In our first experiment we will use optimized test/train split ratio, which is selected by comparing 50 different accuracies. In second experiment we will hold this ratio and change depth. Depth is also calculated to give the maximum accuracy in each experiment.

In our third experiment we will evaluate the results with same depth but with different partition.

```
Experiment 1:
```

Initial accuracy:0.91
Test/Train Adjusted Accuracy: 0.9593
Depth Adjusted Final Accuracy: 0.9837
Optimal depth size: 5
Test/Train Partition: 18.00 %
0/1 Error: 0.016350

```
Experiment 2:
```

Initial accuracy:0.96
Test/Train Adjusted Accuracy: 0.9385
Depth Adjusted Final Accuracy: 0.9665
Optimal depth size: 3
Test/Train Partition: 18.00 %
0/1 Error: 0.033520

```
Experiment 3:
```

Initial accuracy:0.94 Test/Train Adjusted Accuracy: 0.9351 Depth Adjusted Final Accuracy: 0.9514 Optimal depth size: 5 Test/Train Partition: 31.00 % 0/1 Error: 0.048649

We see that the accuracy decreases with the implementation of predetermined train/test split in second experiment. The reason of this, our first split had found a better value for splitting the dataset.

`0/1 Error` for both experiments as they shown above. Here we can interpret that when depth size increase, accuracy also increases.

In [587]:

```
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import tree
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.datasets import load_svmlight_file
import pydotplus
import random

def get_data(filename):
    data = load_svmlight_file(filename)
    return data[0], data[1]
```

In [588]:

```
# Reading dataset
X, y =  get_data("breast-cancer_scale.txt")
col_names = ['age',"menopause","tumor-size","inv-nodes","node-caps","deg-malig","breas
t","breast-quad","irradiat","Class"]
size = random.uniform(0.1, 0.5)
```

In Decision Trees we don't have to standardize our data unlike PCA and logistic regression which are sensitive to effects of not standardizing the data. So we can directly start implementing our model on the data.

In [589]:

```
def split_data(i):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=size)

    # Create Decision Tree classifer object
    clf = DecisionTreeClassifier()

    # Train Decision Tree Classifer
    clf = clf.fit(X_train,y_train)

    # Predict the response values
    y_pred = clf.predict(X_test)

    # Get initial score by using test dataset
    score = clf.score(X_test, y_test)

    return size, score
```

In [590]:

```
#Find the split size, which gives the best accuracy for this seed. In different runs, o
ptimize size will change
sizes = []
scores = []

size_range = list(range(1, 50))
for i in size_range:
    size, score = split_data(i)
    scores.append(score)
    sizes.append(size)

opt_size = scores.index(max(scores))
print("Optimized test size:", opt_size,'%')
```

```
Optimized test size: 31 %
```

In [591]:

```
# In order to make comparison we hold test size same for to tests with different depths
#opt_size = 18
```

The line below stands for implementation of just calculated ideal split size.

In [592]:

```
#Split dataset by using optimal split size for the same seed
ideal_size = sizes[opt_size]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=ideal_size)

# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response values
y_pred = clf.predict(X_test)
score1 = metrics.accuracy_score(y_test, y_pred)
```

Here we will find optimal depth value. The maximum depth of the tree is None, then nodes are expanded until all the leaves contain less than min_samples_split samples. The higher value of maximum depth causes overfitting, however, a lower value causes underfitting.

Additionally, max_depth is not the same thing as depth of a decision tree. max_depth is just a way to preprune a decision tree.

In [593]:

```
# List of values to use as test for max_depth:
max_depth_range = list(range(1, 20))
# List to store the average RMSE for each value of max_depth:
accuracy = []
for depth in max_depth_range:

    clf = DecisionTreeClassifier(max_depth = depth)
    clf.fit(X_train, y_train)
    score = clf.score(X_test, y_test)
    accuracy.append(score)
depth = accuracy.index(max(accuracy))+1
```

In [594]:

```
depth = 5
clf = DecisionTreeClassifier(max_depth=depth)
#random_state = 42
# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

# Model Accuracy, how often is the classifier correct?
score2 = metrics.accuracy_score(y_test, y_pred)
loss = metrics.zero_one_loss(y_test, y_pred)
```

In [595]:

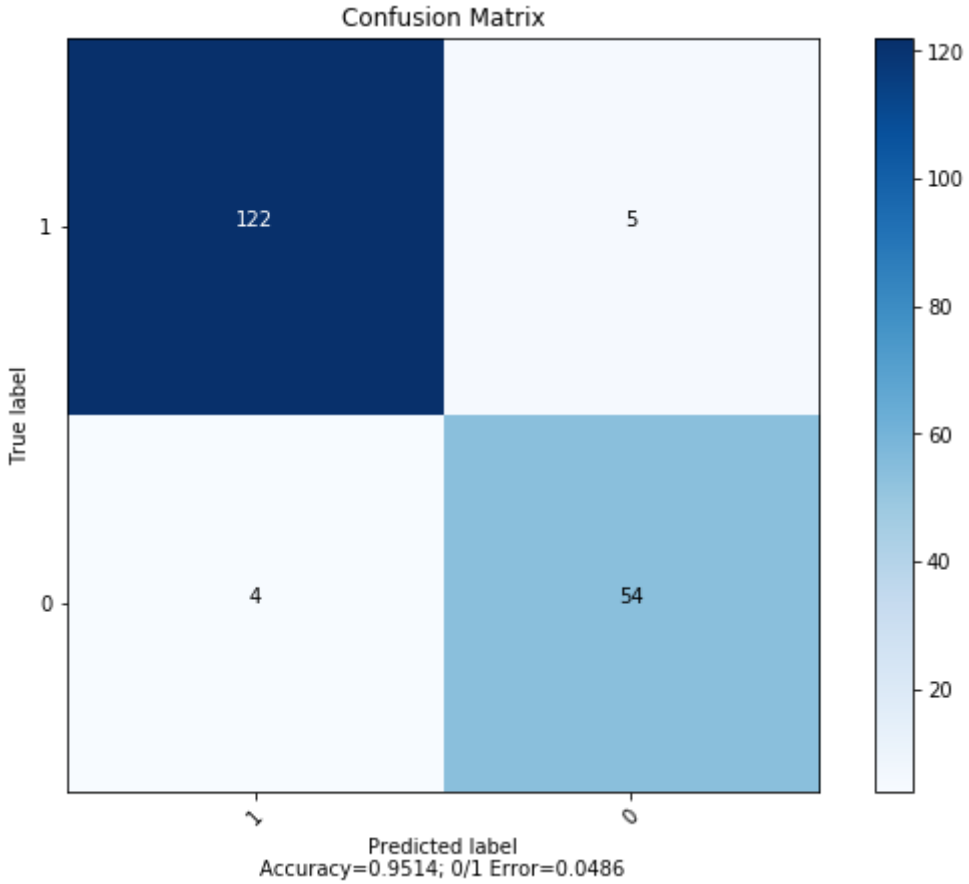
```
# Confusion Matrix from sklearn
def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True):
    import matplotlib.pyplot as plt
    import numpy as np
    import itertools

    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy
    if cmap is None:
        cmap = plt.get_cmap('Blues')
    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="black" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\nAccuracy={:0.4f}; 0/1 Error={:0.4f}'.format(accuracy,
misclass))
    plt.show()
```

Confusion matrix confirms that our model has very good accuracy.

In [596]:

```
# Plot Confusion Matrix
fig = plt.figure()
plot_confusion_matrix(cm= np.asarray(confusion_matrix(y_test, y_pred)),
                      normalize    = False,
                      target_names = ['1', '0'],
                      title        = "Confusion Matrix")
plt.savefig('confusion_matrix'+'.jpg')
```

<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>

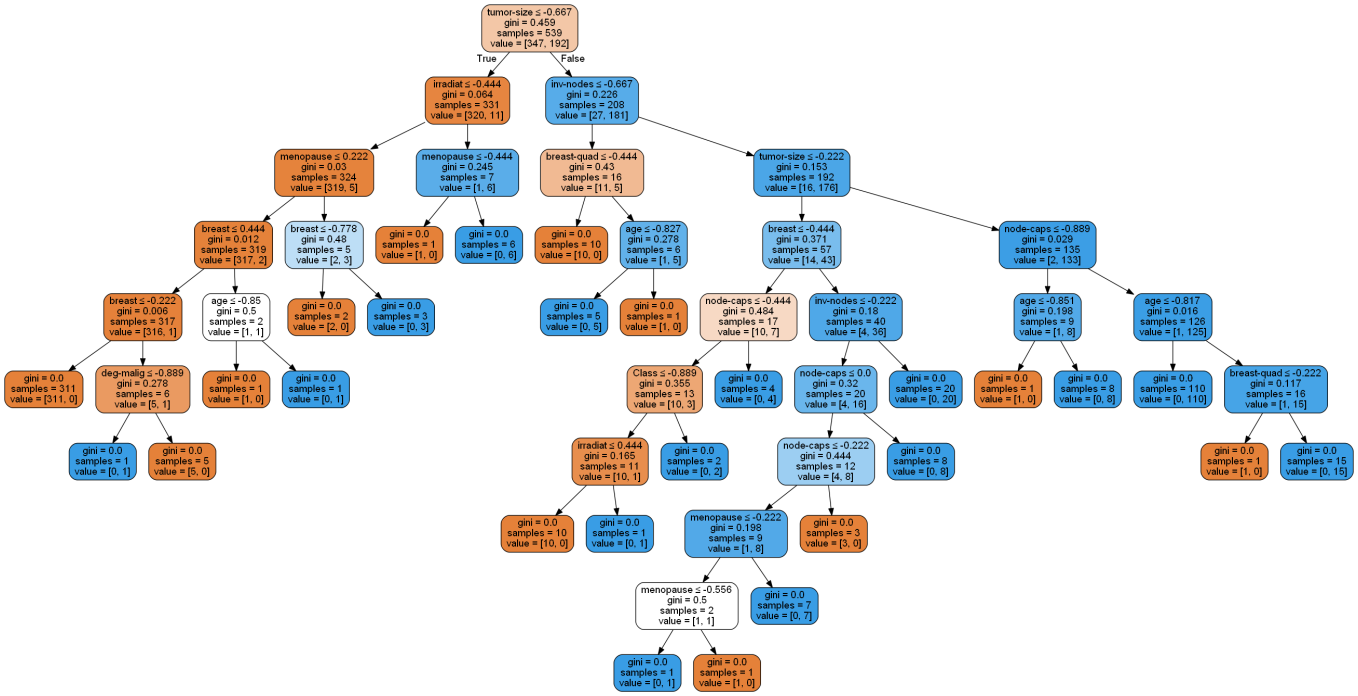
In [597]:

```
def plot_tree():
    dot_data = StringIO()
    export_graphviz(clf, out_file=dot_data,
                    filled=True, rounded=True,
                    special_characters=True, feature_names= col_names)
    graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    Image(graph.create_png())
    graph.write_png("DT for depth=" + str(depth)+ ",Size="  + str(opt_size) + ".png")
```

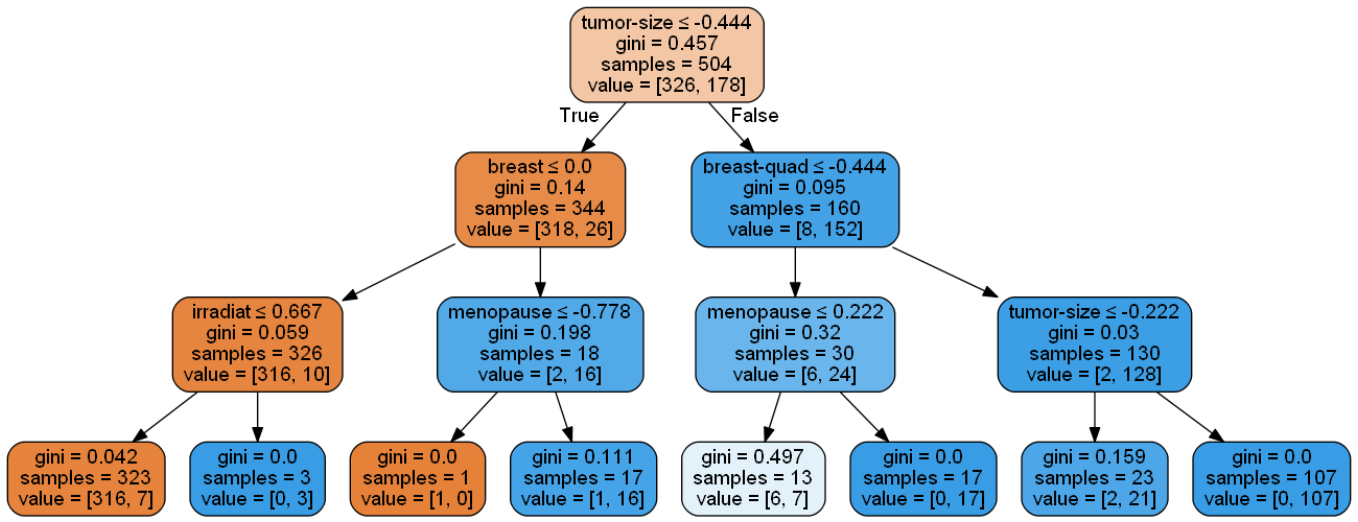
In [598]:

```
plot_tree()
```

Decision Tree for depth = 5



Decision Tree for depth = 3



Here I would like to add one more comparison to our decision trees.Importances of features calculates as shown below.

In [599]:

```
# Calculate importances of each feature
importances = pd.DataFrame({'Feature':col_names,'Importance':np.round(clf.feature_impor
tances_,3)})
importances = importances.sort_values('Importance',ascending=False)
```

Experiment1:

```
      Feature  Importance
```

2 - tumor-size 0.749
3 - inv-nodes 0.111
6 - breast 0.072
1 - menopause 0.024
7 - breast-quad 0.020
8 - irradiat 0.011
0 - age 0.010
4 - node-caps 0.002
5 - deg-malig 0.000
9 - Class 0.000

Experiment2:

```
       Feature  Importance<br>
```

2 - tumor-size 0.741
6 - breast 0.079
3 - inv-nodes 0.047
7 - breast-quad 0.038
1 - menopause 0.021
9 - Class 0.017
4 - node-caps 0.015
8 - irradiat 0.015
5 - deg-malig 0.014
0 - age 0.012

I would like to briefly explain how features are selected to evaluate while creating branches of the tree. Since it has greater importance and as ong as all features created 2 sub-branches, "tumor-size" is selected as first node of Experiment 2. This is our dependent variable. It is followed by other important feature "breast". However, as the third node we expect to have "inv-nodes" as its importance is bigger than all the rest. But we have an another feature here which is "breast-quad".

This is because false value of first node creates a bunch of different possibilities. Our algorithm look for most effective feature to evaluate, the feature that will be able to split as much as data points as it can.

The situation is exactly same in Experiment 1. Feature "irradiat" is selected as second node since its ability of classifying data in greater numbers.

One important thing to note is that if a feature has a low feature importance value, it doesn’t necessarily mean that the feature isn’t important for prediction, it just means that the particular feature wasn’t chosen at a particularly early level of the tree. It could also be that the feature could be identical or highly correlated with another informative feature.

In [602]:

```
# Print findings
#print("Initial accuracy:%0.2f" % (score))
#print("Test/Train Adjusted Accuracy: %0.4f" % (score1))
#print("Depth Adjusted Final Accuracy: %0.4f" % (score2))

#print("\nOptimal depth size: %d" % (depth))
#print("Test/Train Partition: %0.2f" % (opt_size),"%")
#print("\n",importances)

#print("\n0/1 Error: %0.6f" % (loss))

print("\nConfusion Matrix:\n",confusion_matrix(y_test, y_pred))
print("\nClassification Report:\n",classification_report(y_test, y_pred))
```

```
Confusion Matrix:
 [[122   5]
 [  4  54]]

Classification Report:
              precision    recall  f1-score   support

         2.0       0.97      0.96      0.96       127
         4.0       0.92      0.93      0.92        58

    accuracy                           0.95       185
   macro avg       0.94      0.95      0.94       185
weighted avg       0.95      0.95      0.95       185
```

Finally, I will compare experiment 1 and experiment 3.
Even tough they both have same depth, just because of their different partitions the way of our algorithm creates branches has been changed.
Smaller train test learn the model as shown below. Algorithm still thinks that the most important feature to evaluate first is "tumor-size". But it preferred "menopause" rather than "irridiat" or "inv-nodes"

It is also important to note that, algorithm evaluated same feature "tumor-size" in second depth with different value. This means that, result which are obtained by different threshold for "tumor-size" feature, still gives the best output among all the rest.