In [19]:
```
import numpy as np
from sklearn.cross_validation import train_test_split
```

In [20]:
```
datafile = open('../data/spambase.data','r')
data = []
for line in datafile:
    line = [float(element) for element in line.rstrip('\n').split(',')]
    data.append(np.asarray(line))
```

In [21]:
```
num_features = 48
X = [data[i][:num_features] for i in range(len(data))]
y = [int(data[i][-1]) for i in range(len(data))]
```

In [22]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

In [23]:
```
#Making likelihood estimations

#Find the two classes
X_train_class_0 = [X_train[i] for i in range(len(X_train)) if y_train[i]==0]
X_train_class_1 = [X_train[i] for i in range(len(X_train)) if y_train[i]==1]

#Find the class specific likelihoods of each feature
likelihoods_class_0 = np.mean(X_train_class_0, axis=0)/100.0
likelihoods_class_1 = np.mean(X_train_class_1, axis=0)/100.0
```

In [24]:
```
#Calculate the class priors
num_class_0 = float(len(X_train_class_0))
num_class_1 = float(len(X_train_class_1))

prior_probability_class_0 = num_class_0 / (num_class_0 + num_class_1)
prior_probability_class_1 = num_class_1 / (num_class_0 + num_class_1)

log_prior_class_0 = np.log10(prior_probability_class_0)
log_prior_class_1 = np.log10(prior_probability_class_1)
```

In [25]:
```
def calculate_log_likelihoods_with_naive_bayes(feature_vector, Class):
    assert len(feature_vector) == num_features
    log_likelihood = 0.0 #using log-likelihood to avoid underflow
    if Class==0:
        for feature_index in range(len(feature_vector)):
            if feature_vector[feature_index] == 1: #feature present
                log_likelihood += np.log10(likelihoods_class_0[feature_index])
            elif feature_vector[feature_index] == 0: #feature absent
                log_likelihood += np.log10(1.0 - likelihoods_class_0[feature_index])
    elif Class==1:
        for feature_index in range(len(feature_vector)):
            if feature_vector[feature_index] == 1: #feature present
                log_likelihood += np.log10(likelihoods_class_1[feature_index])
            elif feature_vector[feature_index] == 0: #feature absent
                log_likelihood += np.log10(1.0 - likelihoods_class_1[feature_index])
    else:
        raise ValueError("Class takes integer values 0 or 1")

    return log_likelihood
```

In [26]:
```
def calculate_class_posteriors(feature_vector):
    log_likelihood_class_0 = calculate_log_likelihoods_with_naive_bayes(feature_vector, Class=0)
    log_likelihood_class_1 = calculate_log_likelihoods_with_naive_bayes(feature_vector, Class=1)

    log_posterior_class_0 = log_likelihood_class_0 + log_prior_class_0
    log_posterior_class_1 = log_likelihood_class_1 + log_prior_class_1

    return log_posterior_class_0, log_posterior_class_1
```

In [31]:
```
def classify_spam(document_vector):
    feature_vector = [int(element>0.0) for element in document_vector]
    log_posterior_class_0, log_posterior_class_1 = calculate_class_posteriors(feature_vector)
    if log_posterior_class_0 > log_posterior_class_1:
        return 0
    else:
        return 1
```

In [32]:
```
#Predict spam or not on the test set
```

```
predictions = []
for email in X_test:
    predictions.append(classify_spam(email))
```

In [48]:
```
def evaluate_performance(predictions, ground_truth_labels):
    correct_count = 0.0
    for item_index in xrange(len(predictions)):
        if predictions[item_index] == ground_truth_labels[item_index]:
            correct_count += 1.0
    accuracy = correct_count/len(predictions)
    return accuracy
```

In [49]:
```
accuracy_of_naive_bayes = evaluate_performance(predictions, y_test)
print accuracy_of_naive_bayes
```

```
0.892267593397
```

In [50]:
```
for i in range(100):
    print predictions[i], y_test[i]
```

```
0 0
1 0
0 0
1 1
0 0
1 1
0 0
0 0
0 0
1 0
0 0
1 0
0 0
1 1
1 1
0 0
1 1
0 1
1 1
0 0
0 0
1 1
0 1
0 0
1 1
0 0
1 1
0 0
1 1
1 1
1 1
1 1
0 0
1 0
0 0
0 0
1 1
1 1
1 0
1 1
1 1
0 0
0 0
0 0
1 1
1 1
1 1
0 0
0 1
1 1
1 1
0 0
0 0
1 1
0 0
0 0
0 0
0 0
```

```
0 0
0 0
1 1
0 0
1 1
1 1
1 1
0 0
0 0
0 0
1 1
0 0
1 0
0 0
0 0
1 1
1 1
0 0
0 0
0 0
0 0
1 1
0 0
0 0
0 0
0 0
0 0
0 0
1 1
1 1
1 1
1 0
1 1
1 1
0 0
0 1
0 0
0 0
1 1
1 0
1 1
1 1
```