HW1_problem5

March 11, 2015

Data Preparation

In [32]:
```
import pandas as pd

with open("/home/vicky/Downloads/agaricus-lepiota.data") as infile:
    data = pd.read_csv(infile, header=None, names=["class"] + ["f%s" % i for i in xrange(1,23)]

print data.shape
data.head()
```

```
(8124, 23)
```

Out[32]:

	class	f1	f2	f3	f4	f5	f6	f7	f8	f9	...	f13	f14	f15	f16	f17	f18	f19	f20	f21	f22
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	k	s	u
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	n	g
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	n	m
3	p	x	y	w	t	p	f	c	n	n	...	s	w	w	p	w	o	p	k	s	u
4	e	x	s	g	f	n	f	w	b	k	...	s	w	w	p	w	o	e	n	a	g

[5 rows x 23 columns]

Naive Bayes

In [64]:
```
from sklearn.naive_bayes import MultinomialNB
from sklearn.preprocessing import  LabelBinarizer
from sklearn_pandas import DataFrameMapper
import numpy as np
from sklearn.cross_validation import cross_val_score, StratifiedKFold

flabels = data.columns.tolist()
flabels.remove("class")

mapper = DataFrameMapper([(k, LabelBinarizer()) for k in flabels])
features = mapper.fit_transform(data[flabels])

labels = DataFrameMapper([("class", LabelBinarizer())]).fit_transform(data)

nb_model = MultinomialNB()
cross_val_score(nb_model, features, labels.ravel(), cv=5, scoring='roc_auc')
```

Out[64]:
```
array([ 0.85493192,  0.99843922,  0.95022646,  1.        ,  0.89638316])
```

Tree Augment Naive Bayes (TAN)

1. using package PyBool to perform the chow-liu algorithm (http://pythonhosted.org/pybool/apidoc/pybool.html)

```
In [69]: import sys
         sys.path.append("/home/vicky/Downloads/pybool/python/")

In [87]: import pybool.chow_liu_trees as CLT
         data["string"] = data[flabels].apply(lambda x: "".join(x), axis=1)
         s = CLT.build_chow_liu_tree(data.string.tolist(), 22)

In [103]: import networkx
          Ts = networkx.minimum_spanning_tree(s)
          print sorted(Ts.edges())

[(0, 10), (1, 10), (2, 4), (3, 18), (4, 19), (5, 13), (6, 20), (7, 8), (8, 9), (8, 19), (10, 19), (10, 1

In [101]: import networkx as nx
          G=nx.cycle_graph(4)
          G.add_edge(0,3,weight=2) # assign weight 2 to edge 0-3
          T=nx.minimum_spanning_tree(G)
          G.edges()
          print T.edges()

[(0, 1), (1, 2), (2, 3)]

In [111]: td = Ts.to_directed()
          td.

Out[111]: {}

In [ ]:
```