
1          Married-civ-spouse     Exec-managerial         Husband
2                    Divorced   Handlers-cleaners   Not-in-family
3          Married-civ-spouse   Handlers-cleaners         Husband
4          Married-civ-spouse      Prof-specialty            Wife
5          Married-civ-spouse     Exec-managerial            Wife
6       Married-spouse-absent       Other-service   Not-in-family
7          Married-civ-spouse     Exec-managerial         Husband
8               Never-married      Prof-specialty   Not-in-family
9          Married-civ-spouse     Exec-managerial         Husband
10         Married-civ-spouse     Exec-managerial         Husband
11         Married-civ-spouse      Prof-specialty         Husband
12              Never-married        Adm-clerical       Own-child
13              Never-married               Sales   Not-in-family
14         Married-civ-spouse    Transport-moving         Husband
15              Never-married     Farming-fishing       Own-child
16              Never-married   Machine-op-inspct       Unmarried
17         Married-civ-spouse               Sales         Husband
18                   Divorced     Exec-managerial       Unmarried
19         Married-civ-spouse      Prof-specialty         Husband
20                  Separated       Other-service       Unmarried
21         Married-civ-spouse     Farming-fishing         Husband
22         Married-civ-spouse    Transport-moving         Husband
23                   Divorced        Tech-support       Unmarried
24         Married-civ-spouse        Tech-support         Husband
25              Never-married        Craft-repair       Own-child
26                   Divorced     Exec-managerial   Not-in-family
27         Married-civ-spouse        Craft-repair         Husband
28              Never-married     Protective-serv   Not-in-family
29              Never-married               Sales       Own-child
...                       ...                 ...             ...
45192                Divorced   Machine-op-inspct   Not-in-family
```

```
45193        Never-married  Machine-op-inspct   Not-in-family
45194   Married-civ-spouse    Exec-managerial         Husband
45195   Married-civ-spouse     Prof-specialty         Husband
45196        Never-married       Tech-support   Not-in-family
45197   Married-civ-spouse     Prof-specialty         Husband
45198   Married-civ-spouse    Exec-managerial         Husband
45199        Never-married  Machine-op-inspct       Own-child
45200             Divorced    Exec-managerial   Not-in-family
45201   Married-civ-spouse    Protective-serv         Husband
45202              Widowed              Sales  Other-relative
45203        Never-married              Sales   Not-in-family
45204   Married-civ-spouse     Prof-specialty         Husband
45205   Married-civ-spouse       Craft-repair         Husband
45206        Never-married       Craft-repair       Own-child
45207              Widowed    Farming-fishing   Not-in-family
45208            Separated    Priv-house-serv   Not-in-family
45209        Never-married       Adm-clerical       Unmarried
45210             Divorced     Prof-specialty   Not-in-family
45211   Married-civ-spouse  Handlers-cleaners         Husband
45212   Married-civ-spouse       Adm-clerical         Husband
45213   Married-civ-spouse              Sales         Husband
45214   Married-civ-spouse       Craft-repair         Husband
45215        Never-married      Other-service       Own-child
45216             Divorced      Other-service   Not-in-family
45217        Never-married     Prof-specialty       Own-child
45218             Divorced     Prof-specialty   Not-in-family
45219   Married-civ-spouse     Prof-specialty         Husband
45220             Divorced       Adm-clerical       Own-child
45221   Married-civ-spouse    Exec-managerial         Husband

                     race     sex  capital-gain  capital-loss  \
0                   White    Male        2174.0           0.0
1                   White    Male           0.0           0.0
2                   White    Male           0.0           0.0
3                   Black    Male           0.0           0.0
4                   Black  Female           0.0           0.0
5                   White  Female           0.0           0.0
6                   Black  Female           0.0           0.0
7                   White    Male           0.0           0.0
8                   White  Female       14084.0           0.0
9                   White    Male        5178.0           0.0
10                  Black    Male           0.0           0.0
11     Asian-Pac-Islander    Male           0.0           0.0
12                  White  Female           0.0           0.0
13                  Black    Male           0.0           0.0
14     Amer-Indian-Eskimo    Male           0.0           0.0
15                  White    Male           0.0           0.0
16                  White    Male           0.0           0.0
```

```
17                     White     Male           0.0           0.0
18                     White   Female           0.0           0.0
19                     White     Male           0.0           0.0
20                     Black   Female           0.0           0.0
21                     Black     Male           0.0           0.0
22                     White     Male           0.0        2042.0
23                     White   Female           0.0           0.0
24                     White     Male           0.0           0.0
25                     White     Male           0.0           0.0
26                     White     Male           0.0           0.0
27                     White     Male           0.0           0.0
28                     White     Male           0.0           0.0
29                     Black     Male           0.0           0.0
...                      ...      ...           ...           ...
45192                  Black     Male           0.0           0.0
45193                  White     Male           0.0           0.0
45194                  White     Male           0.0           0.0
45195                  White     Male        7688.0           0.0
45196                  White   Female           0.0        1669.0
45197                  White     Male           0.0           0.0
45198                  White     Male           0.0        1902.0
45199                  White     Male           0.0           0.0
45200                  White   Female           0.0           0.0
45201                  White     Male           0.0           0.0
45202                  White   Female           0.0           0.0
45203                  White   Female           0.0           0.0
45204                  White     Male       15024.0           0.0
45205                  White     Male           0.0           0.0
45206                  White     Male           0.0           0.0
45207                  White     Male           0.0           0.0
45208                  White   Female           0.0           0.0
45209                  Black   Female           0.0           0.0
45210                  White     Male           0.0           0.0
45211                  White     Male           0.0           0.0
45212                  White     Male           0.0           0.0
45213                  White     Male           0.0           0.0
45214                  White     Male           0.0           0.0
45215                  White   Female           0.0           0.0
45216                  White     Male           0.0           0.0
45217                  White     Male           0.0           0.0
45218                  White   Female           0.0           0.0
45219                  White     Male           0.0           0.0
45220     Asian-Pac-Islander     Male        5455.0           0.0
45221                  White     Male           0.0           0.0

       hours-per-week  native-country income
0                40.0   United-States  <=50K
1                13.0   United-States  <=50K
```

2	40.0	United-States	<=50K
3	40.0	United-States	<=50K
4	40.0	Cuba	<=50K
5	40.0	United-States	<=50K
6	16.0	Jamaica	<=50K
7	45.0	United-States	>50K
8	50.0	United-States	>50K
9	40.0	United-States	>50K
10	80.0	United-States	>50K
11	40.0	India	>50K
12	30.0	United-States	<=50K
13	50.0	United-States	<=50K
14	45.0	Mexico	<=50K
15	35.0	United-States	<=50K
16	40.0	United-States	<=50K
17	50.0	United-States	<=50K
18	45.0	United-States	>50K
19	60.0	United-States	>50K
20	20.0	United-States	<=50K
21	40.0	United-States	<=50K
22	40.0	United-States	<=50K
23	40.0	United-States	<=50K
24	40.0	United-States	>50K
25	40.0	United-States	<=50K
26	80.0	United-States	<=50K
27	40.0	United-States	<=50K
28	52.0	United-States	<=50K
29	44.0	United-States	<=50K
...	...	...	...
45192	40.0	United-States	<=50K
45193	40.0	United-States	<=50K
45194	40.0	Canada	>50K
45195	40.0	United-States	>50K
45196	40.0	United-States	<=50K
45197	50.0	United-States	>50K
45198	50.0	United-States	>50K
45199	40.0	United-States	<=50K
45200	40.0	United-States	<=50K
45201	40.0	United-States	<=50K
45202	8.0	United-States	<=50K
45203	45.0	United-States	<=50K
45204	55.0	United-States	>50K
45205	40.0	United-States	<=50K
45206	40.0	United-States	<=50K
45207	32.0	United-States	<=50K
45208	32.0	United-States	<=50K
45209	40.0	United-States	<=50K
45210	36.0	United-States	<=50K

```
45211            40.0  United-States  <=50K
45212            40.0  United-States  <=50K
45213            48.0  United-States  <=50K
45214            40.0  United-States  <=50K
45215            40.0  United-States  <=50K
45216            40.0  United-States  <=50K
45217            40.0  United-States  <=50K
45218            36.0  United-States  <=50K
45219            50.0  United-States  <=50K
45220            40.0  United-States  <=50K
45221            60.0  United-States   >50K

[45222 rows x 14 columns]
```

In [29]:
```
data.dtypes
```

Out[29]:
```
age                  int64
workclass           object
education_level     object
education-num      float64
marital-status      object
occupation          object
relationship        object
race                object
sex                 object
capital-gain       float64
capital-loss       float64
hours-per-week     float64
native-country      object
income              object
dtype: object
```

In [30]:
```
# TODO: Total number of records
n_records = data.shape[0]

# TODO: Number of records where individual's income is more than $50,000
more = data[data['income'] =='>50K']
n_greater_50k = more.shape[0]

# TODO: Number of records where individual's income is at most $50,000
less = data[data['income'] =='<=50K']
n_at_most_50k = less.shape[0]

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

# Print the results
print("Total number of records: {}".format(n_records))
```

```python
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)
```

```
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: 24.78439697492371%
```

** 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.

0.5 Preparing the Data

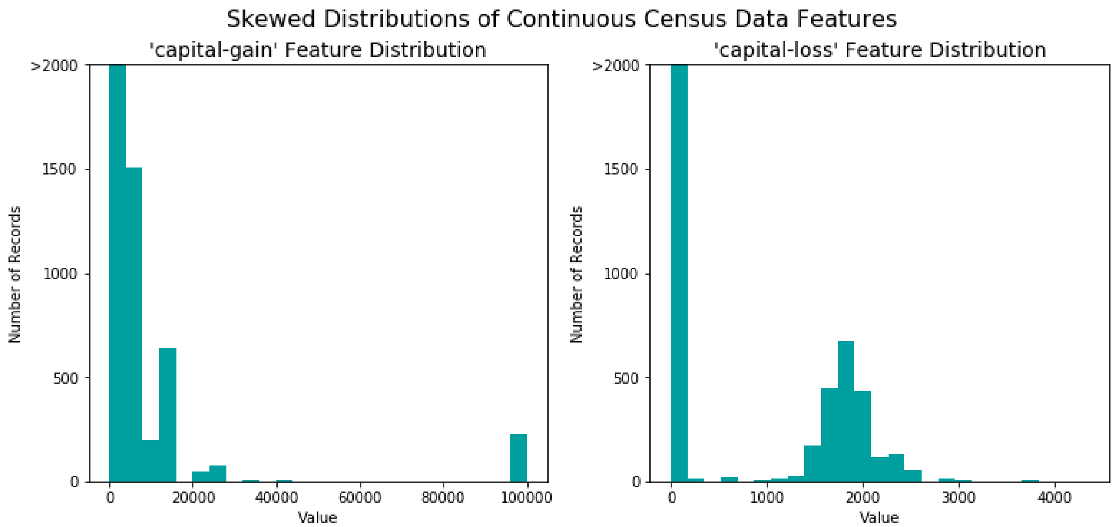
Before data can be used as input for machine learning algorithms, it often must be cleaned, formatted, and restructured — this is typically known as **preprocessing**. Fortunately, for this dataset, there are no invalid or missing entries we must deal with, however, there are some qualities about certain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

0.5.1 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.

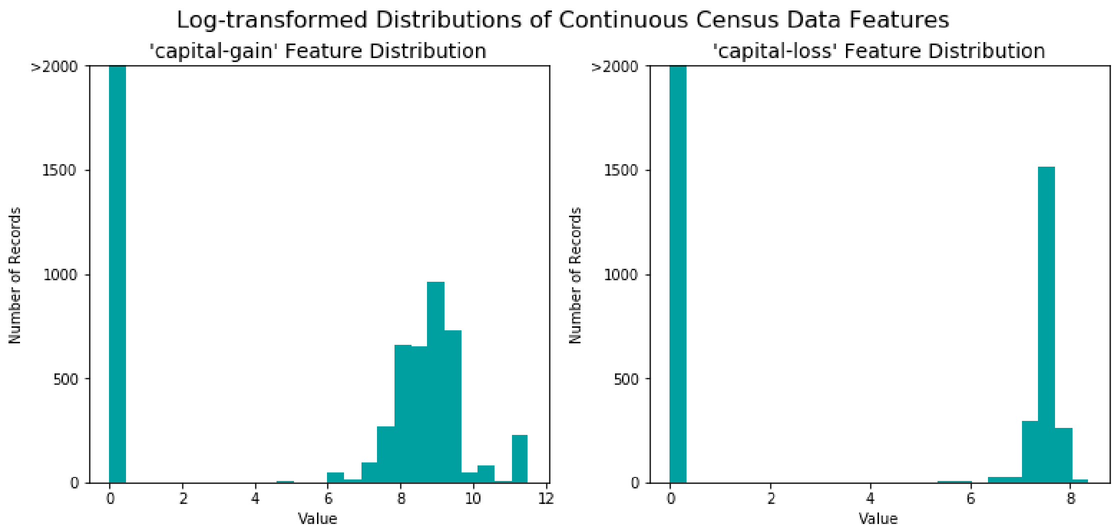
```
In [31]: # Split the data into features and target label
         income_raw = data['income']
         features_raw = data.drop('income', axis = 1)

         # Visualize skewed continuous features of original data
         vs.distribution(data)
```



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 [32]: # 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)
```



0.5.2 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 [33]:
```
# 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_transforme
# Show an example of a record with scaling applied
display(features_log_minmax_transform.head(n = 20))
```

```
        age         workclass education_level  education-num  \
0  0.301370         State-gov       Bachelors       0.800000
1  0.452055  Self-emp-not-inc       Bachelors       0.800000
2  0.287671           Private         HS-grad       0.533333
3  0.493151           Private            11th       0.400000
4  0.150685           Private       Bachelors       0.800000
5  0.273973           Private         Masters       0.866667
```

```
6   0.438356           Private          9th       0.266667
7   0.479452  Self-emp-not-inc      HS-grad       0.533333
8   0.191781           Private      Masters       0.866667
9   0.342466           Private    Bachelors       0.800000
10  0.273973           Private  Some-college      0.600000
11  0.178082         State-gov    Bachelors       0.800000
12  0.082192           Private    Bachelors       0.800000
13  0.205479           Private   Assoc-acdm       0.733333
14  0.232877           Private      7th-8th       0.200000
15  0.109589  Self-emp-not-inc      HS-grad       0.533333
16  0.205479           Private      HS-grad       0.533333
17  0.287671           Private         11th       0.400000
18  0.356164  Self-emp-not-inc      Masters       0.866667
19  0.315068           Private    Doctorate       1.000000

           marital-status         occupation    relationship  \
0           Never-married       Adm-clerical   Not-in-family
1      Married-civ-spouse    Exec-managerial         Husband
2                Divorced  Handlers-cleaners   Not-in-family
3      Married-civ-spouse  Handlers-cleaners         Husband
4      Married-civ-spouse     Prof-specialty            Wife
5      Married-civ-spouse    Exec-managerial            Wife
6   Married-spouse-absent      Other-service   Not-in-family
7      Married-civ-spouse    Exec-managerial         Husband
8           Never-married     Prof-specialty   Not-in-family
9      Married-civ-spouse    Exec-managerial         Husband
10     Married-civ-spouse    Exec-managerial         Husband
11     Married-civ-spouse     Prof-specialty         Husband
12          Never-married       Adm-clerical       Own-child
13          Never-married              Sales   Not-in-family
14     Married-civ-spouse   Transport-moving         Husband
15          Never-married    Farming-fishing       Own-child
16          Never-married  Machine-op-inspct       Unmarried
17     Married-civ-spouse              Sales         Husband
18               Divorced    Exec-managerial       Unmarried
19     Married-civ-spouse     Prof-specialty         Husband

                  race      sex  capital-gain  capital-loss  hours-per-week  \
0                White     Male      0.667492           0.0        0.397959
1                White     Male      0.000000           0.0        0.122449
2                White     Male      0.000000           0.0        0.397959
3                Black     Male      0.000000           0.0        0.397959
4                Black   Female      0.000000           0.0        0.397959
5                White   Female      0.000000           0.0        0.397959
6                Black   Female      0.000000           0.0        0.153061
7                White     Male      0.000000           0.0        0.448980
8                White   Female      0.829751           0.0        0.500000
9                White     Male      0.742849           0.0        0.397959
```

```
10               Black     Male      0.000000            0.0         0.806122
11   Asian-Pac-Islander     Male      0.000000            0.0         0.397959
12               White   Female      0.000000            0.0         0.295918
13               Black     Male      0.000000            0.0         0.500000
14   Amer-Indian-Eskimo     Male      0.000000            0.0         0.448980
15               White     Male      0.000000            0.0         0.346939
16               White     Male      0.000000            0.0         0.397959
17               White     Male      0.000000            0.0         0.500000
18               White   Female      0.000000            0.0         0.448980
19               White     Male      0.000000            0.0         0.602041

   native-country
0   United-States
1   United-States
2   United-States
3   United-States
4            Cuba
5   United-States
6         Jamaica
7   United-States
8   United-States
9   United-States
10  United-States
11          India
12  United-States
13  United-States
14         Mexico
15  United-States
16  United-States
17  United-States
18  United-States
19  United-States
```

0.5.3 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 [34]:
```
# TODO: One-hot encode the 'features_log_minmax_transform' data using pandas.get_dummie
features_final = pd.get_dummies(features_log_minmax_transform)

# TODO: Encode the 'income_raw' data to numerical values
income = income_raw.map({"<=50K": 0, ">50K":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)
```

```
103 total features after one-hot encoding.
['age', 'education-num', 'capital-gain', 'capital-loss', 'hours-per-week', 'workclass_ Federal-g
```

0.5.4 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 [35]:
```
# 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.
```

Note: this Workspace is running on `sklearn` v0.19. If you use the newer version (>="0.20"), the `sklearn.cross_validation` has been replaced with `sklearn.model_selection`.

0.6 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*.

0.6.1 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:

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

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).

0.6.2 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 [36]: TP = np.sum(income) # Counting the ones as this is the naive case. Note that 'income' i
         #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

         # TODO: Calculate accuracy, precision and recall
         accuracy = (TP+TN)/(TP+TN+FP+FN)
         recall = (TP)/(TP+FN)
         precision = (TP)/(TP+FP)

         # TODO: Calculate F-score using the formula above for beta = 0.5 and correct values for
         fscore = ((1+((0.5)**2))*(precision*recall))/((precision*((0.5)**2))+recall)

         # Print the results
         print("Naive Predictor: [Accuracy score: {:.4f}, F-score: {:.4f}]".format(accuracy, fsc
```

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

0.6.3 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

0.6.4 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 aboveˆ, with 4 parts for each of the three models you pick. Please include references with your answer.
**Answer: ”’
1-Logistic Regression

- we can apply Logistic Regression to classify plants according to their features
- the model will work well in the data which has lots of labels
- the model won’t work well in a nonlinear relationship between independent and dependant variables

Overall, Logistic Regression is a good choice since the labels are binary and the problem seems linear.
2- Decision Tree

- this model can use to classify electronic devices according to there shape, size, trade mark...etc
- the model is easy to implement and understand even for non-technical people

- the model could not work well if the data have a tiny variety

Overall, Decision Trees is a good choice since there are lots of independent variables, and this model can handle all of them easily to generate efficient rules.
3- AdaBoost

- this model could apply to determine students bass or fail in a course
- Boost model is working sequentially and every time it learns from the previous trained

- one of the disadvantages of AdaBoost is a weak learner

Overall, AdaBoost is a good choice even if it requires a longer time to train the data since it can deal with outliers and fit all the points perfectly.
''' **

0.6.5 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 [37]: # TODO: Import two metrics from sklearn - fbeta_score and accuracy_score
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import fbeta_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
             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 .pred
             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)

        # 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)

        # Success
        print("{} trained on {} samples.".format(learner.__class__.__name__, sample_size))

        # Return the results
        return results
```

0.6.6 Implementation: Initial Model Evaluation

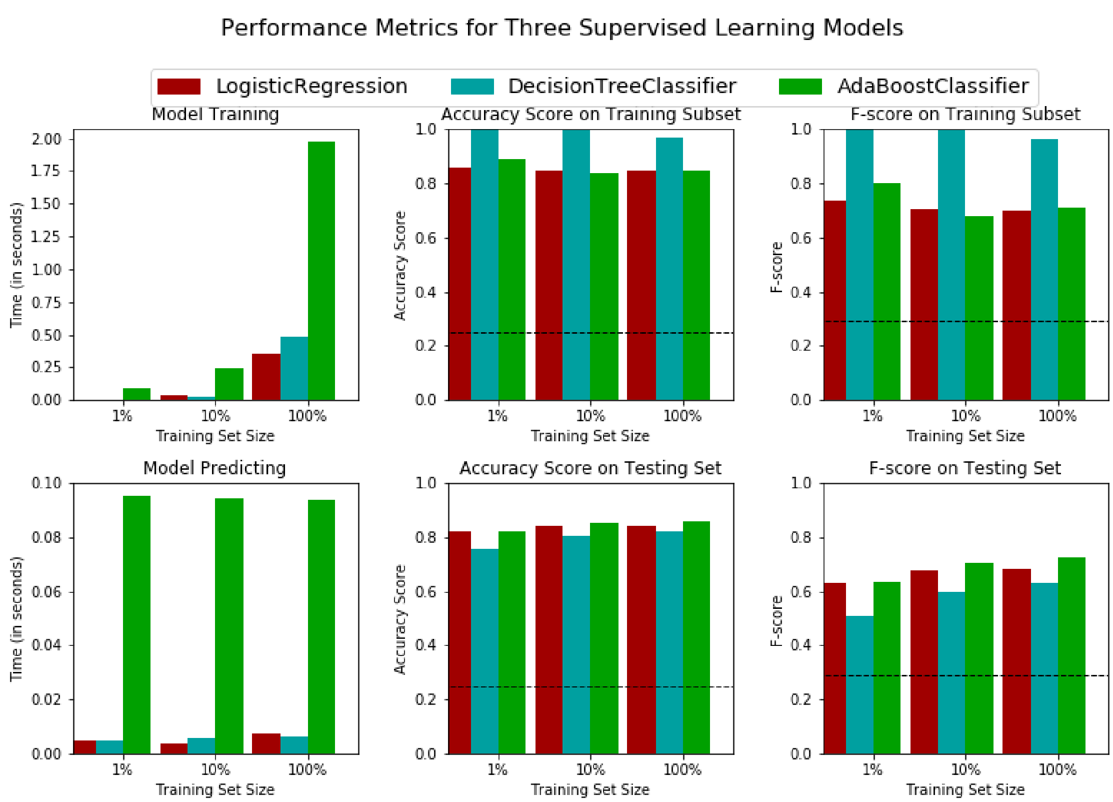
In the code cell, you will need to implement the following: - Import the three supervised learning models you've discussed in the previous section. - Initialize the three models and store them in `'clf_A'`, `'clf_B'`, and `'clf_C'`. - Use a `'random_state'` for each model you use, if provided. - **Note:** 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 [40]: # 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
         # TODO: Initialize the three models
         clf_A = LogisticRegression(random_state=42)
         clf_B = DecisionTreeClassifier(random_state=42)
         clf_C = AdaBoostClassifier(random_state=42)

         # 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 `
         # HINT: samples_1 is 1% of samples_100 (ensure to set the count of the values to be `in
         samples_100 = len(y_train)
         samples_10 = int(len(y_train)*(10/100))
```

```
samples_1 = int(len(y_train)*(1/100))

# 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)
```

```
LogisticRegression trained on 361 samples.
LogisticRegression trained on 3617 samples.
LogisticRegression trained on 36177 samples.
DecisionTreeClassifier trained on 361 samples.
DecisionTreeClassifier trained on 3617 samples.
DecisionTreeClassifier trained on 36177 samples.
AdaBoostClassifier trained on 361 samples.
AdaBoostClassifier trained on 3617 samples.
AdaBoostClassifier trained on 36177 samples.
```



0.7 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.

0.7.1 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:

- the highest F-score when 100% of the training data is used is for AdaBoost classifier
- logistic regression is the fastest model then Decision tree respectively
- the most appropriate model is the Adaboost classifier because it shows the highest F-score in testing data

**

0.7.2 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 AdaBoost classifier is a strong classifier that builds from multiple weak classifiers. It works stochastically, and each time the new model corrects the previous model until it becomes strong and efficient.

AdaBoost is a very successful algorithm developed for binary classification so, it will support the work.

It works as the following: 1- initialize equal weights for all the data point 2- increase the weight for the misclassified point after deploying the first model 3- repeat the second step and each time increase the weight for misclassified points 4- contain all the models together to become one strong model

Reference: Boosting in Machine Learning, Geeksforgeeks.org, 03-May-2019. [Online]. Available: https://www.geeksforgeeks.org/boosting-in-machine-learning-boosting-and-adaboost/. [Accessed: 12-Jun-2021].**

0.7.3 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 [41]:
```
# TODO: Import 'GridSearchCV', 'make_scorer', and any other necessary libraries
from sklearn.metrics import make_scorer
from sklearn.model_selection import GridSearchCV
# TODO: Initialize the classifier
clf = clf_C

# 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':[25,50,75,100],'learning_rate':[0.1,1,1.5,2],'algorithm':[

# 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 usin
grid_obj = GridSearchCV(clf, parameters, scoring=scorer)

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

# Get the estimator
best_clf = grid_fit.best_estimator_

# Make predictions using the unoptimized and 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, prediction
print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, predictions, beta =
print("\nOptimized Model\n------")
print("Final accuracy score on the testing data: {:.4f}".format(accuracy_score(y_test,
print("Final F-score on the testing data: {:.4f}".format(fbeta_score(y_test, best_predi
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  'precision', 'predicted', average, warn_for)
/opt/conda/lib/python3.6/site-packages/sklearn/metrics/classification.py:1135: UndefinedMetricWa
  '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: 0.8647
Final F-score on the testing data: 0.7382
```

0.7.4 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.

Metric	Unoptimized Model	Optimized Model
Accuracy Score	0.8576	0.8647
F-score	0.7246	0.7382

Results: Answer: the accuracy score in Optimized model is greater than Unoptimized Model also the F-score is improved in Optimized Model

---

0.8 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.

0.8.1 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? the most important features in my opinion in order are:

1- Work class 2- Native country 3- Hours per week 4- Occupation 5- Sex

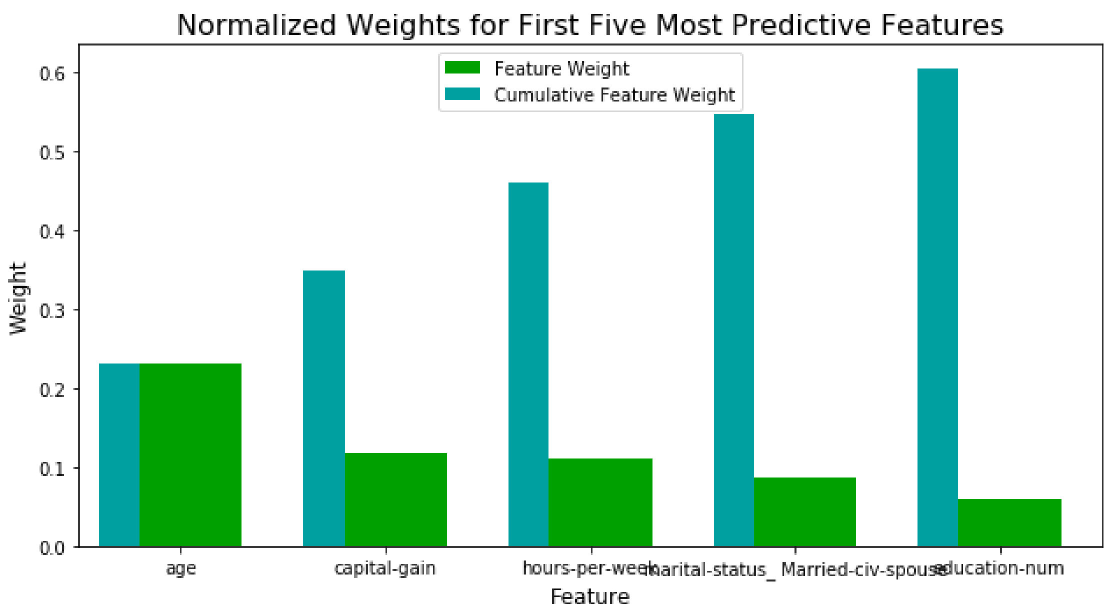
0.8.2 Implementation - Extracting Feature Importance

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 [42]: # 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()
         model.fit(X_train, y_train)
         p= model.predict(X_test)
         print(accuracy_score(y_test, p))
         # TODO: Extract the feature importances using .feature_importances_
         importances = model.feature_importances_

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

```
0.839137645108
```



0.8.3 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? I choose work class, occupation, and hours of a week since they determine how much the worker will gain. And they are different from one country to another and from gender to another

The figure shows different features than I was thought. These features are more relevant because they have the most weights

0.8.4 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 [43]:
```
# 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, be
print("\nFinal Model trained on reduced data\n------")
print("Accuracy on testing data: {:.4f}".format(accuracy_score(y_test, reduced_predicti
print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, reduced_predictions,
```

```
Final Model trained on full data
------
Accuracy on testing data: 0.8647
F-score on testing data: 0.7382

Final Model trained on reduced data
------
Accuracy on testing data: 0.8495
F-score on testing data: 0.7067
```

0.8.5 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?

the Accuracy score and F-score are less in the reduced data model, and if the time was a factor, I prefer to choose the reduced data model

> **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.

0.9 Before You Submit

You will also need run the following in order to convert the Jupyter notebook into HTML, so that your submission will include both files.

```
In [44]: !!jupyter nbconvert *.ipynb
```

```
Out[44]: ['[NbConvertApp] Converting notebook finding_donors.ipynb to html',
          '[NbConvertApp] Writing 532290 bytes to finding_donors.html']
```