finding_donors

October 27, 2017

1 Machine Learning Engineer Nanodegree

1.1 Supervised Learning

1.2 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.

1.3 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.

1.4 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 [1]: # 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
        # The data has been orderd by Income Level DESC,
        # Capital Loss ASC, Age DESC, Capital Gain DESC, Hours per week DESC,
        # Education Num DESC
        data = pd.read_csv("census_ordered.csv")

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

```
   age workclass education_level  education-num       marital-status  \
0   90   Private     Prof-school             15   Married-civ-spouse

        occupation relationship    race    sex  capital-gain  capital-loss  \
0   Prof-specialty      Husband   White   Male         20051             0

   hours-per-week  native-country income
0              72   United-States   >50K
```

1.4.1 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 [8]: # TODO: Total number of records
        n_records = len(data)

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

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

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

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

```
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%
```

1.5 Preparing the Data

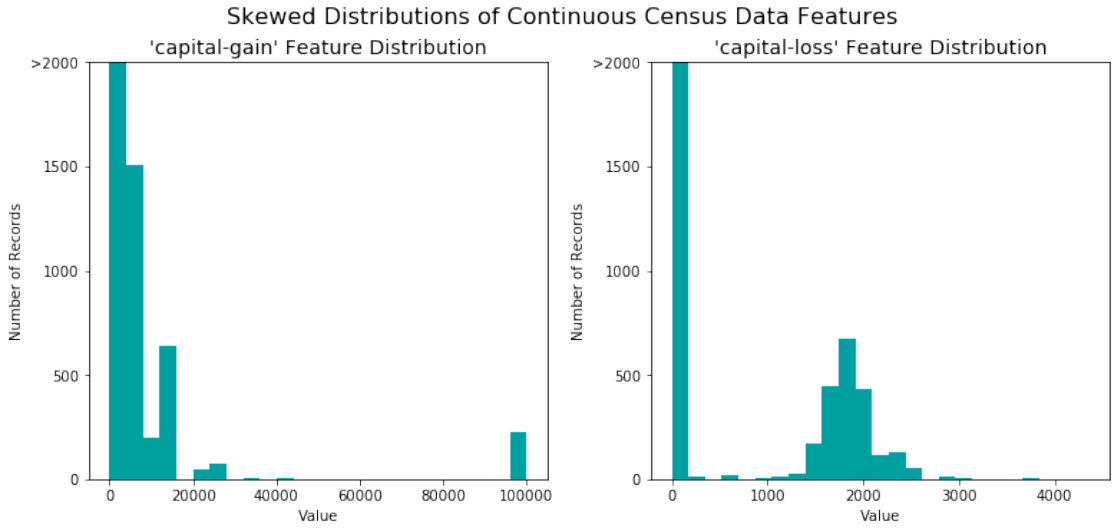
Before data can be used as input for machine learning algorithms, it often must be cleaned, formatted, and restructured — this is typically known as **preprocessing**. Fortunately, for this dataset, there are no invalid or missing entries we must deal with, however, there are some qualities about certain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

1.5.1 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.

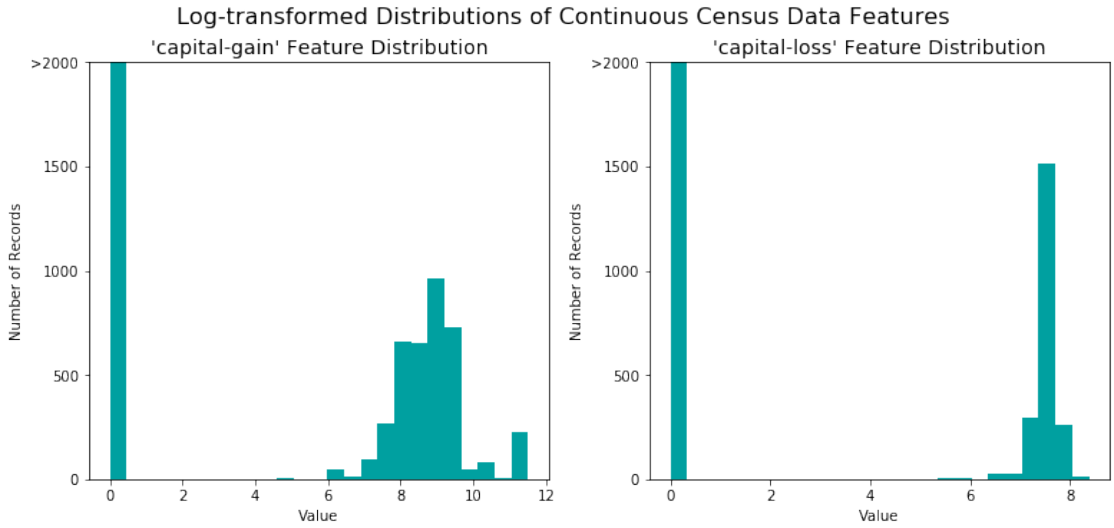
In [3]:
```
# Split the data into features and target label
income_raw = data['income']
features_raw = data.drop('income', axis = 1)

# Visualize skewed continuous features of original data
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 [4]:
```
# Log-transform the skewed features
skewed = ['capital-gain', 'capital-loss']
features_raw[skewed] = data[skewed].apply(lambda x: np.log(x + 1))

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



1.5.2 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 [5]: # Import sklearn.preprocessing.StandardScaler
        from sklearn.preprocessing import MinMaxScaler

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

        # Show an example of a record with scaling applied
        display(features_raw.head(n = 1))
```

```
   age workclass education_level  education-num       marital-status  \
0  1.0   Private     Prof-school       0.933333   Married-civ-spouse

        occupation relationship    race    sex  capital-gain  capital-loss  \
0   Prof-specialty      Husband   White   Male      0.200512           0.0

   hours-per-week  native-country
0         0.72449   United-States
```

1.5.3 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_raw'` data. - Convert the target label `'income_raw'` to numerical entries. - Set records with "<=50K" to `0` and records with ">50K" to `1`.

In [6]:
```
# TODO: One-hot encode the 'features_raw' data using pandas.get_dummies()
features = pd.get_dummies(features_raw)

# TODO: Encode the 'income_raw' data to numerical values
income_encoded = pd.get_dummies(income_raw)
income_transformation_matrix = np.array([0, 1])
income = income_encoded * income_transformation_matrix
income = income.sum(axis=1)

# Print the number of features after one-hot encoding
encoded = list(features)
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-
```

1.5.4 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 [7]: # Import train_test_split
        from sklearn.model_selection 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, income, test_size = 0.2,

        # 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.
```

1.6 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*.

1.6.1 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 $\mathbf{F}_{0.5}$ **score** (or F-score for simplicity).

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

1.6.2 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?
Note: You must use the code cell below and assign your results to `'accuracy'` and `'fscore'` to be used later.

```
In [11]: from sklearn.metrics import accuracy_score, fbeta_score

         # TODO: Calculate accuracy
         accuracy = float(n_greater_50k / n_records)

         precision = n_greater_50k / (n_at_most_50k + n_greater_50k)

         recall = n_greater_50k / (n_greater_50k + 0)

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

         beta = 1.0
         f1_score = (1 + beta**2) * (precision * recall) / (beta**2 * precision + recall)

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

         print("Naive Predictor: [F1-score]: {:.4f}]".format(f1_score))
```

```
Naive Predictor: [Accuracy score: 0.2478, F-score: 0.2917]
Naive Predictor: [F1-score]: 0.3972]
```

1.6.3 Supervised Learning Models

The following supervised learning models 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

1.6.4 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.* (You may need to do research for this — give references!) - *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?*

Answer:

Support Vector Machines

One of the examples that Support Vector Machines is capable to do is to analyse the signatures that a metamorphic malwares can produce. The goal of the SVM is to deduce the outcome of running the metamorphic malwares on the affected system and to classify whether there has been significant effect or not.

****Strengths****

****A support vector machine is more suitable when it is relevant for the domain to be dependent with the feature weights and the decision vectors produced by those feature weights. An SVM performs well for a low C value (error factor) so that overfitting can be prevented, and peforms well with greater relevance of the feature with the data. An SVM can be optimised with the help of a Stochastic Gradient Descent by minimising the cost function, and also by using the SMO algorithm which acts like a decision tree at certain cases. SVM parameters make use of a vector form to represent the algorithm, which helps in early selection of the hyperplane without considering the entire set of features.****

****Weaknesses****

****An SVM happens to be expensive when the interdependency between the features must be considered to a level where there is no need for a hyperplane to calculate the functional and geometric distances. Unlike other algorithms, there is no usual way of expectation - maximisation techniques applied when the distances are calculated.****

****Problem Analysis****

The features have been distributed into a numerical set and a sparse set. The SVM for sparse feature set is effective when the geometric margins are vectorized. SVC is capable to vectorize the feature sets while classification.* _____

K-Nearest Neighbors

K-Nearest neighbors can be used to classify Asset Valuation of a business using Cash Inflows and Outflows into multiple operating departments in order to find a mapping between the Class and the Role that the class is playing in Cash Flow. This classification can be used to find the relevant distance vector for the Cash Inflow and Outflow using a distance query such that suitable prediction can be made using the classes against the training data. Regression on the time based cash flows is made across all classes using the distance value obtained from the features of the training data.

****Strengths****

****The K-Nearest Neighbors Algorithm is capable of predictions using suitable measures that can be formed from the different entities that are considered for analysis. The measures can be varied and each entity can be grouped for classifying the data into a multi class or multi label classification. K-Nearest Neighbor will perform well when the dataset is large as the time taken to analyse the problem is short compared to the SVM. K-Nearest Neighbor requires formation of hypothesis to measure the distance value, which may be applicable to only few data sets.****

****Weaknesses****

****One of the weaknesses of the algorithm is that the nearest neighbors must be specified and is dependent on the feature space. K-Nearest neighbor algorithm needs to be optimised based on the number of neighbors using a GridSearchCV by varying the value of the leaf_size parameter.****

****Problem Analysis****

****In the charity ML example it is difficult to find what is the order determining the performance or accuracy of the algorithm. The KNN algorithm indexes the data and hierarchically classifies the features from the feature set. It is also effective when only numerical features are used.*** _____

Logistic Regression

Logistic regression determines the outcome based on an activation function. Logistic Regression can be used to find the risk of cancer due to smoking. It maps the supervised data to a set of feature sets of the cigarette smokers fits the curve based on a known outcome of cancer.

Strengths

****The logistic regression is a highly performant supervised learning algorithm. It either produces values which are close to 1 or close to 0. It is effective when the features set size is large compared to KNN and SVC which show issues in performance. The relationa between the feature sets are linear but the logit function is exponential. Logistic regression provides several alternatives for activation functions.***

Weaknesses

****It is a generalised algorithm and not an algorithm that sets rules or case-based algorithm. The algorithm will crumble if the solution must be approched in a diagnostic perpective.****

****Problem Analysis****

****The charity ML data offers several feature sets which are not used fully in several cases. In determining the right outcome of being an actual donor for charity ML, the algorithm focuses on maximising the likelihood of being a donor.**** _____

References

http://journal.uad.ac.id/index.php/TELKOMNIKA/article/viewFile/3850/2763
http://www.cig.ase.ro/articles/14_1_4.pdf
http://people.stern.nyu.edu/jsimonof/classes/2301/pdf/logistic.pdf

1.6.5 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 [14]: # TODO: Import two metrics from sklearn - fbeta_score and accuracy_score
         from sklearn.metrics import accuracy_score, fbeta_score
         from sklearn.naive_bayes import GaussianNB

         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'
    start = time() # Get start time
    learner.fit(X_train[0:sample_size], y_train[0: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,
    #       then get predictions on the first 300 training samples
    start = time() # Get start time
    predictions_test = learner.predict(X_test)
    predictions_train = learner.predict(X_train[0:300])
    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
    results['acc_train'] = accuracy_score(y_train[0:300], predictions_train)

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

    # TODO: Compute F-score on the the first 300 training samples
    results['f_train'] = fbeta_score(y_train[0:300], predictions_train, 0.5)

    # TODO: Compute F-score on the test set
    results['f_test'] = fbeta_score(y_test, predictions_test, 0.5)

    results['pred_value'] = predictions_train

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

    # Return the results
    return results
```

1.6.6 Implementation: Initial Model Evaluation

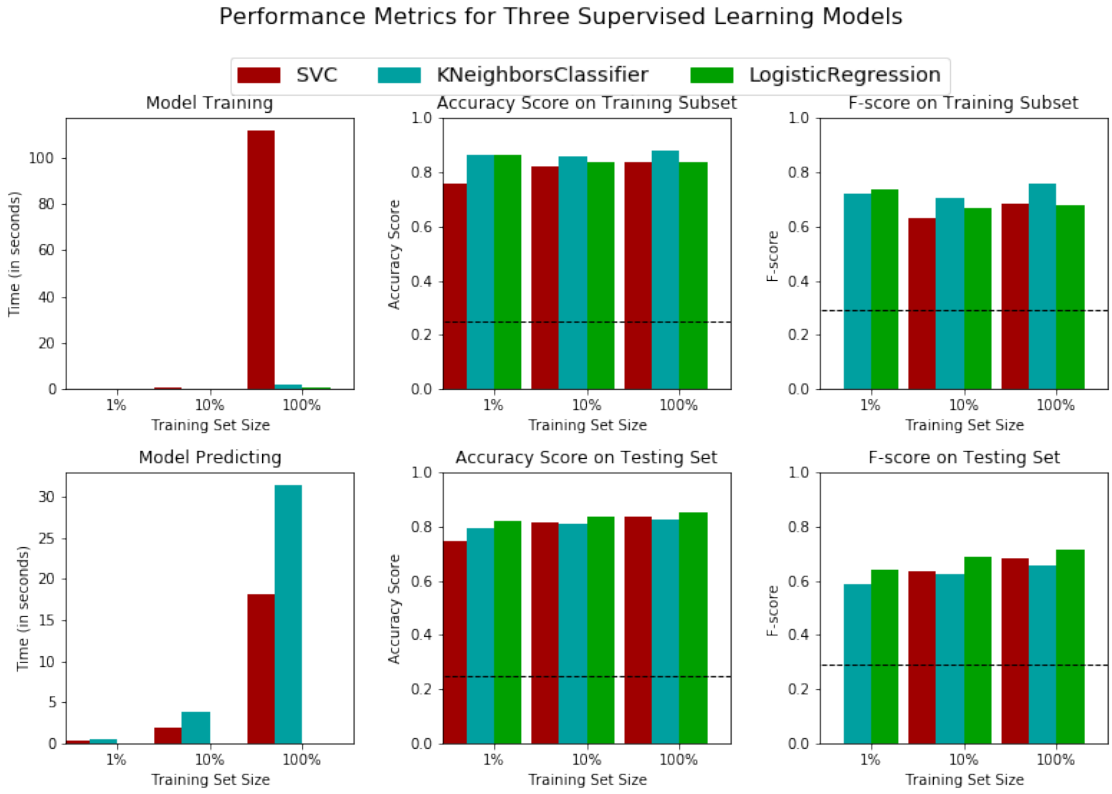
In the code cell, you will need to implement the following: - Import the three supervised learning models you've discussed in the previous section. - Initialize the three models and store them in `'clf_A'`, `'clf_B'`, and `'clf_C'`. - Use a `'random_state'` for each model you use, if provided. - **Note:** 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 [15]:
```
# TODO: Import the three supervised learning models from sklearn
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

# TODO: Initialize the three models
clf_A = SVC()
clf_B = KNeighborsClassifier(n_neighbors=5)
clf_C = LogisticRegression()

# TODO: Calculate the number of samples for 1%, 10%, and 100% of the training data
n = len(X_train)
samples_1 = int(round(0.01 * n))
samples_10 = int(round(0.1 * n))
samples_100 = n

# 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, sample_size in enumerate([samples_1, samples_10, samples_100]):
        results[clf_name][i] = \
        train_predict(clf, sample_size, X_train, y_train, X_test, y_test)

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

```
C:\Users\Aswin\Anaconda3\lib\site-packages\sklearn\metrics\classification.py:1113: UndefinedMe
  'precision', 'predicted', average, warn_for)


SVC trained on 362 samples.
SVC trained on 3618 samples.
SVC trained on 36177 samples.
KNeighborsClassifier trained on 362 samples.
KNeighborsClassifier trained on 3618 samples.
KNeighborsClassifier trained on 36177 samples.
LogisticRegression trained on 362 samples.
LogisticRegression trained on 3618 samples.
LogisticRegression trained on 36177 samples.
```



1.7 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.

1.7.1 Question 3 - Choosing the Best Model

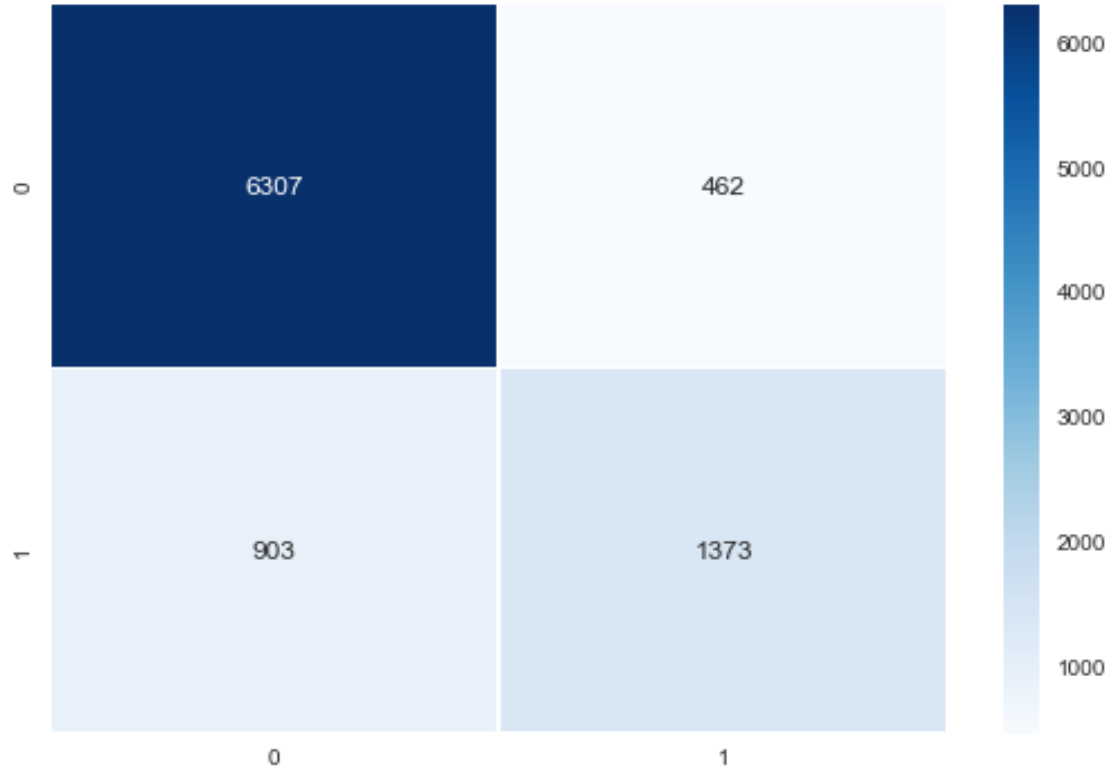
Based on the evaluation you performed earlier, in one to two paragraphs, explain to CharityML* which of the three models you believe to be most appropriate for the task of identifying individuals that make more than $50,000.*

Hint: Your answer should include discussion of the metrics, prediction/training time, and the algorithm's suitability for the data.

```
In [16]: from sklearn.metrics import accuracy_score
         import seaborn
         import matplotlib.pyplot as plt

In [18]: print("Confusion Matrix for Logistic: ")
```

```
        y_pred_logit = clf_C.predict(X_test)
        accuracy = accuracy_score(y_test.values, y_pred_logit)

        from sklearn.metrics import confusion_matrix

        confusion = confusion_matrix(y_test.values, y_pred_logit)

        f, ax = plt.subplots(figsize=(9, 6))
        seaborn.heatmap(confusion, cmap="Blues",  annot=True, fmt="d", linewidths=.5, ax=ax)

        print("Accuracy score for Logit: ")

        print(accuracy)
```

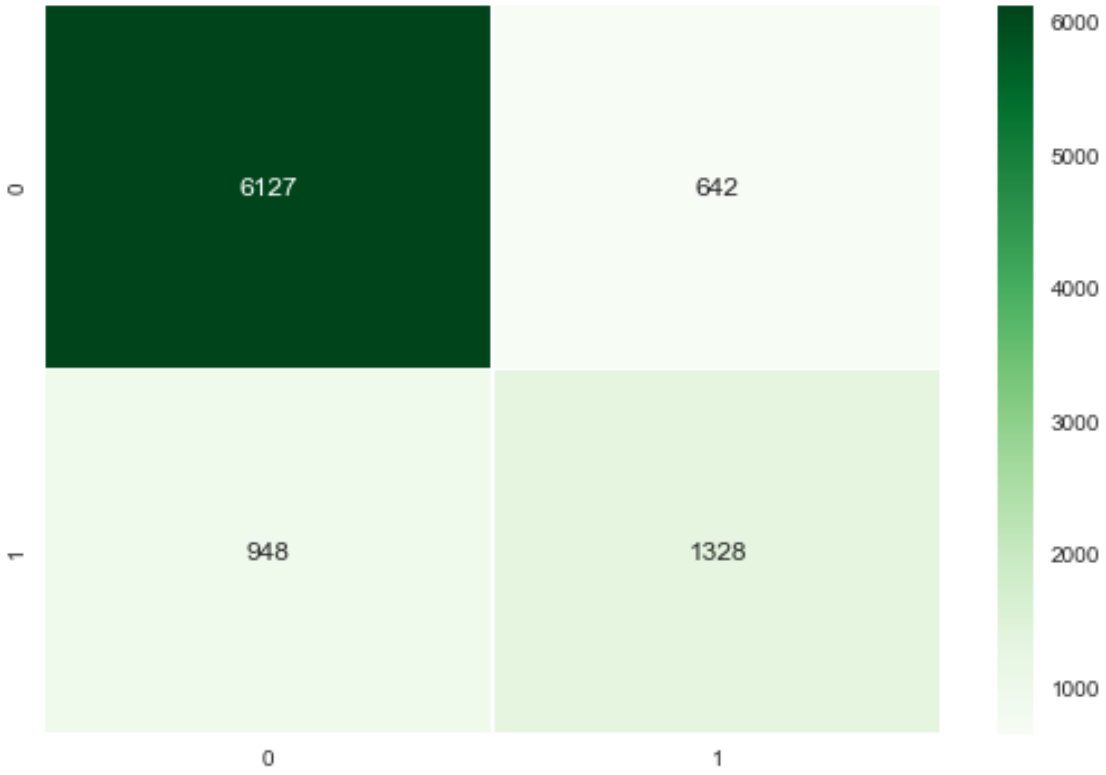
```
Confusion Matrix for Logistic:
Accuracy score for Logit:
0.849087893864
```




```
In [19]: print("Confusion Matrix for K-Nearest: ")

        y_pred_knn = clf_B.predict(X_test)
        accuracy = accuracy_score(y_test.values, y_pred_knn)
```

```
        from sklearn.metrics import confusion_matrix

        confusion = confusion_matrix(y_test.values, y_pred_knn)

        f, ax = plt.subplots(figsize=(9, 6))
        seaborn.heatmap(confusion, cmap="Greens",  annot=True, fmt="d", linewidths=.5, ax=ax)

        print("Accuracy score for KNN: ")

        print(accuracy)
```

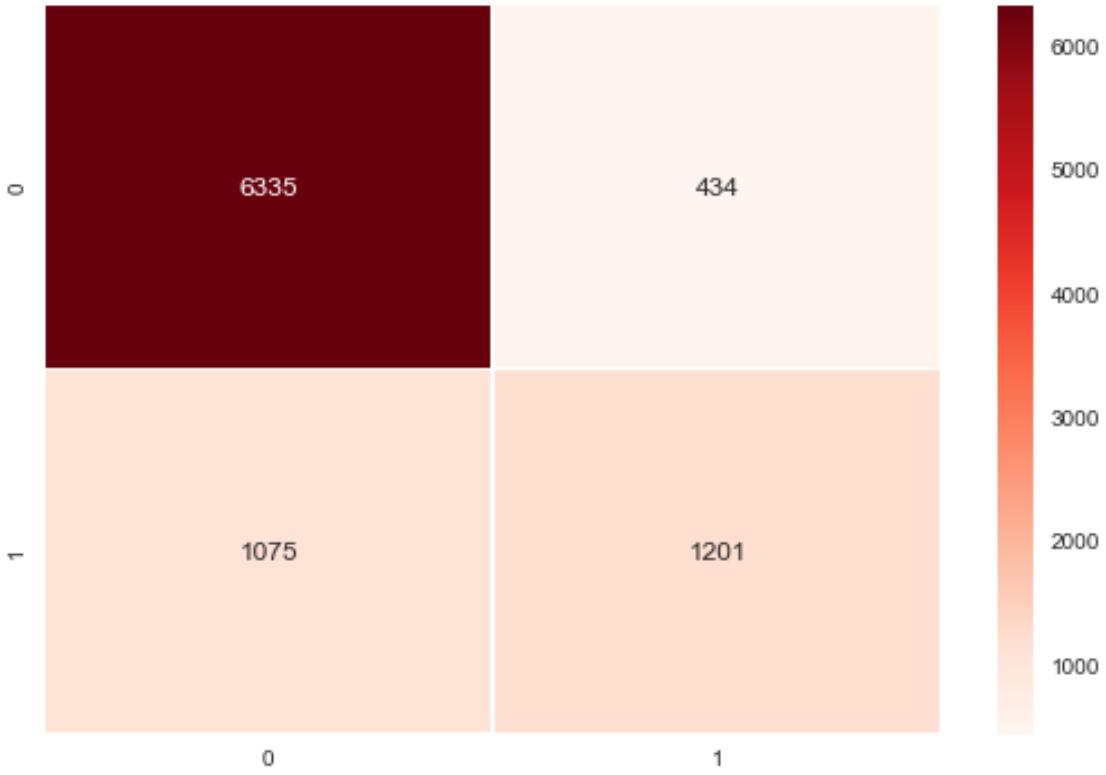
```
Confusion Matrix for K-Nearest:
Accuracy score for KNN:
0.824212271973
```




```
In [20]: print("Confusion Matrix for SVC: ")

         y_pred_svc = clf_A.predict(X_test)
         accuracy = accuracy_score(y_test.values, y_pred_svc)

         from sklearn.metrics import confusion_matrix
```

```
        confusion = confusion_matrix(y_test.values, y_pred_svc)

        f, ax = plt.subplots(figsize=(9, 6))
        seaborn.heatmap(confusion, cmap="Reds",  annot=True, fmt="d", linewidths=.5, ax=ax)

        print("Accuracy score for SVC: ")

        print(accuracy)
```

```
Confusion Matrix for SVC:
Accuracy score for SVC:
0.833167495854
```




```
In [22]: income_level = []
         for idx in range(len(y_test)):
             if y_test.values[idx] == 1:
                 income_level.append(idx)

         filtered_y_values = y_test.iloc[income_level]
         filtered_X_test = X_test.iloc[income_level]
```

```
filtered_y_pred_logit = y_pred_logit[income_level]
filtered_y_pred_knn = y_pred_knn[income_level]
filtered_y_pred_svc = y_pred_svc[income_level]

print("Accuracy for Predictor, Logistic: ", accuracy_score(filtered_y_values, filtered
print("F-score for Predictor, Logistic: ", fbeta_score(filtered_y_values, filtered_y_

print("Accuracy for Predictor, KNN: ", accuracy_score(filtered_y_values, filtered_y_p
print("F-score for Predictor, KNN: ", fbeta_score(filtered_y_values, filtered_y_pred_

print("Accuracy for Predictor, SVC: ", accuracy_score(filtered_y_values, filtered_y_p
print("F-score for Predictor, SVC: ", fbeta_score(filtered_y_values, filtered_y_pred_
```

```
Accuracy for Predictor, Logistic:  0.603251318102
F-score for Predictor, Logistic:  0.883753861998
Accuracy for Predictor, KNN:  0.583479789104
F-score for Predictor, KNN:  0.875065893516
Accuracy for Predictor, SVC:  0.527680140598
F-score for Predictor, SVC:  0.848163841808
```

Answer:

Out of these classifiers, the Logistic Regression is found to be the most appropriate. The accuracy and fscore of the model is the highest and hence it isthe best chosen model.

The objective of the Charity Donor ML is to find the best possible donor so accuracy score is more important.

The prediction time for KNN is the highest and the training time for SVC is the highest. Logistic Regression has the least training time and the least prediction time. SVM usually takes a long time because of the tedious task of deriving the optimum model based on the vector, functional and geometric margins.

Logistic Regression maximises the likelihood using gradient descent or other suitable algorithms. The charity donor ML requires donors who can prolong their donation schemes, and hence the Logistic Regression is a good model because it relates the set of data to the outcome with a maximised likelihood.

1.7.2 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 or technical jargon, such as describing equations or discussing the algorithm implementation.*

In [23]:
```
# For data above $50,000
y_pred = clf_C.predict(filtered_X_test)
from sklearn.metrics import r2_score, f1_score

print("R2 Coefficient of Determination: ", r2_score(filtered_y_values, filtered_y_pred

print("F1 Score: ", f1_score(filtered_y_values, filtered_y_pred_logit))
```

```
R2 Coefficient of Determination:  0.0
F1 Score:  0.75253494108
```

```
In [24]: display(data.describe())

         display(features.describe())
```

	age	education-num	capital-gain	capital-loss	hours-per-week
count	45222.000000	45222.000000	45222.000000	45222.000000	45222.000000
mean	38.547941	10.118460	1101.430344	88.595418	40.938017
std	13.217870	2.552881	7506.430084	404.956092	12.007508
min	17.000000	1.000000	0.000000	0.000000	1.000000
25%	28.000000	9.000000	0.000000	0.000000	40.000000
50%	37.000000	10.000000	0.000000	0.000000	40.000000
75%	47.000000	13.000000	0.000000	0.000000	45.000000
max	90.000000	16.000000	99999.000000	4356.000000	99.000000

	age	education-num	capital-gain	capital-loss	\
count	45222.000000	45222.000000	45222.000000	45222.000000	
mean	0.295177	0.607897	0.011014	0.020339	
std	0.181067	0.170192	0.075065	0.092965	
min	0.000000	0.000000	0.000000	0.000000	
25%	0.150685	0.533333	0.000000	0.000000	
50%	0.273973	0.600000	0.000000	0.000000	
75%	0.410959	0.800000	0.000000	0.000000	
max	1.000000	1.000000	1.000000	1.000000	

	hours-per-week	workclass_ Federal-gov	workclass_ Local-gov	\
count	45222.000000	45222.000000	45222.000000	
mean	0.407531	0.031091	0.068551	
std	0.122526	0.173566	0.252691	
min	0.000000	0.000000	0.000000	
25%	0.397959	0.000000	0.000000	
50%	0.397959	0.000000	0.000000	
75%	0.448980	0.000000	0.000000	
max	1.000000	1.000000	1.000000	

	workclass_ Private	workclass_ Self-emp-inc	\
count	45222.000000	45222.000000	
mean	0.736522	0.036398	
std	0.440524	0.187281	
min	0.000000	0.000000	
25%	0.000000	0.000000	
50%	1.000000	0.000000	
75%	1.000000	0.000000	
max	1.000000	1.000000	

```
       workclass_ Self-emp-not-inc             ...              \
count                 45222.000000             ...
mean                      0.083941             ...
std                       0.277303             ...
min                       0.000000             ...
25%                       0.000000             ...
50%                       0.000000             ...
75%                       0.000000             ...
max                       1.000000             ...

       native-country_ Portugal  native-country_ Puerto-Rico  \
count              45222.000000                 45222.000000
mean                   0.001371                     0.003870
std                    0.037002                     0.062088
min                    0.000000                     0.000000
25%                    0.000000                     0.000000
50%                    0.000000                     0.000000
75%                    0.000000                     0.000000
max                    1.000000                     1.000000

       native-country_ Scotland  native-country_ South  \
count              45222.000000           45222.000000
mean                   0.000442               0.002233
std                    0.021026               0.047207
min                    0.000000               0.000000
25%                    0.000000               0.000000
50%                    0.000000               0.000000
75%                    0.000000               0.000000
max                    1.000000               1.000000

       native-country_ Taiwan  native-country_ Thailand  \
count            45222.000000              45222.000000
mean                 0.001216                  0.000641
std                  0.034854                  0.025316
min                  0.000000                  0.000000
25%                  0.000000                  0.000000
50%                  0.000000                  0.000000
75%                  0.000000                  0.000000
max                  1.000000                  1.000000

       native-country_ Trinadad&Tobago  native-country_ United-States  \
count                     45222.000000                   45222.000000
mean                          0.000575                       0.913095
std                           0.023971                       0.281698
min                           0.000000                       0.000000
25%                           0.000000                       1.000000
50%                           0.000000                       1.000000
```

```
75%                          0.000000                          1.000000
max                          1.000000                          1.000000

       native-country_ Vietnam  native-country_ Yugoslavia
count              45222.000000                45222.000000
mean                   0.001835                    0.000509
std                    0.042803                    0.022547
min                    0.000000                    0.000000
25%                    0.000000                    0.000000
50%                    0.000000                    0.000000
75%                    0.000000                    0.000000
max                    1.000000                    1.000000

[8 rows x 103 columns]
```

Answer:

The R2 score of 0.0 implies that the model obtained from the Logistic Regression is a constant model that predicts the expectation of y always.

The logistic regresion is a better model because it can go deeper into the statistical observations of the dataset. The statistical description of the dataset provides the range of feature set. This range of data is then fitted into the logistic regression curve. The logistic regression then calculates the likelihood of the data mapped to the outcome and then iterates the search parameters such that the derivative of the cost function is equated to zero.

The accuracy of the fitted curve using Logistic Regression is usually high because it is closely fitted to the true outcome. Logistic Regression does not memorize the results and finds the relevant coefficients or search parameters for the features.

1.7.3 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 [33]: # TODO: Import 'GridSearchCV', 'make_scorer', and any other necessary libraries
         from sklearn.model_selection import GridSearchCV
         from sklearn.metrics import make_scorer
         from sklearn.metrics import fbeta_score
```

```
        from sklearn.svm import SVC

        # TODO: Initialize the classifier
        clf = LogisticRegression()
        clf_name = clf.__class__.__name__

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

        # TODO: Create the parameters list you wish to tune
        parameters = {'solver': ['liblinear', 'sag'], 'C': [1.0, 0.5, 1.5, 3.5]}

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

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

        # # Get the estimator
        best_clf = grid_fit.best_estimator_

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

        # Report the before-and-afterscores
        print("Unoptimized model\n------")
        print("Accuracy score on testing data: {:.4f}".format(accuracy_score(y_test.values, pr
        print("F-score on testing data: {:.4f}".format(fbeta_score(y_test.values, predictions
        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.values, be

        print(grid_fit.best_estimator_)
```

```
Unoptimized model
------
Accuracy score on testing data: 0.8491
F-score on testing data: 0.7139

Optimized Model
------
Final accuracy score on the testing data: 0.8493
Final F-score on the testing data: 0.7144
LogisticRegression(C=3.5, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

1.7.4 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	Benchmark Predictor	Unoptimized Model	Optimized Model
Accuracy Score	0.2478	0.8491	0.8493
F-score	0.2917	0.7139	0.7144

Results: Answer:

The accuracy score and fscore is greater for the optimized model by a slightly small margin.

Naive Bayes predictor considers only the Bayesian estimation which is suitable for predictable distributions with relevant parameters of estimation.

Logistic Regression surpases the benchmark predictior by fitting a curve with the outcome of income greater than 50K. The final Logistic Regression model is having a solver using 'liblinear' and the regularization factor of 3.5.

1.8 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.

1.8.1 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?

Answer:

The five features that are important are:

1. Occupation

2. Workclass

3. Education Level
4. Capital Gain
5. Education Number of Years

The occupation is one of the most important feature as it is likely that the charitable organisations tie up with the companies and the employees. The work class is slightly dependent due to the decisions taken on spending money for charities or seeking schemes related to charitable organisations. The education level is an important feature because it determines who the person is. The capital gain is important because it is likely that if the capital gain is more, there will be more investments in charity. The donations are dependent on the education number of years because it is likely that the individual spenbs time in charitable foundations.

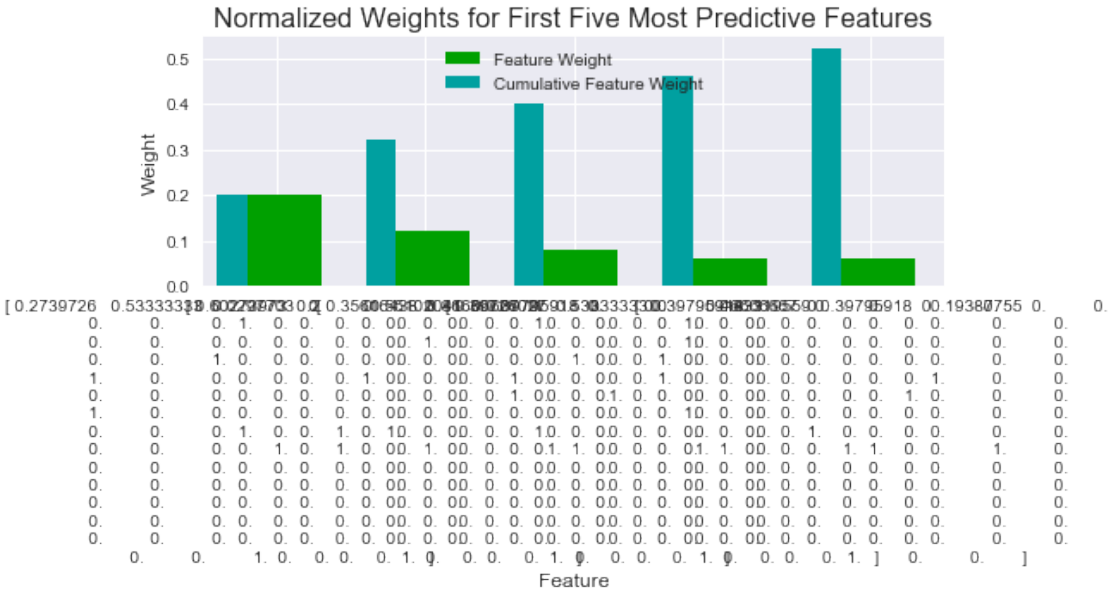
1.8.2 Implementation - Extracting Feature Importance

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 [38]:
```
# TODO: Import a supervised learning model that has 'feature_importances_'
from sklearn.ensemble import AdaBoostClassifier

# TODO: Train the supervised model on the training set
model = AdaBoostClassifier()

# TODO: Extract the feature importances
classifier = model.fit(X_train, y_train)

print("--- The feature importances of AdaBoost ---")
print(classifier.feature_importances_)

# # Plot
vs.feature_plot(classifier.feature_importances_, X_train.values, y_train.values)
```

```
--- The feature importances of AdaBoost ---
[ 0.12  0.06  0.08  0.2   0.06  0.02  0.    0.    0.    0.02  0.    0.    0.
  0.    0.    0.    0.    0.    0.    0.    0.    0.02  0.    0.02  0.02
  0.    0.    0.    0.    0.02  0.02  0.    0.    0.    0.02  0.    0.    0.
  0.02  0.02  0.02  0.02  0.02  0.    0.02  0.02  0.    0.02  0.    0.
  0.02  0.    0.02  0.    0.04  0.    0.    0.    0.    0.02  0.02  0.02
  0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
  0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
  0.02  0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
  0.    0.    0.  ]
```



1.8.3 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?*

Answer:

**The features obtained from feature importances are: - Capital Loss - Age - Capital Gain - Hours per week - Education Num

The largest factor is 0.2 which is for the Capital Loss feature, the second is Age, the third is Capital Gain, the fourth is Hours per week, the fifth is Education Num (0.04).

My answer is not close to the prediction features.

Greater the Capital loss implies less likely the donation will be, lesser the capital loss, more donations would be prominent. Greater the age, highly likely the donations will be because the income level will be the highest. More the Capital Gain, greater the donations because the satisfaction rate and savings would be higher for people who have more capital gain and they are relevant sources. More hours per week implies more savings and hence greater likelihood for donating. Greater the education number of years, greater the likelihood of people speding time and investing time for charitable foundations.

1.8.4 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 [39]: # 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(classifier.feature_impor
         X_test_reduced = X_test[X_test.columns.values[(np.argsort(classifier.feature_importan

         # 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_predictio
         print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, best_predictions,
         print("\nFinal Model trained on reduced data\n------")
         print("Accuracy on testing data: {:.4f}".format(accuracy_score(y_test, reduced_predic
         print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, reduced_prediction
```

```
Final Model trained on full data
------
Accuracy on testing data: 0.8493
F-score on testing data: 0.7144

Final Model trained on reduced data
------
Accuracy on testing data: 0.8083
F-score on testing data: 0.6155
```

1.8.5 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?

```
In [28]: display(data.describe())

         display(features.describe())
```

```
                age  education-num  capital-gain  capital-loss  hours-per-week
count  45222.000000   45222.000000  45222.000000  45222.000000    45222.000000
mean      38.547941      10.118460   1101.430344     88.595418       40.938017
```

```
std       13.217870       2.552881   7506.430084    404.956092       12.007508
min       17.000000       1.000000      0.000000      0.000000        1.000000
25%       28.000000       9.000000      0.000000      0.000000       40.000000
50%       37.000000      10.000000      0.000000      0.000000       40.000000
75%       47.000000      13.000000      0.000000      0.000000       45.000000
max       90.000000      16.000000  99999.000000   4356.000000       99.000000


                age  education-num  capital-gain  capital-loss  \
count  45222.000000   45222.000000  45222.000000  45222.000000
mean       0.295177       0.607897      0.011014      0.020339
std        0.181067       0.170192      0.075065      0.092965
min        0.000000       0.000000      0.000000      0.000000
25%        0.150685       0.533333      0.000000      0.000000
50%        0.273973       0.600000      0.000000      0.000000
75%        0.410959       0.800000      0.000000      0.000000
max        1.000000       1.000000      1.000000      1.000000

       hours-per-week  workclass_ Federal-gov  workclass_ Local-gov  \
count    45222.000000            45222.000000          45222.000000
mean         0.407531                0.031091              0.068551
std          0.122526                0.173566              0.252691
min          0.000000                0.000000              0.000000
25%          0.397959                0.000000              0.000000
50%          0.397959                0.000000              0.000000
75%          0.448980                0.000000              0.000000
max          1.000000                1.000000              1.000000

       workclass_ Private  workclass_ Self-emp-inc  \
count        45222.000000             45222.000000
mean             0.736522                 0.036398
std              0.440524                 0.187281
min              0.000000                 0.000000
25%              0.000000                 0.000000
50%              1.000000                 0.000000
75%              1.000000                 0.000000
max              1.000000                 1.000000

       workclass_ Self-emp-not-inc             ...              \
count                 45222.000000             ...
mean                      0.083941             ...
std                       0.277303             ...
min                       0.000000             ...
25%                       0.000000             ...
50%                       0.000000             ...
75%                       0.000000             ...
max                       1.000000             ...
```

```
       native-country_ Portugal  native-country_ Puerto-Rico  \
count              45222.000000                 45222.000000
mean                   0.001371                     0.003870
std                    0.037002                     0.062088
min                    0.000000                     0.000000
25%                    0.000000                     0.000000
50%                    0.000000                     0.000000
75%                    0.000000                     0.000000
max                    1.000000                     1.000000

       native-country_ Scotland  native-country_ South  \
count              45222.000000           45222.000000
mean                   0.000442               0.002233
std                    0.021026               0.047207
min                    0.000000               0.000000
25%                    0.000000               0.000000
50%                    0.000000               0.000000
75%                    0.000000               0.000000
max                    1.000000               1.000000

       native-country_ Taiwan  native-country_ Thailand  \
count            45222.000000              45222.000000
mean                0.001216                  0.000641
std                 0.034854                  0.025316
min                 0.000000                  0.000000
25%                 0.000000                  0.000000
50%                 0.000000                  0.000000
75%                 0.000000                  0.000000
max                 1.000000                  1.000000

       native-country_ Trinadad&Tobago  native-country_ United-States  \
count                     45222.000000                   45222.000000
mean                          0.000575                       0.913095
std                           0.023971                       0.281698
min                           0.000000                       0.000000
25%                           0.000000                       1.000000
50%                           0.000000                       1.000000
75%                           0.000000                       1.000000
max                           1.000000                       1.000000

       native-country_ Vietnam  native-country_ Yugoslavia
count              45222.000000                45222.000000
mean                   0.001835                    0.000509
std                    0.042803                    0.022547
min                    0.000000                    0.000000
25%                    0.000000                    0.000000
50%                    0.000000                    0.000000
75%                    0.000000                    0.000000
```

```
max                   1.000000                    1.000000

[8 rows x 103 columns]
```

Answer:

The F-score of the final model describes the bias that model with full data has against the census data. It omits the features that are showing low value for the coefficient.

The accuracy score is also lesser for the reduced feature set, compared with that of the full data feature set. Both the models executed with the same training time.

The reduced model will not be chosen because of the training time factor. It would be that the statistical descriptions on the dataset might be the answer

The variations of the capital loss is less compared with other features. The education num is a highly factored feature because of it's high value of normalised mean.

Hours per week is a factor, with widely separated distribution values within all the quartiles.

> **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.