Machine Learning Engineer Nanodegree

Supervised Learning

Project: Finding Donors for *CharityML*

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

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

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

Getting Started

In this project, you will employ several supervised algorithms of your choice to accurately model individuals' income using data collected from the 1994 U.S. Census. You will then choose the best candidate algorithm from preliminary results and further optimize this algorithm to best model the data. Your goal with this implementation is to construct a model that accurately predicts whether an individual makes more than $50,000. This sort of task can arise in a non-profit setting, where organizations survive on donations. Understanding an individual's income can help a non-profit better understand how large of a donation to request, or whether or not they should reach out to begin with. While it can be difficult to determine an individual's general income bracket directly from public sources, we can (as we will see) infer this value from other publically available features.

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

Exploring the Data

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

In [1]:

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

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

# Pretty display for notebooks
%matplotlib inline

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

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

	age	workclass	education_level	education-num	marital-status	occupation	relationship	
0	39	State-gov	Bachelors	13.0	Never-married	Adm-clerical	Not-in-family	W

Implementation: Data Exploration

A cursory investigation of the dataset will determine how many individuals fit into either group, and will tell us about the percentage of these individuals making more than $50,000. In the code cell below, you will need to compute the following:

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

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

In [2]:

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

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

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

# 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: {:3.2f}%".format(gr
eater_percent))
```

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

Featureset Exploration

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

Preparing the Data

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

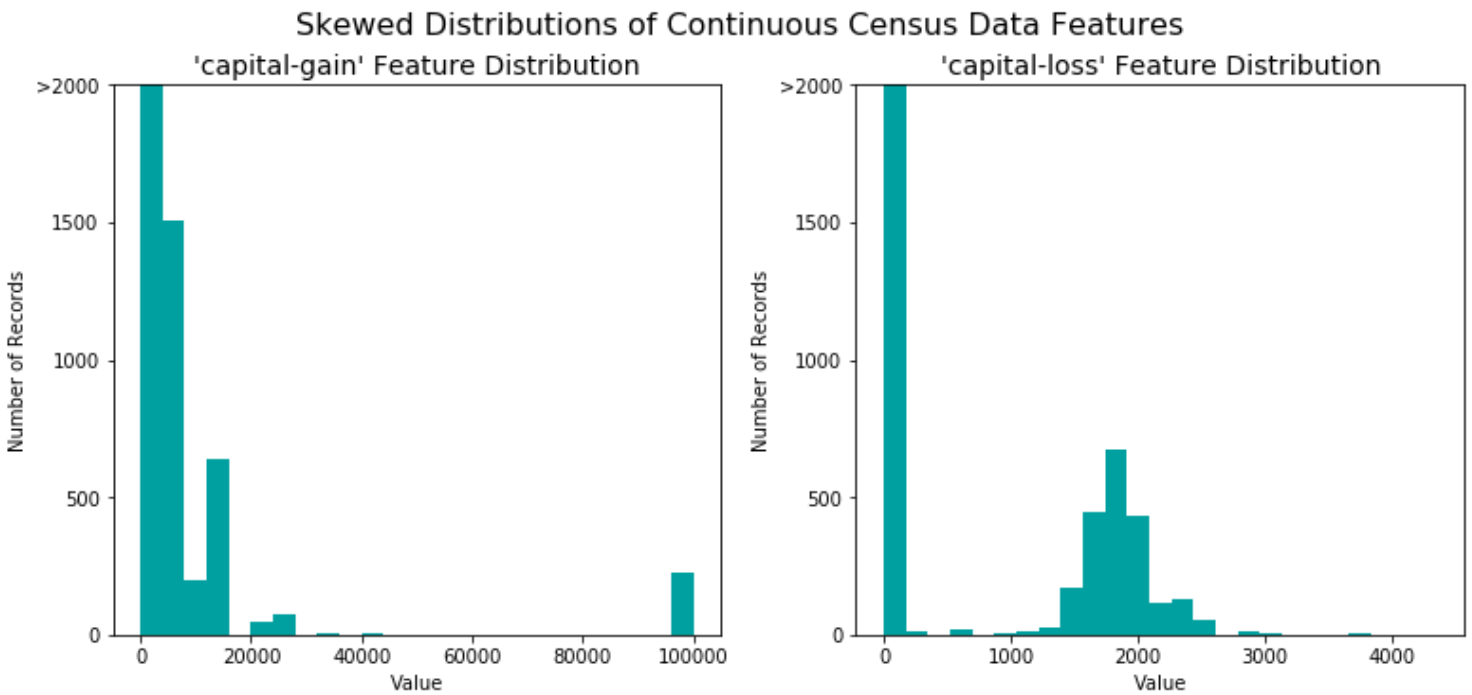
Transforming Skewed Continuous Features

A dataset may sometimes contain at least one feature whose values tend to lie near a single number, but will also have a non-trivial number of vastly larger or smaller values than that single number. Algorithms can be sensitive to such distributions of values and can underperform if the range is not properly normalized. With the census dataset two features fit this description: `'capital-gain'` and `'capital-loss'`.

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

In [3]:

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

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

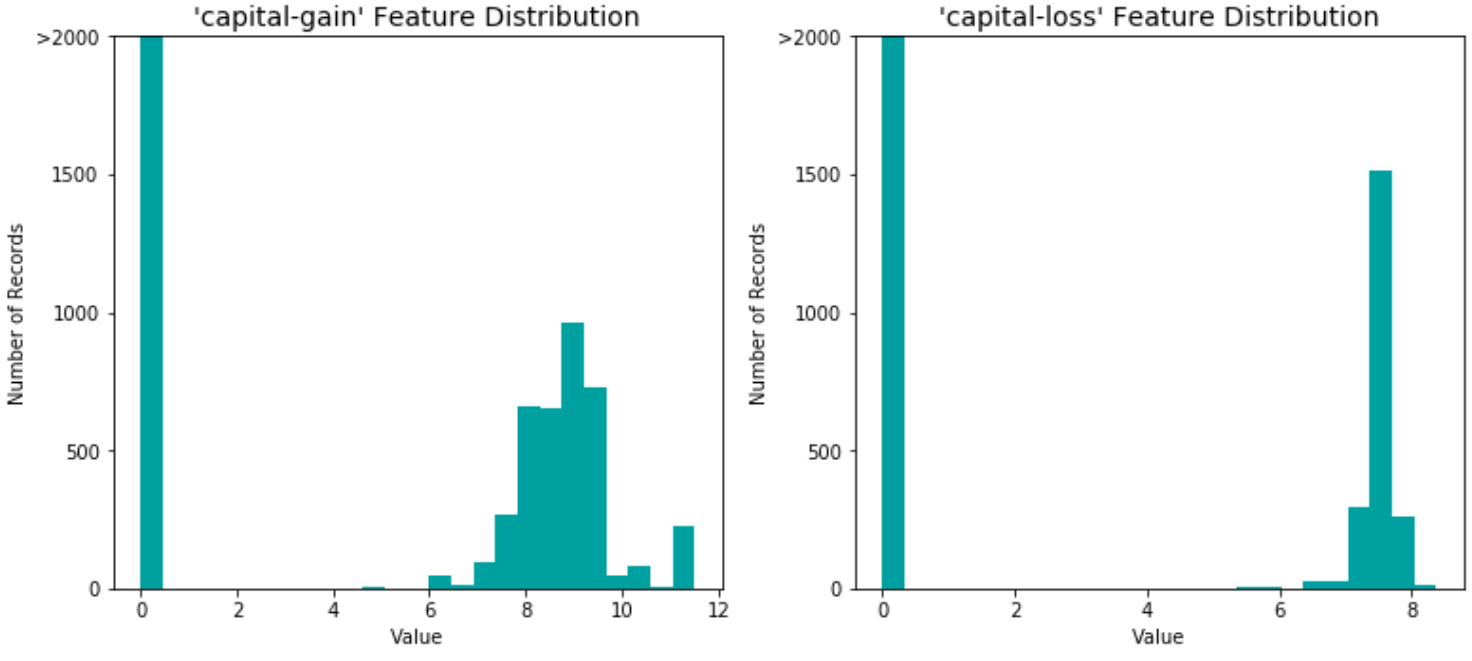


For highly-skewed feature distributions such as `'capital-gain'` and `'capital-loss'`, it is common practice to apply a logarithmic transformation (https://en.wikipedia.org/wiki/Data_transformation_(statistics)) on the data so that the very large and very small values do not negatively affect the performance 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_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)
```



Normalizing Numerical Features

In addition to performing transformations on features that are highly skewed, it is often good practice to perform some type of scaling on numerical features. Applying a scaling to the data does not change the shape of each feature's distribution (such as `'capital-gain'` or `'capital-loss'` above); however, normalization ensures that each feature is treated equally when applying supervised learners. Note that once scaling is applied, observing the data in its raw form will no longer have the same original meaning, as exampled below.

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

In [5]:

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

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

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

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

	age	workclass	education_level	education-num	marital-status	occupation	relatio
0	0.301370	State-gov	Bachelors	0.800000	Never-married	Adm-clerical	Not-in-family
1	0.452055	Self-emp-not-inc	Bachelors	0.800000	Married-civ-spouse	Exec-managerial	Husba
2	0.287671	Private	HS-grad	0.533333	Divorced	Handlers-cleaners	Not-in-family
3	0.493151	Private	11th	0.400000	Married-civ-spouse	Handlers-cleaners	Husba
4	0.150685	Private	Bachelors	0.800000	Married-civ-spouse	Prof-specialty	Wife

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

In [6]:

```
# TODO: One-hot encode the 'features_log_minmax_transform' data using pandas.g
et_dummies()
features_final = pd.DataFrame(data = features_log_minmax_transform)
features_final = pd.get_dummies(features_final)
# TODO: Encode the 'income_raw' data to numerical values
income = income_raw.replace({'<=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.
```

Shuffle and Split Data

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

Run the code cell below to perform this split.

In [8]:

```
# 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.
```

Evaluating Model Performance

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

Metrics and the Naive Predictor

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

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

In particular, when $\beta = 0.5$, more emphasis is placed on precision. This is called the **$F_{0.5}$ score** (or F-score for simplicity).

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

Note: Recap of accuracy, precision, recall

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

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

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

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

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

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

Question 1 - Naive Predictor Performace

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

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

HINT:

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

In [9]:

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

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

accuracy = TP/(TP+FP)
recall = TP/(TP +FN)
precision = TP/(TP+FP)
# TODO: Calculate F-score using the formula above for beta = 0.5 and correct v
alues for precision and recall.
fscore = (1+0.5**2)*((precision*recall)/(((0.5**2)*precision)+recall))

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

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

Supervised Learning Models

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

- Gaussian Naive Bayes (GaussianNB)
- Decision Trees
- Ensemble Methods (Bagging, AdaBoost, Random Forest, Gradient Boosting)
- K-Nearest Neighbors (KNeighbors)
- Stochastic Gradient Descent Classifier (SGDC)
- Support Vector Machines (SVM)
- Logistic Regression

Question 2 - Model Application

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

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

HINT:

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

Answer:

Although Ensemble methods may provide an improvement, I have chosen to use base models to compare. For example I will choose a Decision Tree model, rather than Random Forest or AdaBoost. This is in order to compare the models directly rather than poential advantages given by Ensemble Methods. The three models chosen to consider are Decision Tree, K-Nearest Neighbors and SVM.

Below is an overview of classifier comparisons in visual form with 3 different sets of data:

Decision Tree

- Describe one real-world application in industry where the model can be applied.
  - Decision Trees have been used as a tool for fault diagnosis. **Sugumaran and Ramachandran (2007) (http://adsabs.harvard.edu/abs/2007MSSP...21..930S)** created a decision tree model to identify the features which are significant for a bearing failure in rotary machines. The evaluations showed a very high level of classification accuracy and additionally indicated a removal of particular measurements which were time consuming, expensive but added little benefit to diagnosis.
- What are the strengths of the model;
  - Splits within a Decision Tree invariably chooses the features which are the most important.
  - Though we have scaled/normalised our numerical data, a descision tree does not necessarily require this to be done as part of the data preparation.
  - Decision making process is transparent in that you can see the decision process made (also known as a white box). In other models, this underlying decision process can be difficult to see, particularly in high-dimensional data (also known as a black box).
  - This transparency additionally allows you to show the process clearly to none data-scientists e.g. showing the model in working to management.
  - Model is relatively fast in both its fit and predict functions when compared to other models.
- What are the weaknesses of the model;
  - A small change in training data can involve a potentially major tree change.

    - Prone to overfitting to the training data
    - Problems associated with **NP-complete** where a greedy algorithm process may not provide the optimal outcome. This can be reduced to some extent with parameter changes, but is somthing that needs consideration
- What makes this model a good candidate for the problem, given what you know about the data?
    - A decision tree should be able to efficiently choose the most useful features which can be used differentiate between the classifier we require.
    - The model should be able to show which features are most effective in a clear manner that could be of use for further research.

K-Nearest Neighbors

- Describe one real-world application in industry where the model can be applied.
    - Recommender Systems often use this model. For example recommending similar music or movies based on previous viewing or listening activity
- What are the strengths of the model; when does it perform well?
    - No assumptions made about the data
    - Simple classifier that works well on basic recognition problems
- What are the weaknesses of the model; when does it perform poorly?
    - sensitive to outliers
    - Need to handle missing data and fill in
    - Sensitive to irrelevant attributes
    - Computationally expensive. Data is stored and tested at the testing/predict time not at predict (lazy learner).
    - As the number of data points and dimensions increases the system will become slower and slower (this could be a problem when predicting large numbers of potential donors)
- What makes this model a good candidate for the problem, given what you know about the data?
    - We may find classification based on the features provided is computationally simple and kNN model would provide the classification in a simple manner.
    - The training data will not significantly increase over time and so the predict times should be consistant each time.

SVM

- Describe one real-world application in industry where the model can be applied.
    - SVMs have been used frequently in Bioinformatics. **Haranath Varanasi (https://www.sciencedirect.com/science/article/pii/B9780124116436000387)** for example used SVM for predicting breast cancer diagnosis. Using 11 features of a potential tumor, the model was able to classify the results into two categories i.e. benign or malignant
- What are the strengths of the model;
    - high-dimensional spaces can be more easily seperated
    - May lead to better generalisation than other classifyers
- What are the weaknesses of the model;
    - Long training time on large data sets
- What makes this model a good candidate for the problem, given what you know about the data?

    - Though train times could be long, the SVM should be able to consider the volume of features to provide a potentially more accurate classifyer
    - Once training is complete, prediction times should be consistant and not overly slow.

References:

- SKlearn Classifier Comparison: http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html (http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html)
- data-flair, real-life applications of SVM 8th Aug 2017: https://data-flair.training/blogs/applications-of-svm/ (https://data-flair.training/blogs/applications-of-svm/)
- sciencedirect: Mutliple SVM reports and realwold examples: https://www.sciencedirect.com/topics/neuroscience/support-vector-machines (https://www.sciencedirect.com/topics/neuroscience/support-vector-machines)
- Haranath Varanasi, Predicting Breast Cancer Diagnosis Using Support Vector Machines,2015: https://www.sciencedirect.com/science/article/pii/B9780124116436000387 (https://www.sciencedirect.com/science/article/pii/B9780124116436000387)
- G. C. Cawley and N. L. C. Talbot, Over-fitting in model selection and subsequent selection bias in performance evaluation, Journal of Machine Learning Research, 2010. Research, vol. 11, pp. 2079-2107, July 2010. :http://jmlr.csail.mit.edu/papers/volume11/cawley10a/cawley10a.pdf (http://jmlr.csail.mit.edu/papers/volume11/cawley10a/cawley10a.pdf) http://www.svms.org/disadvantages.html (http://www.svms.org/disadvantages.html) Wikipedia: https://en.wikipedia.org/wiki/Support_vector_machine (https://en.wikipedia.org/wiki/Support_vector_machine) Support Vector Machines for Dummies: http://blog.aylien.com/support-vector-machines-for-dummies-a-simple/ (http://blog.aylien.com/support-vector-machines-for-dummies-a-simple/)
- Victor Lavrenko, kNN Pros and cons, Sept 15th 2015: : https://www.youtube.com/watch?v=aqou1ma8ZIs (https://www.youtube.com/watch?v=aqou1ma8ZIs)
- Kardi Teknomo: Strength and Weakness of K-Nearest Neighbor Algorithm: https://people.revoledu.com/kardi/tutorial/KNN/Strength%20and%20Weakness.htm (https://people.revoledu.com/kardi/tutorial/KNN/Strength%20and%20Weakness.htm)
- nickgillian.com wiki: http://www.nickgillian.com/wiki/pmwiki.php/GRT/KNN (http://www.nickgillian.com/wiki/pmwiki.php/GRT/KNN)
- Advantages of Decision Tree Analysis, N.Nayab 2/9/2011: https://www.brighthubpm.com/project-planning/106000-advantages-of-decision-tree-analysis/ (https://www.brighthubpm.com/project-planning/106000-advantages-of-decision-tree-analysis/)
- Disadvantaged to Using Decision Trees, N Nayab 2/9/2011:https://www.brighthubpm.com/project-planning/106005-disadvantages-to-using-decision-trees/ (https://www.brighthubpm.com/project-planning/106005-disadvantages-to-using-decision-trees/)
- Bala Deshpande, 4 Key Advantages of using decision trees for predictive analytics 2/9/2011: http://www.simafore.com/blog/bid/62333/4-key-advantages-of-using-decision-trees-for-predictive-analytics (http://www.simafore.com/blog/bid/62333/4-key-advantages-of-using-decision-trees-for-predictive-analytics)
- Decision Tree Applications for Data Modelling (Artificial Intelligence): http://what-when-how.com/artificial-intelligence/decision-tree-applications-for-data-modelling-artificial-intelligence/ (http://what-when-how.com/artificial-intelligence/decision-tree-applications-for-data-modelling-artificial-intelligence/)
- Sugumaran, Ramachandran , Feature selection using Decision Tree and classification through

Proximal Support Vector Machine for fault diagnostics of roller bearing, Feb 2007: http://adsabs.harvard.edu/abs/2007MSSP...21..930S (http://adsabs.harvard.edu/abs/2007MSSP...21..930S)

Implementation - Creating a Training and Predicting Pipeline

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

- Import `fbeta_score` and `accuracy_score` from `sklearn.metrics` (http://scikit-learn.org/stable/modules/classes.html#sklearn-metrics-metrics).
- Fit the learner to the sampled training data and record the training time.
- Perform predictions on the test data `X_test`, and also on the first 300 training points `X_train[:300]`.
    - Record the total prediction time.
- Calculate the accuracy score for both the training subset and testing set.
- Calculate the F-score for both the training subset and testing set.
    - Make sure that you set the `beta` parameter!

In [10]:

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

def train_predict(learner, sample_size, X_train, y_train, X_test, y_test):
    '''
    inputs:
       - learner: the learning algorithm to be trained and predicted on
       - sample_size: the size of samples (number) to be drawn from training s
et
       - 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_si
ze' using .fit(training_features[:], training_labels[:])
    start = time() # Get start time
    learner = learner.fit(X_train[:sample_size], y_train[:sample_size])
    end = time() # Get end time

    # TODO: Calculate the training time
    results['train_time'] = end-start
    # TODO: Get the predictions on the test set(X_test),
    #       then get predictions on the first 300 training samples(X_train) us
ing .predict()
    start = time() # Get start time
    predictions_test = learner.predict(X_test)
    predictions_train = learner.predict(X_train[:300])
    end = time() # Get end time
    # TODO: Calculate the total prediction time
    results['pred_time'] = end-start
    # TODO: Compute accuracy on the first 300 training samples which is y_trai
n[: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__, sampl
e_size))
    # Return the results
    return results
```

Implementation: Initial Model Evaluation

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

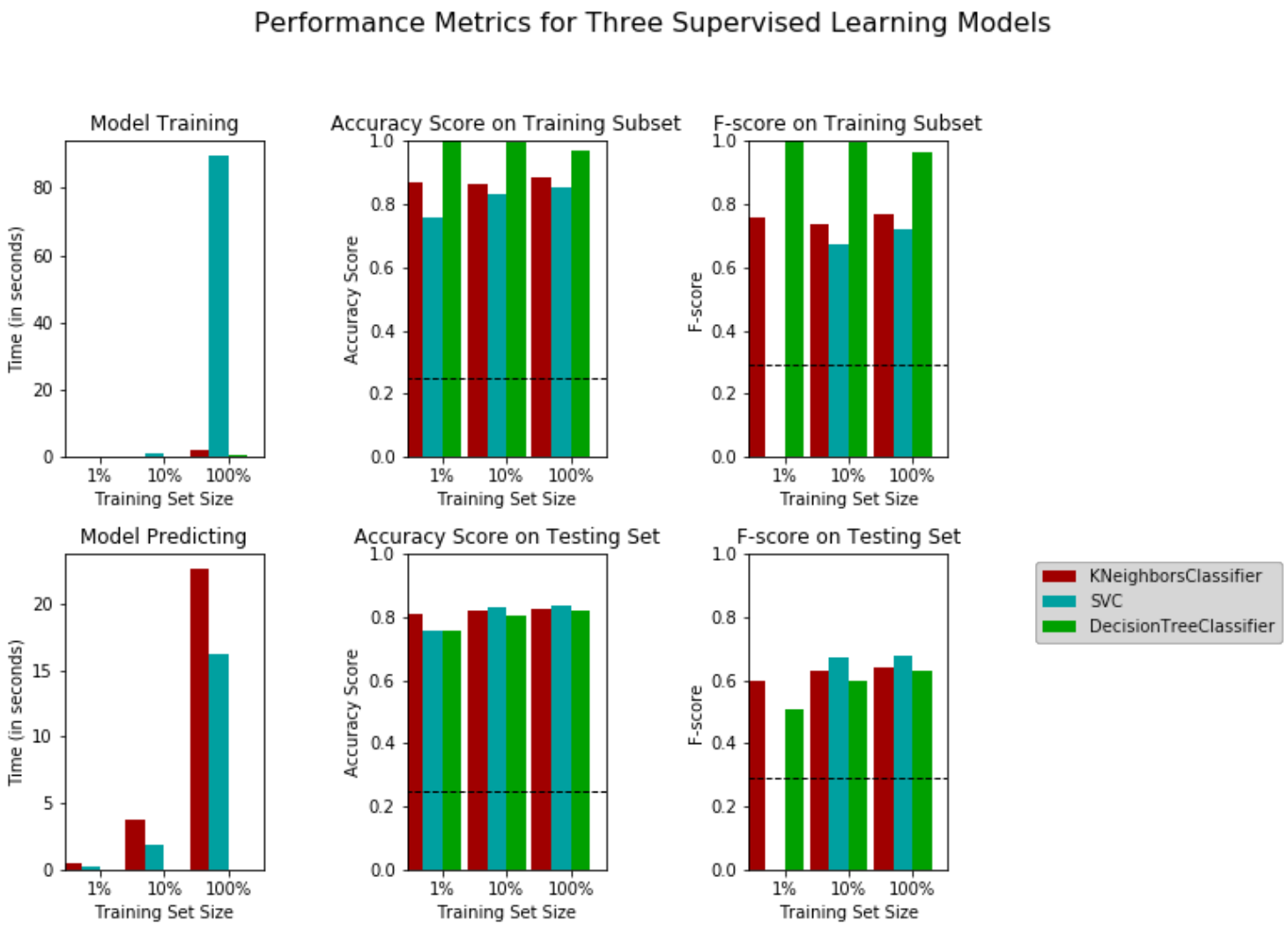
- Import the three supervised learning models you've discussed in the previous section.
- Initialize the three models and store them in `'clf_A'`, `'clf_B'`, and `'clf_C'`.
  - Use a `'random_state'` for each model you use, if provided.
  - **Note:** Use the default settings for each model — you will tune one specific model in a later section.
- Calculate the number of records equal to 1%, 10%, and 100% of the training data.
  - Store those values in `'samples_1'`, `'samples_10'`, and `'samples_100'` respectively.

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

In [11]:

```
# TODO: Import the three supervised learning models from sklearn
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

# TODO: Initialize the three models
clf_A = DecisionTreeClassifier(random_state=42)
clf_B = KNeighborsClassifier()
clf_C = SVC(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 value
s to be `int` and not `float`)
# HINT: samples_1 is 1% of samples_100 (ensure to set the count of the values
to be `int` and not `float`)
samples_100 = len(y_train)
samples_10 = int(len(y_train)*.10)
samples_1 = int(len(y_train)*.01)

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

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

```
DecisionTreeClassifier trained on 361 samples.
DecisionTreeClassifier trained on 3617 samples.
DecisionTreeClassifier trained on 36177 samples.
KNeighborsClassifier trained on 361 samples.
KNeighborsClassifier trained on 3617 samples.
KNeighborsClassifier trained on 36177 samples.

/usr/local/lib/python2.7/site-packages/sklearn/metrics/classificat
ion.py:1135: UndefinedMetricWarning: F-score is ill-defined and be
ing set to 0.0 due to no predicted samples.
  'precision', 'predicted', average, warn_for)

SVC trained on 361 samples.
SVC trained on 3617 samples.
SVC trained on 36177 samples.
```



Improving Results

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

Question 3 - Choosing the Best Model

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

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

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

Answer:

The f_beta score is set to a value of 0.5 in the `train_predict` method. As the sklearn documentation shows, a value less than 1 lends more weight to precision, whereas a beta greater than 1 favours recall. In our requirements we are more interested in precision of our model. That is, we want to be able to ensure that when we predict someone has an income greater than $50K we are correct as many times as possible, we want as little False Positives as possible since this affects the CharityML's costs unecessarily. As such the fbeta_score value will be more valuable to use than the accuracy score. Similarly the accuracy score for the test set for each model reported similar scores making comparissons there not worthwhile. As such the results to evaluate will be the fbeta scores and the times taken to fit and predict.

Below is a table of the results when testing 100% of the samples.

Model	Training fbeta score	Test fbeta score	Training Time	Predict Time
Decision Tree	0.964	0.629	0.366	0.009
KNeighbor	0.771	0.639	1.95	22.301
SVM	0.72	0.674	89.485	16.384

When looking at the training set. The decision tree shows to have the highest fbeta-score of 96%, though on the test set its fbeta score is significantly less than the training fbeta. This pattern was through the sample test ranges and this margin indicates that the decision tree is `overfitting the data`.

KNeighbors and SVM do show higher training fbeta scores as expected, though the margin compared to test set is much narrower. KNeighbors' Test score of 0.639 is similar to that of the decision tree's of 0.629. The SVM scores higher in the test set with a fbeta_score of 0.674

When looking at the training times, the SVM is significantly slower taking 89 seconds compared to the KNeighbor, though predict times for the two are similar (22.3 and 16.38). As the training data increases, the training time will increase. However; for this particualr use case, I believe the training time will be less of an issue. The charity would prefer to see the benefits associated with the increased fbeta score. Prediction times will not be a significant issue either, since the charity will not require extremely fast prediction times.

As such based on fbeta_score, considering the training and prediction times, and the benefits previously explained in Question 2; the **SVM would be the proposed model to suggest CharityML use.**

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 Support Vector Machine model is trained on data already available from census data to predict one of two classifications. The prediction being whether a person is likely to have an income over $50K or not.

It is taking into consideration many types, or 'features' of information from the census data; for example age, marital status, race, education, country and many more. Of this training data it knows whether or not that person earns more than 50K, hence training the model. The training attepmts to seperate the data into the two groups of income based on the features provided. Once the model is trained, it can then be given new data on which it is able to make new predictions of whether that person earns more than $50k.

Because there are many features in the census, knowing how and which to use to differentiate between two classifications successfully is difficult to acheive without some form of machine learning.

The image below shows an example how an SVM can take data from two features in a 2Dimensional plot and modify to distinguish between two classification types by raising to 3Dimensional to create a slice. The SVM can do this with a much higher number of features/dimensions in the same manner.

When testing the model, we provide a group of test cases which we already know the classification to. We can then compare the results the model return to the actual values to see how successful the model is.

Implementation: Model Tuning

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

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

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

In [12]:

```
# TODO: Import 'GridSearchCV', 'make_scorer', and any other necessary librarie
s
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer
# TODO: Initialize the classifier
clf = SVC( random_state=42)

# TODO: Create the parameters list you wish to tune, using a dictionary if nee
ded.
# HINT: parameters = {'parameter_1': [value1, value2], 'parameter_2': [value1,
value2]}]
parameters = {'C':[10, 20, 30], 'gamma':[0.01, 0.1, 1], 'probability':[True,Fa
lse]}

# 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 me
thod using GridSearchCV()
grid_obj = GridSearchCV(clf, parameters, scoring=scorer)

# TODO: Fit the grid search object to the training data and find the optimal p
arameters using fit()
grid_fit = grid_obj.fit(X_train, y_train)
# What are the best parameters?
print (grid_fit.best_params_)
# 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, p
redictions)))
print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, predictions
, beta = 0.5)))
print("\nOptimized Model\n------")
print("Final accuracy score on the testing data: {:.4f}".format(accuracy_score
(y_test, best_predictions)))
print("Final F-score on the testing data: {:.4f}".format(fbeta_score(y_test, b
est_predictions, beta = 0.5)))
```

```
{'C': 20, 'probability': True, 'gamma': 0.01}
Unoptimized model
------
Accuracy score on testing data: 0.8371
F-score on testing data: 0.6745

Optimized Model
------
Final accuracy score on the testing data: 0.8405
Final F-score on the testing data: 0.6825
```

Question 5 - Final Model Evaluation

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

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

Results:

Metric	Unoptimized Model	Optimized Model
Accuracy Score	0.8371	0.8405
F-score	0.6745	0.6825

Answer:

As the table shows, the optomized model gave a small improvement for both accuracy (a gain of 0.34%) and F-score (a gain of 0.8%). As such the hyper parameter updates found during the GridSearchCV were of some benefit but slight.

The naive predictor in Question 1 assumes everyone earns more than 50K. As previously shown this provided an Accuracy score of 0.2478 and F-score or 0.2917 The naive predictor is the equivalent of CharityML selecting anyone in the hope they are someone who earns more than $50K and is therefore more likely someone who will provide a donation.

The optomized SVM's F-Score of 0.6825 is much higher than that of the naive predictor's 0.2917. The result shows a greater likelyhood (over twice as likely) of selecting someone who earns more than $50k compared to the naive model. Using the SVM model should help increase the number of donations given to CharityML.

Feature Importance

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

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

Question 6 - Feature Relevance Observation

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

Answer:

1. Education
    - I believe if someone has completed a certain level of education experience, whether completing year 9 or higher, they are more likely to have a greater income than someone who finished school at an earlier stage or who dropped out completely.
2. Age
    - Someone early in their career is less likely to be earning a high level of income. It is more likely someone will earn more as they progress through their career. As such age should be a good differentiator.
3. Capital-gains
    - Someone who has received capital gains is more likely to be earning a higher income. The reason being that in order to receive capital gains, there needs to be some level of initial capital investment. Somone on a lower income is less likely to have the available funds to provide this investment.
4. education-num
    - Alongside education, I believe that someone is more likely to earn more if they have completed their education in one continuous block. If someone has gone back to finish school or complate a degree later in life it could be that this gap could prevent potential earnings.
5. Hours-per-week
    - Irrespective of education, age, job type etc. if someone is not working many hours then their income is likely to be affected. Therefore a certain level of working hours again is likely to be a good differentiator after other considerations are used.

Implementation - Extracting Feature Importance

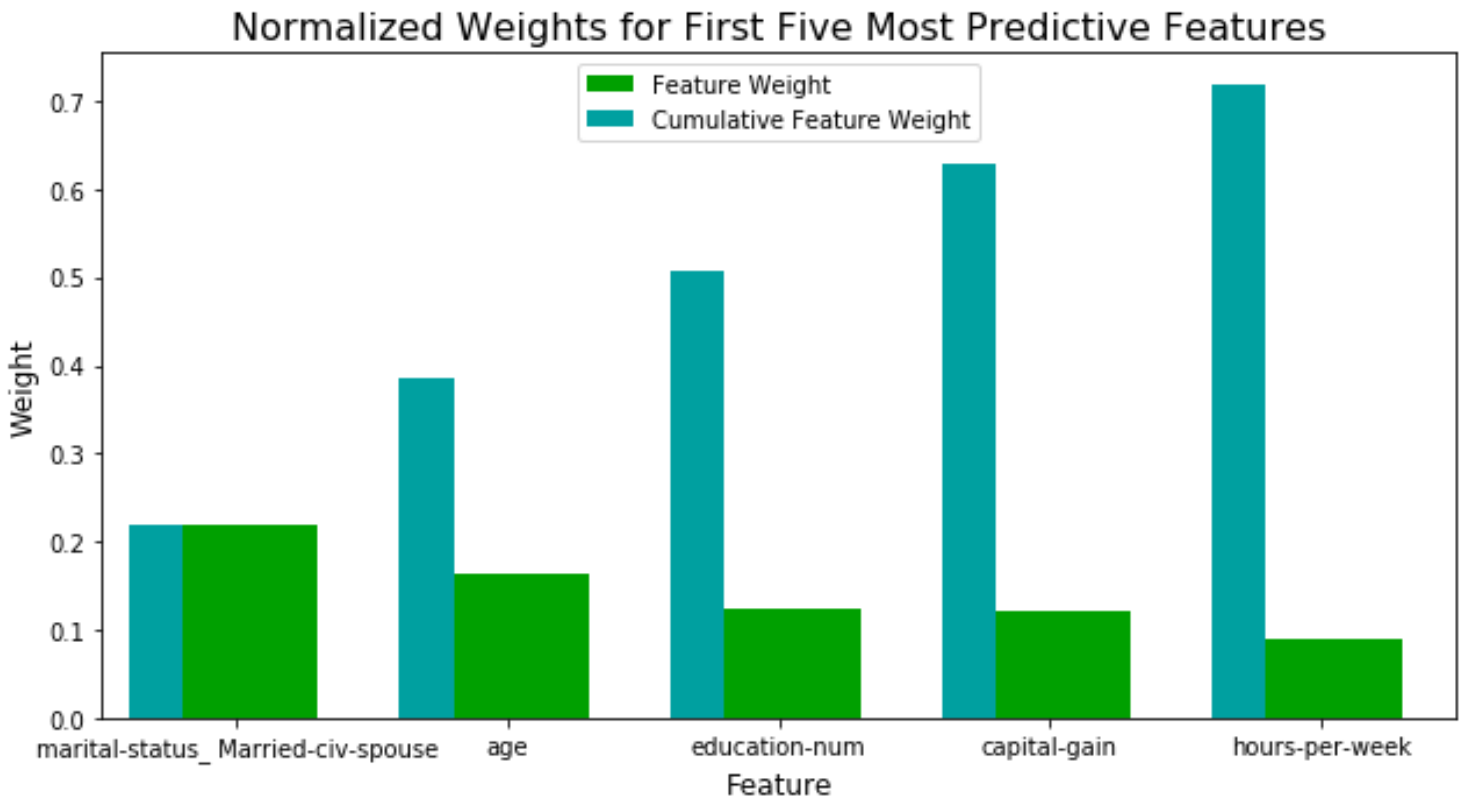
Choose a `scikit-learn` supervised learning algorithm that has a `feature_importance_` attribute availble for it. This attribute is a function that ranks the importance of each feature when making predictions based on the chosen algorithm.

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 [14]:

```
# TODO: Import a supervised learning model that has 'feature_importances_'
#already have DecisionTreeClassifier imported

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

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

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



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:

Of the features shown in the decision tree reported as most important, I selected four out of five namely age, education-num, capital-gain, and hours-per-week. Of those, most which were selected were in the same order of importance other than education-num which I priorotised after capital-gains.

I did not consider marital_status in my feature list, and specifically whether someone is married as shown in the chart above. This feature initially does not seem related, but with hindsight it does make sense. Its likely that someone who is recently married, or has been married for some time are in a particular financial position. They are probably more likely to be looking at (or already have) other large purchase items such as houses and therefore an income over $50k. I had not considered this and chose features I thought were more directly related to income. In doing so I had selected education as my primary feature of importance as the item related to income.

I think the choices I selected in question 6 were comparable, but its certainly interesting to see features which I would not have naturally selected.

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 [15]:

```
# 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_pr
edictions)))
print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, best_predic
tions, beta = 0.5)))
print("\nFinal Model trained on reduced data\n------")
print("Accuracy on testing data: {:.4f}".format(accuracy_score(y_test, reduced
_predictions)))
print("F-score on testing data: {:.4f}".format(fbeta_score(y_test, reduced_pre
dictions, beta = 0.5)))
```

```
Final Model trained on full data
------
Accuracy on testing data: 0.8405
F-score on testing data: 0.6825

Final Model trained on reduced data
------
Accuracy on testing data: 0.8289
F-score on testing data: 0.6538
```

Question 8 - Effects of Feature Selection

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

Answer:

Metric	Full Data	Reduced Data
Accuracy Score	0.8405	0.8289
F-score	0.6825	0.6538

The difference in accuracy between the full data and the data with the five features is very similar with a difference of 1.16% The difference in F-score between full data and reduced data is 2.87% to 0.6538 which is reduced but still much better than the naive predictor's results of 0.2917

As we discussed previously I believe that the training time is not a factor that needs to be considered for CharityML. However; if timing was an issue, by reducing the number of features from 103 down to these 5, the training time is significantly reduced. For the level of loss in precision, this could be an acceptable compromise.

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.