In this example, we continue to drill a bit futher into the use of scikit-learn for classification, as well as the use of cross-validation for evaluation model performance.

In [1]:
```
import numpy as np
import pandas as pd
```

In [2]:
```
vstable = pd.read_csv("http://facweb.cs.depaul.edu/mobasher/classes/csc478/c

vstable.shape
```

Out[2]:
```
(50, 7)
```

In [3]:
```
vstable.head()
```

Out[3]:

	Gender	Income	Age	Rentals	Avg Per Visit	Genre	Incidentals
Cust ID							
1	M	45000	25	32	2.5	Action	Yes
2	F	54000	33	12	3.4	Drama	No
3	F	32000	20	42	1.6	Comedy	No
4	F	59000	70	16	4.2	Drama	Yes
5	M	37000	35	25	3.2	Action	Yes

Let's separate the target attribute and the attributes used for model training

In [4]:
```
vs_records = vstable[['Gender','Income','Age','Rentals','Avg Per Visit','Ger
vs_records.head()
```

Out[4]:

	Gender	Income	Age	Rentals	Avg Per Visit	Genre
Cust ID						
1	M	45000	25	32	2.5	Action
2	F	54000	33	12	3.4	Drama
3	F	32000	20	42	1.6	Comedy
4	F	59000	70	16	4.2	Drama
5	M	37000	35	25	3.2	Action

In [5]:
```
vs_target = vstable.Incidentals
vs_target.head()
```

Out[5]:
```
Cust ID
1    Yes
2     No
3     No
4    Yes
5    Yes
Name: Incidentals, dtype: object
```

As before, we use Pandas "get_dummies" function to create dummy variables.

In [6]:
```
vs_matrix = pd.get_dummies(vs_records[['Gender','Income','Age','Rentals','Av
vs_matrix.head(10)
```

Out[6]:

Cust ID	Income	Age	Rentals	Avg Per Visit	Gender_F	Gender_M	Genre_Action	Genre_Comedy	Genr
1	45000	25	32	2.5	0	1	1	0	
2	54000	33	12	3.4	1	0	0	0	
3	32000	20	42	1.6	1	0	0	1	
4	59000	70	16	4.2	1	0	0	0	
5	37000	35	25	3.2	0	1	1	0	
6	18000	20	29	1.7	0	1	1	0	
7	29000	45	19	3.8	1	0	0	0	
8	74000	25	31	2.4	0	1	1	0	
9	38000	21	18	2.1	0	1	0	1	
10	65000	40	21	3.3	1	0	0	0	

Next, we divide the data into randomized training and test partitions (note that the same split should also be perfromed on the target attribute). The easiest way to do this is to use the "train_test_split" module of "sklearn.cross_validation".

In [9]:

```
from sklearn.model_selection import train_test_split
vs_train, vs_test, vs_target_train, vs_target_test = train_test_split(vs_mat

print(vs_test.shape)
vs_test[0:5]
```

```
(10, 9)
```

Out[9]:

Cust ID	Income	Age	Rentals	Avg Per Visit	Gender_F	Gender_M	Genre_Action	Genre_Comedy	Genr
6	18000	20	29	1.7	0	1	1	0	
28	57000	52	22	4.1	0	1	0	1	
38	41000	38	20	3.3	0	1	0	0	
16	17000	19	26	2.2	0	1	1	0	
41	50000	33	17	1.4	1	0	0	0	

In [10]:

```
print(vs_train.shape)
vs_train[0:5]
```

```
(40, 9)
```

Out[10]:

Cust ID	Income	Age	Rentals	Avg Per Visit	Gender_F	Gender_M	Genre_Action	Genre_Comedy	Genr
30	41000	25	17	1.4	0	1	1	0	
35	74000	29	43	4.6	0	1	1	0	
18	6000	16	39	1.8	1	0	1	0	
40	17000	19	32	1.8	0	1	1	0	
2	54000	33	12	3.4	1	0	0	0	

Let's try KNN Classifier - Note that in this example we did not normalize the data.

In [11]:

```
from sklearn import neighbors, tree, naive_bayes
```

First, we'll use KNN classifer. You can vary K and monitor the accuracy metrics (see below) to find the best value.

In [12]:
```
n_neighbors = 5

knnclf = neighbors.KNeighborsClassifier(n_neighbors, weights='distance')
knnclf.fit(vs_train, vs_target_train)
```

Out[12]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='distance')
```

Next, we call the predict function on the test intances to produce the predicted classes.

In [13]:
```
knnpreds_test = knnclf.predict(vs_test)
```

In [15]:
```
print(knnpreds_test)
```

```
['No' 'Yes' 'Yes' 'No' 'No' 'Yes' 'Yes' 'Yes' 'No' 'No']
```

In [16]:
```
from sklearn.metrics import classification_report
```

In [17]:
```
print(classification_report(vs_target_test, knnpreds_test))
```

```
              precision    recall  f1-score   support

          No       0.40      0.50      0.44         4
         Yes       0.60      0.50      0.55         6

    accuracy                           0.50        10
   macro avg       0.50      0.50      0.49        10
weighted avg       0.52      0.50      0.51        10
```

In [19]:
```
print(knnclf.score(vs_test, vs_target_test))
```

```
0.5
```

In [20]:
```
print(knnclf.score(vs_train, vs_target_train))
```

```
1.0
```

You may notice that accuracy on test data is much lower than in part 1 of this example (previous notebook) when the data was normalized and rescaled. This may indicate that normalization in KNN is very important to improve performance and to avoid overfitting.

Next, let's use a decision tree classifier:

In [21]:
```
treeclf = tree.DecisionTreeClassifier(criterion='entropy', min_samples_split
treeclf = treeclf.fit(vs_train, vs_target_train)
```

In [22]:
```
print(treeclf.score(vs_test, vs_target_test))
```

```
0.6
```

In [23]:
```
print(treeclf.score(vs_train, vs_target_train))
```

```
0.95
```

Now, let's try Gaussian and Multinomial Naive Bayes classifiers:

In [24]:
```
nbclf = naive_bayes.GaussianNB()
nbclf = nbclf.fit(vs_train, vs_target_train)
print("Score on Training: ", nbclf.score(vs_train, vs_target_train))
print("Score on Test: ", nbclf.score(vs_test, vs_target_test))
```

```
Score on Training:  0.675
Score on Test:  0.8
```

In [25]:
```
nbmclf = naive_bayes.MultinomialNB()
nbmclf = nbclf.fit(vs_train, vs_target_train)
print("Score on Training: ", nbmclf.score(vs_train, vs_target_train))
print("Score on Test: ", nbmclf.score(vs_test, vs_target_test))
```

```
Score on Training:  0.675
Score on Test:  0.8
```

Finally, let's try linear discriminant analysis:

In [27]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

ldclf = LinearDiscriminantAnalysis()
ldclf = ldclf.fit(vs_train, vs_target_train)
print("Score on Training: ", ldclf.score(vs_train, vs_target_train))
print("Score on Test: ", ldclf.score(vs_test, vs_target_test))
```

```
Score on Training:  0.725
Score on Test:  0.9
```

```
C:\Users\bmobashe\AppData\Local\Continuum\anaconda3\lib\site-packages\skle
  warnings.warn("Variables are collinear.")
```

Let's explore various decision tree parameters and also the use of cross-validation for evaluation:

In [29]:
```
import graphviz
from sklearn.tree import export_graphviz
from sklearn.model_selection import cross_val_score
```

In [25]:
```
treeclf = tree.DecisionTreeClassifier(criterion='entropy')
```

In [30]:
```
cv_scores = cross_val_score(treeclf, vs_matrix, vs_target, cv=5)
cv_scores
```

Out[30]:
```
array([0.45454545, 0.3       , 0.8       , 0.7       , 0.77777778])
```

In [31]:
```
print("Overall Accuracy on X-Val: %0.2f (+/- %0.2f)" % (cv_scores.mean(), cv
```

```
Overall Accuracy on X-Val: 0.61 (+/- 0.39)
```

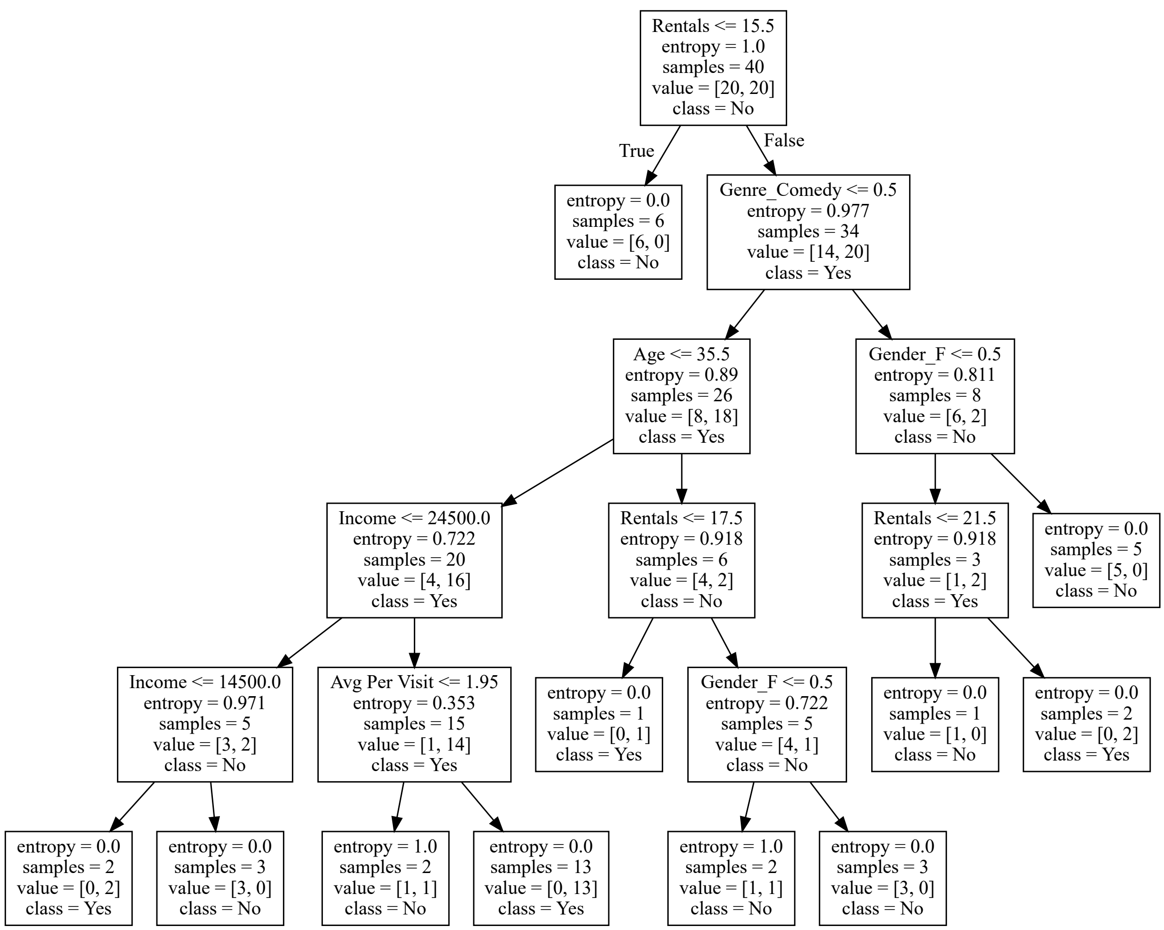
In [32]:
```
treeclf = treeclf.fit(vs_train, vs_target_train)
print("Accuracy on Training: ",  treeclf.score(vs_train, vs_target_train))
```

```
Accuracy on Training:  0.95
```

In [33]:
```
export_graphviz(treeclf,out_file='tree.dot', feature_names=vs_train.columns,

with open("tree.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

Out[33]:



We can obtain summary results on how informative are each of the features in the data:

In [34]:

```
print("Feature Importances:\n{}".format(treeclf.feature_importances_))
```

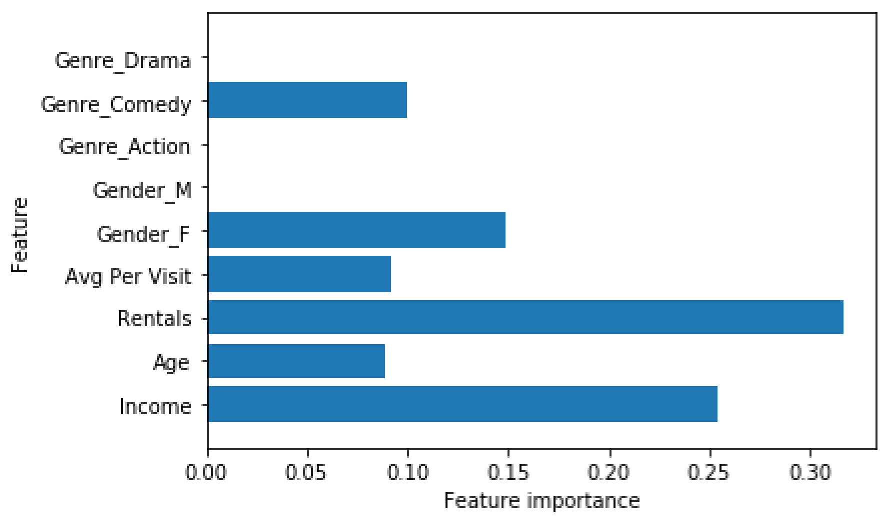
```
Feature Importances:
[0.25383811 0.08901238 0.31730045 0.0916775  0.14847161 0.
 0.         0.09969995 0.        ]
```

In [35]:

```
import pylab as plt
%matplotlib inline

def plot_feature_importances(model, n_features, feature_names):
    plt.barh(range(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), feature_names)
    plt.xlabel("Feature importance")
    plt.ylabel("Feature")
    plt.ylim(-1, n_features)

plot_feature_importances(treeclf, len(vs_matrix.columns), vs_matrix.columns)
```



The above evaluation results indicate overfitting. Pruning the tree may help in reducing overfitting.

In [39]:

```
treeclf = tree.DecisionTreeClassifier(criterion='entropy', min_samples_leaf=
cv_scores = cross_val_score(treeclf, vs_matrix, vs_target, cv=5)
print(cv_scores)
print("Overall Accuracy on X-Val: %0.2f (+/- %0.2f)" % (cv_scores.mean(), cv

treeclf = treeclf.fit(vs_train, vs_target_train)
print("Accuracy on Training: ",  treeclf.score(vs_train, vs_target_train))
```

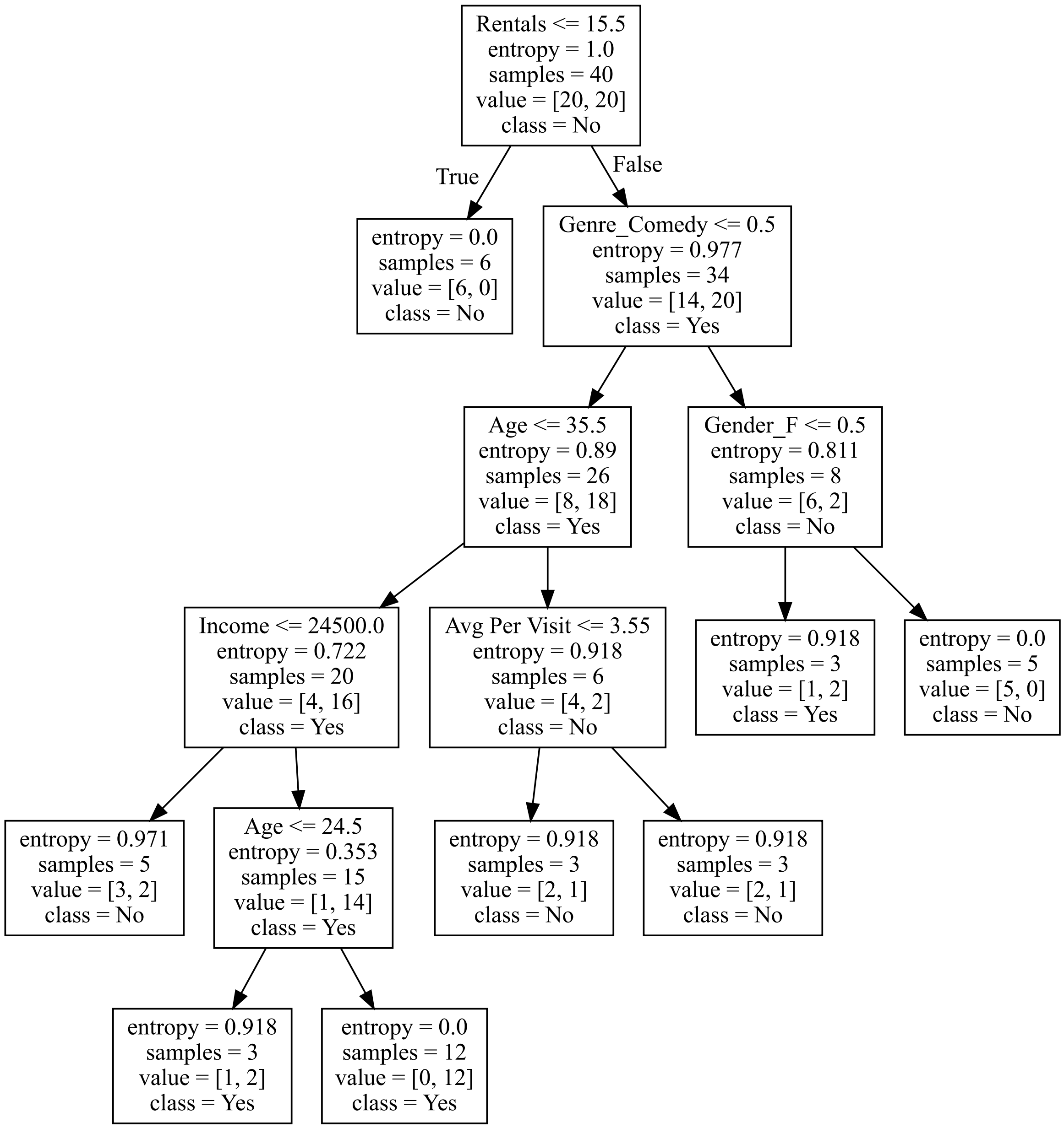
```
[0.72727273 0.4        0.7        0.7        0.77777778]
Overall Accuracy on X-Val: 0.66 (+/- 0.27)
Accuracy on Training:  0.85
```

In [40]:

```
export_graphviz(treeclf,out_file='tree.dot', feature_names=vs_train.columns,

with open("tree.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

Out[40]:



In [35]:

```
treeclf = tree.DecisionTreeClassifier(criterion='entropy', max_depth=4)
cv_scores = cross_validation.cross_val_score(treeclf, vs_matrix, vs_target,
print cv_scores
print("Overall Accuracy on X-Val: %0.2f (+/- %0.2f)" % (cv_scores.mean(), cv

treeclf = treeclf.fit(vs_train, vs_target_train)
print "Accuracy on Training: ",  treeclf.score(vs_train, vs_target_train)
```

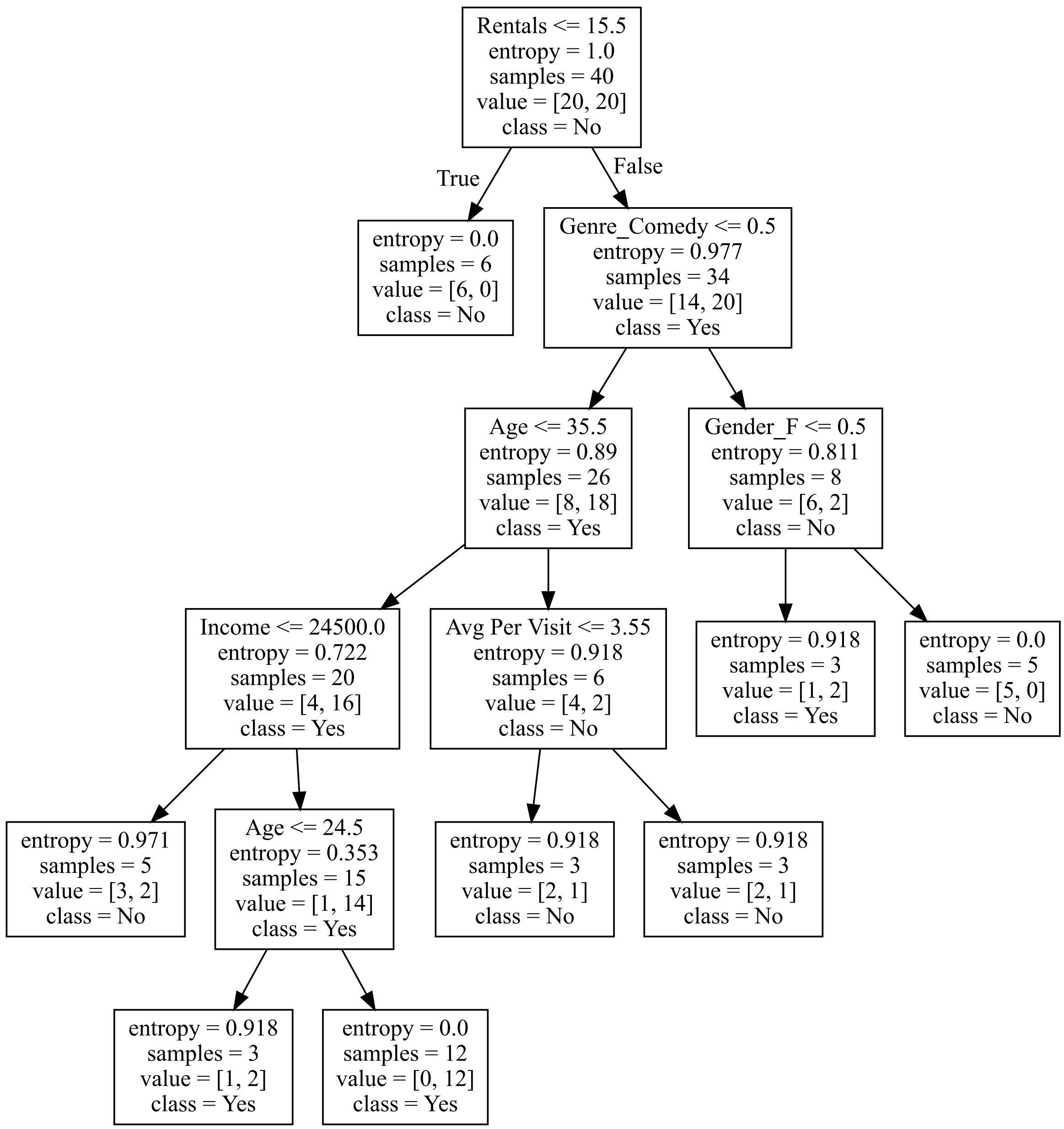
```
[ 0.45454545  0.3         0.8         0.7         0.77777778]
Overall Accuracy on X-Val: 0.61 (+/- 0.39)
Accuracy on Training:  0.9
```

In [41]:

```
export_graphviz(treeclf,out_file='tree.dot', feature_names=vs_train.columns,

with open("tree.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

Out[41]:



In [43]:

```
treeclf = tree.DecisionTreeClassifier(criterion='gini', min_samples_leaf=3,
cv_scores = cross_val_score(treeclf, vs_matrix, vs_target, cv=5)
print(cv_scores)
print("Overall Accuracy on X-Val: %0.2f (+/- %0.2f)" % (cv_scores.mean(), cv

treeclf = treeclf.fit(vs_train, vs_target_train)
print("Accuracy on Training: ",  treeclf.score(vs_train, vs_target_train))
```

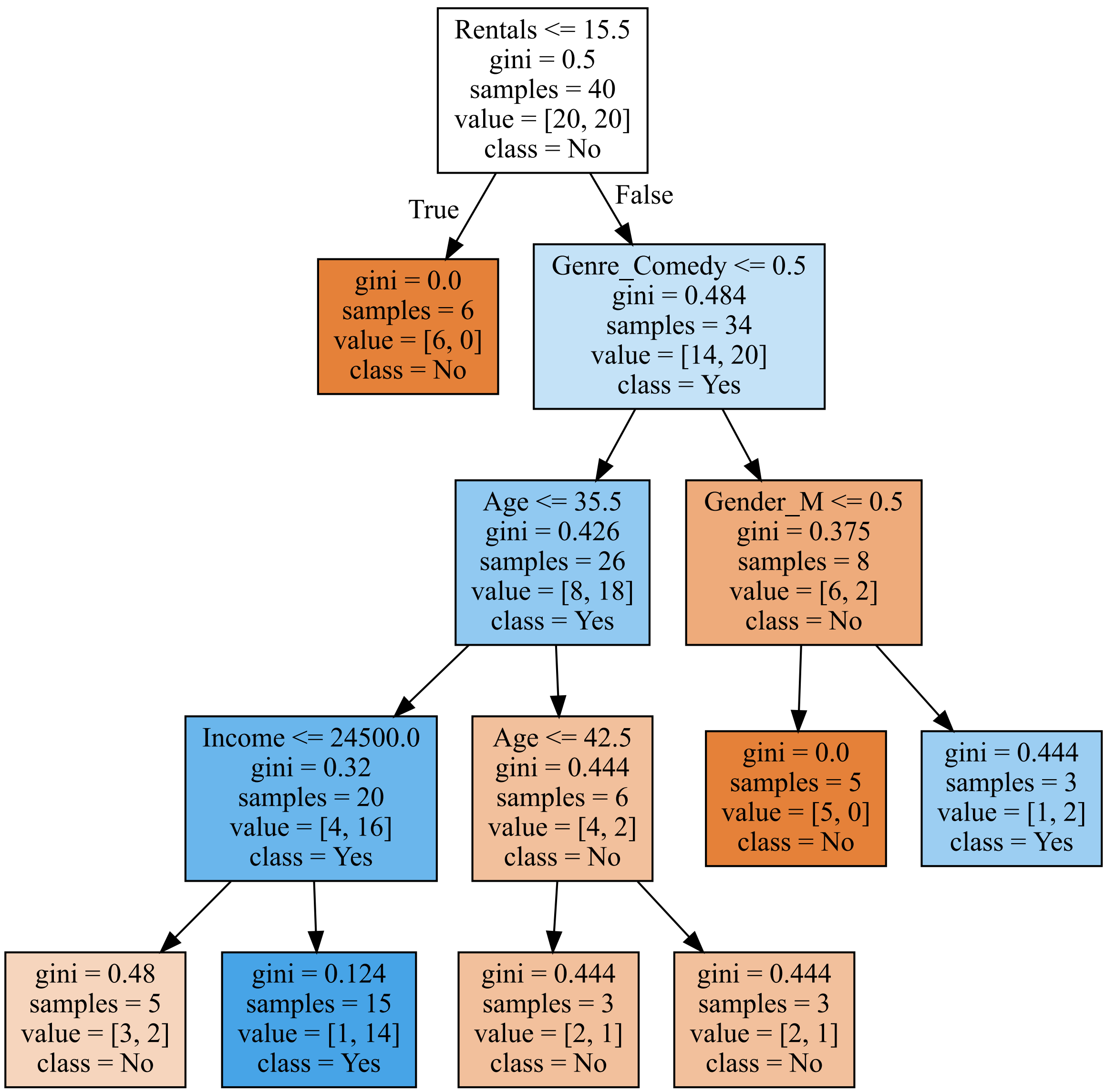
```
[0.81818182 0.4        0.8        0.9        0.77777778]
Overall Accuracy on X-Val: 0.74 (+/- 0.35)
Accuracy on Training:  0.85
```

In [44]:

```
export_graphviz(treeclf,out_file='tree.dot', feature_names=vs_train.columns,

with open("tree.dot") as f:
    dot_graph = f.read()
graphviz.Source(dot_graph)
```

Out[44]:



In [ ]:

```
```