Adult Dataset: A Machine Learning Project

For this project I will be using the “Adult” dataset hosted on UCI’s Machine Learning Repository. The dependent variable that in all cases we will be trying to predict is whether or not an “individual” has an income greater than $50,000 a year.

Here is the set of variables contained in the data.

age – The age of the individual

type_employer – The type of employer the individual has. Whether they are government, military, private, and so on.

fnlwgt – The # of people the census takers believe that observation represents.

education – The highest level of education achieved for that individual

education_num – Highest level of education in numerical form

marital – Marital status of the individual

occupation – The occupation of the individual

relationship – A bit more difficult to explain. Contains family relationship values like husband, father, and so on, but only contains one per observation. race – descriptions of the individuals race. Black, White, Eskimo, and so on

sex – Biological Sex

capital_gain – Capital gains recorded

capital_loss – Capital Losses recorded

hr_per_week – Hours worked per week

country – Country of origin for person

income – Boolean Variable. Whether or not the person makes more than $50,000 per annum income.

In [6]:
```
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import pandas as pd

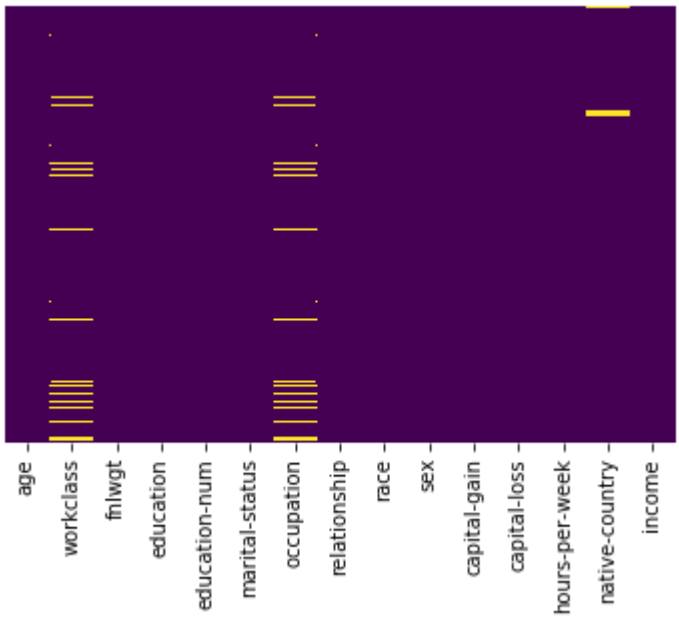
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

df = pd.read_csv('adult.csv')
```

Let's look for missing values

In [2]:
```
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

Out[2]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7fa39feed0>
```



It seems that it is possible to drop all the missing data and still have more than 30K of rows. Later, we could do a more accurate prediction with Logistic Regression and other models

In [3]:
```
df=df.dropna()
```

Converting Categorical Features To Dummy Variables

In [4]:
```
df2 = df
Education = pd.get_dummies(df2['education'],drop_first=True)
Workclass = pd.get_dummies(df2['workclass'],drop_first=True)
Maritalstatus = pd.get_dummies(df2['marital-status'],drop_first=True)
Occupation = pd.get_dummies(df2['occupation'],drop_first=True)
Relationship = pd.get_dummies(df2['relationship'],drop_first=True)
Race = pd.get_dummies(df2['race'],drop_first=True)
Sex = pd.get_dummies(df2['sex'],drop_first=True)
Nativecountry = pd.get_dummies(df2['native-country'],drop_first=True)
Income = pd.get_dummies(df2['income'],drop_first=True)
```

In [5]:
```
df2.drop(['sex','education','income','native-country','race','relationship','o
ccupation','marital-status','workclass'],axis=1,inplace=True)
```

In [6]:
```
df2 = pd.concat([df2,Education,Workclass,Maritalstatus,Occupation,Relationship
,Race,Sex,Nativecountry,Income],axis=1)
```

Building a Logistic Regression model

In [7]:
```
X = df2.iloc[:,:-1]
y = df2.iloc[:,-1:]
```

In [8]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.3)
```

In [9]:
```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

```
/home/y/anaconda2/lib/python2.7/site-packages/sklearn/utils/validation.py:57
8: DataConversionWarning: A column-vector y was passed when a 1d array was ex
pected. Please change the shape of y to (n_samples, ), for example using rave
l().
  y = column_or_1d(y, warn=True)
```

Out[9]:
```
LogisticRegression(C=1.0, 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)
```

In [10]:
```
predictions = logmodel.predict(X_test)
```

In [11]:
```
print(classification_report(y_test,predictions))
print(confusion_matrix(y_test,predictions))
```

```
             precision    recall  f1-score   support

          0       0.80      0.97      0.88      6796
          1       0.74      0.27      0.39      2253

avg / total       0.78      0.79      0.75      9049

[[6580  216]
 [1653  600]]
```

Precision: TP/(TP+FP) = 600/(600+216) = 0.74

Recall: TP/(TP+FN) = 600/(600+1653) = 0.27

While the prediction of the total model is 78%, there are around 39% chance for the model to predict a True Positive and 88% to predict a True Negative

This results are not great, perhaps a different model will do better. Let's try the Random Forest Classifier

Building a Random Forest Model

In [12]:
```
rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(X_train, y_train)
```

```
/home/y/anaconda2/lib/python2.7/site-packages/ipykernel_launcher.py:2: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples,), for example using ravel().
```

Out[12]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [13]:
```
rfc_pred = rfc.predict(X_test)
```

In [14]:
```
print(classification_report(y_test,rfc_pred))
print(confusion_matrix(y_test,rfc_pred))
```

```
             precision    recall  f1-score   support

          0       0.88      0.92      0.90      6796
          1       0.72      0.62      0.67      2253

avg / total       0.84      0.85      0.84      9049

[[6264  532]
 [ 851 1402]]
```

Precision: TP/(TP+FP) = 1402/(1402+532) = 0.72

Recall: TP/(TP+FN) = 1402/(1402+851) = 0.62

While the prediction of the total model is 84%, there are around 67% chance for the model to predict a True Positive and 90% to predict a True Negative. Better results!

Now, let's try this models again but now we will keep all the dummy variables and lose all the continuous variables

Building a Logistic Regression Model With Only Dummy Variables

In [15]:
```
df2 = pd.concat([Education,Workclass,Maritalstatus,Occupation,Relationship,Race,Sex,Nativecountry,Income],axis=1)
```

In [16]:
```
X = df2.iloc[:,:-1]
y = df2.iloc[:,-1:]
```

In [17]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,test_size=0.3)
```

In [18]:
```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

```
/home/y/anaconda2/lib/python2.7/site-packages/sklearn/utils/validation.py:578: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[18]:
```
LogisticRegression(C=1.0, 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)
```

In [19]:
```
predictions = logmodel.predict(X_test)
```

In [20]:
```
print(classification_report(y_test,predictions))
print(confusion_matrix(y_test,predictions))
```

```
             precision    recall  f1-score   support

          0       0.86      0.92      0.88      6772
          1       0.68      0.54      0.60      2277

avg / total       0.81      0.82      0.81      9049

[[6198  574]
 [1049 1228]]
```

Precision: TP/(TP+FP) = 1228/(1228+574) = 0.68

Recall: TP/(TP+FN) = 1228/(1228+1049) = 0.54

While the prediction of the total model is 81%, there are around 60% chance for the model to predict a True Positive and 88% to predict a True Negative. A better result comparing to the dummy variables and the continuous variables analysis

Building a Random Forest Model With Only Dummy Variables

In [21]:
```
rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(X_train, y_train)
```

```
/home/y/anaconda2/lib/python2.7/site-packages/ipykernel_launcher.py:2: DataCo
nversionWarning: A column-vector y was passed when a 1d array was expected. P
lease change the shape of y to (n_samples,), for example using ravel().
```

Out[21]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [22]:
```
rfc_pred = rfc.predict(X_test)
```

In [23]:
```
print(classification_report(y_test,rfc_pred))
print(confusion_matrix(y_test,rfc_pred))
```

```
             precision    recall  f1-score   support

          0       0.85      0.91      0.88      6772
          1       0.67      0.53      0.59      2277

avg / total       0.81      0.82      0.81      9049

[[6182  590]
 [1072 1205]]
```

Precision: TP/(TP+FP) = 1205/(1205+590) = 0.67

Recall: TP/(TP+FN) = 1205/(1205+1072) = 0.53

While the prediction of the total model is 81%, there are around 59% chance for the model to predict a True Positive and 88% to predict a True Negative. A lower result comparing to the dummy variables and the continuous variables analysis. But there is no different between this model and the logistic regression model

The Logistic Regression model showed low results while analyzing with dummy and continuous variables and better results while analyzing with only dummy variables.

The Random Forest model showed the best results while analyzing with dummy and continuous variables and the same results as the Logistic Regression model while analyzing solely with dummy variables.

This shows that while the Random Forest model is more flexible and can operate better with different kinds of variables, the Logistic Regression model operates better with just dummy variables.