In [6]:
```
%matplotlib notebook
import pandas as pd, numpy as np
from sklearn.model_selection import cross_val_score, KFold, train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree._tree import TREE_LEAF
from sklearn.metrics import accuracy_score

data = pd.read_csv('spam.data.csv', delimiter = ' ', header = None)
X, y = data.iloc[:, :-1].values, data.iloc[:, -1:].values.flatten()
```

Problem 3

Part a

We build a CART classifier with no pruning and report the number of leaf nodes as well as the 10-fold cross-validation error rate.

In [7]:
```
#function to identify leaf nodes
def two_leaf_nodes(tree, nodes):
    def is_leaf(node, tree = tree):
        if node != TREE_LEAF and tree.children_left[node] == TREE_LEAF: return True
        else: return False
    return [node for node in nodes if is_leaf(tree.children_left[node])
                and is_leaf(tree.children_right[node])]

# initializig classifier
clf_cart = DecisionTreeClassifier(random_state = 0)

#training classifier
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.1)
clf_cart.fit(X_train, y_train)

#computing number of leaf nodes and 10-fold CVE
nu_leaf_nodes = 2 * len(two_leaf_nodes(clf_cart.tree_, range(clf_cart.tree_.node_count)))
cv_error = np.mean(1 - cross_val_score(clf_cart, X, y, cv = 10))

print 'Number of leaf nodes = ' + str(nu_leaf_nodes)
print '10-fold cross validation error = ' + str(cv_error)
```

```
Number of leaf nodes = 162
10-fold cross validation error = 0.0993618158383
```

Part b

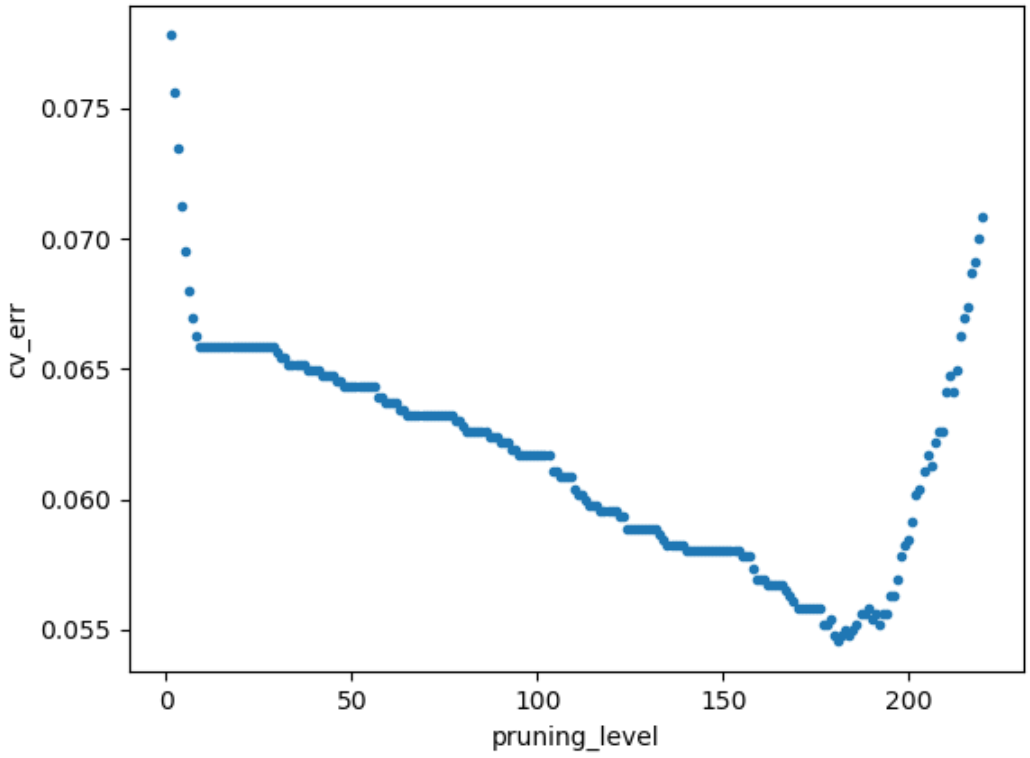
We perform increasing levels of pruning to the tree and observe the 10-fold CV error as function of number of leaves pruned.

In [8]:
```
# Functions to prune the node that most degrades performance of the model on test data

def make_node_leaf(node, tree, nodes):
    nodes.remove(tree.children_left[node])
    nodes.remove(tree.children_right[node])
    pruned_nodes = [tree.children_left[node], tree.children_right[node]]
    tree.children_left[node] = tree.children_right[node] = TREE_LEAF
    return nodes, pruned_nodes

def prune_worst_node_pair(clf_cart, nodes, X_test, y_test):
    tree = clf_cart.tree_
    two_leaf_nodes_ = two_leaf_nodes(tree, nodes)
    errs = []
    for node in two_leaf_nodes_:
        left, right = tree.children_left[node], tree.children_right[node]
        tree.children_left[node] = tree.children_right[node] = TREE_LEAF
        errs.append(1 - accuracy_score(clf_cart.predict(X_test), y_test))
        tree.children_left[node], tree.children_right[node] = left, right
    argmin_errs = np.argmin(errs)
    err = errs[argmin_errs]
    nodes, pruned_node_pair = make_node_leaf(two_leaf_nodes_[argmin_errs], tree, nodes)
    return err, pruned_node_pair, nodes
```

In [9]:
```
# Function to iteratively prune nodes in the tree and compute the 10-fold CVE as well as return the optimal
 tree

from copy import deepcopy

def cross_val_prune(clf_cart, folds = 10, pairs_to_prune = 1):
    errs = np.zeros(shape = (folds, pairs_to_prune))
    unpruned_clfs, pruned_clfs = [], []
    split_indices = list(KFold(n_splits = folds, shuffle = True).split(X))
    for fold_nu in range(folds):
        train_index, test_index = split_indices[fold_nu]
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]
        clf_cart.fit(X_train, y_train)
        unpruned_clfs.append(deepcopy(clf_cart))
        nodes = range(clf_cart.tree_.node_count)
        for prune_nu in range(pairs_to_prune):
            errs[fold_nu, prune_nu], pruned_node_pair, nodes = prune_worst_node_pair(clf_cart \
            , nodes, X_test, y_test)
        pruned_clfs.append(deepcopy(clf_cart))
    least_err_arg = np.argmin(errs[:, -1])
    unpruned_clf, pruned_clf = unpruned_clfs[least_err_arg], pruned_clfs[least_err_arg]
    errs_mean = np.mean(errs, axis = 0)
    return errs_mean, unpruned_clf, pruned_clf
```

In [10]:
```
# Examining the effect of pruning on the 10-fold CVE

from matplotlib import pyplot as plt

cv_errs = cross_val_prune(DecisionTreeClassifier(random_state = 0), pairs_to_prune = 220)[0]

print 'Lowest cross validation error of ' + str(round(np.amin(cv_errs), 4)) \
+ ' obtained when we prune ' + str(np.argmin(cv_errs) + 1) + ' leaves from the tree.'

ax = plt.subplots()[1]
ax.set_xlabel('pruning_level'); ax.set_ylabel('cv_err')
ax.plot(np.arange(len(cv_errs)) + 1., cv_errs, linestyle = '', marker = '.')
```

```
Lowest cross validation error of 0.0546 obtained when we prune 181 leaves from the tree.
```



Out[10]:
```
[<matplotlib.lines.Line2D at 0xc17d358>]
```

Part c

Plotting the optimal tree with pruning that minimizes the 10-fold CVE. We do this by simply returning the one out of 10 trees that produces the smallest CVE at the optimal pruning level obtained from Part b.

In [ ]:
```
cv_errs, unpruned_clf, pruned_clf = cross_val_prune(DecisionTreeClassifier(random_state = 0),
                                                    pairs_to_prune = 176)

from sklearn import tree as TreeModule
import graphviz
dot_data = TreeModule.export_graphviz(unpruned_clf, out_file=None,
                          feature_names=None,
                          class_names=['spam', 'not spam'],
                          filled=True, rounded=True,
                          special_characters=True)
graph = graphviz.Source(dot_data)
graph.render("unpruned")

dot_data = TreeModule.export_graphviz(pruned_clf, out_file=None,
                          feature_names=None,
                          class_names=['spam', 'not spam'],
                          filled=True, rounded=True,
                          special_characters=True)
graph = graphviz.Source(dot_data)
graph.render("pruned")
```

In [ ]:
```

```