titanic-decisiontree

March 30, 2017

0.1 The problem we would like to solve is to determine if a Titanic's passenger would have survived, given her age, passenger class, and sex.

```
In [1]: import IPython
        import sklearn as sk
        import numpy as np
        import matplotlib
        import matplotlib.pyplot as plt
        import pydot
        import pyparsing

        print 'IPython version:', IPython.__version__
        print 'numpy version:', np.__version__
        print 'scikit-learn version:', sk.__version__
        print 'matplotlib version:', matplotlib.__version__
        print 'pydot version:', pydot.__version__
        print 'pyparsing version:', pyparsing.__version__
```

```
IPython version: 4.0.0
numpy version: 1.11.3
scikit-learn version: 0.18.1
matplotlib version: 1.5.1
pydot version: 1.0.28
pyparsing version: 1.5.6
```

```
In [2]: import csv
        with open('titanic-data.txt', 'rb') as csvfile:
            titanic_reader = csv.reader(csvfile, delimiter=',', quotechar='"')

            # Header contains feature names
            row = titanic_reader.next()
            feature_names = np.array(row)

            # Load dataset, and target classes
            titanic_X, titanic_y = [], []
            for row in titanic_reader:
                titanic_X.append(row)
```

```
            titanic_y.append(row[2]) # The target value is "survived"

        titanic_X = np.array(titanic_X)
        titanic_y = np.array(titanic_y)
```

```
In [4]: print feature_names, titanic_X[0], titanic_y[0]
```

```
['row.names' 'pclass' 'survived' 'name' 'age' 'embarked' 'home.dest' 'room'
 'ticket' 'boat' 'sex'] ['1' '1st' '1' 'Allen, Miss Elisabeth Walton' '29.0000' 'Southampton'
 'St Louis, MO' 'B-5' '24160 L221' '2' 'female'] 1
```

```
In [5]: # we keep the class, the age and the sex
        titanic_X = titanic_X[:, [1, 4, 10]]
        feature_names = feature_names[[1, 4, 10]]
        print feature_names
        print titanic_X[12], titanic_y[12]
```

```
['pclass' 'age' 'sex']
['1st' 'NA' 'female'] 1
```

```
In [6]: ages = titanic_X[:, 1]
        mean_age = np.mean(titanic_X[ages != 'NA', 1].astype(np.float))
        titanic_X[titanic_X[:, 1] == 'NA', 1] = mean_age
        print feature_names
        print titanic_X[12], titanic_y[12]
```

```
['pclass' 'age' 'sex']
['1st' '31.1941810427' 'female'] 1
```

```
In [7]: # Sklearn
        from sklearn.preprocessing import LabelEncoder
        enc = LabelEncoder()
        label_encoder = enc.fit(titanic_X[:, 2])
        print "Categorical classes:", label_encoder.classes_
        integer_classes = label_encoder.transform(label_encoder.classes_)
        print "Integer classes:", integer_classes
        t = label_encoder.transform(titanic_X[:, 2])
        titanic_X[:, 2] = t
        print 'Feature names:',feature_names
        print 'Features for instance number 12:',titanic_X[12], titanic_y[12]
```

```
Categorical classes: ['female' 'male']
Integer classes: [0 1]
Feature names: ['pclass' 'age' 'sex']
Features for instance number 12: ['1st' '31.1941810427' '0'] 1
```

In [8]:
```
from sklearn.preprocessing import OneHotEncoder

enc = LabelEncoder()
label_encoder = enc.fit(titanic_X[:, 0])
print "Categorical classes:", label_encoder.classes_
integer_classes = label_encoder.transform(label_encoder.classes_).reshape(3, 1)
print "Integer classes:", integer_classes
enc = OneHotEncoder()
one_hot_encoder = enc.fit(integer_classes)
# First, convert clases to 0-(N-1) integers using label_encoder
num_of_rows = titanic_X.shape[0]
t = label_encoder.transform(titanic_X[:, 0]).reshape(num_of_rows, 1)
# Second, create a sparse matrix with three columns, each one indicating if the instan
new_features = one_hot_encoder.transform(t)
# Add the new features to titanix_X
titanic_X = np.concatenate([titanic_X, new_features.toarray()], axis = 1)
#Eliminate converted columns
titanic_X = np.delete(titanic_X, [0], 1)
# Update feature names
feature_names = ['age', 'sex', 'first_class', 'second_class', 'third_class']
# Convert to numerical values
titanic_X = titanic_X.astype(float)
titanic_y = titanic_y.astype(float)
```

```
Categorical classes: ['1st' '2nd' '3rd']
Integer classes: [[0]
 [1]
 [2]]
```

In [9]:
```
print 'New feature names:',feature_names
print 'Values:',titanic_X[0]
```

```
New feature names: ['age', 'sex', 'first_class', 'second_class', 'third_class']
Values: [ 29.   0.   1.   0.   0.]
```

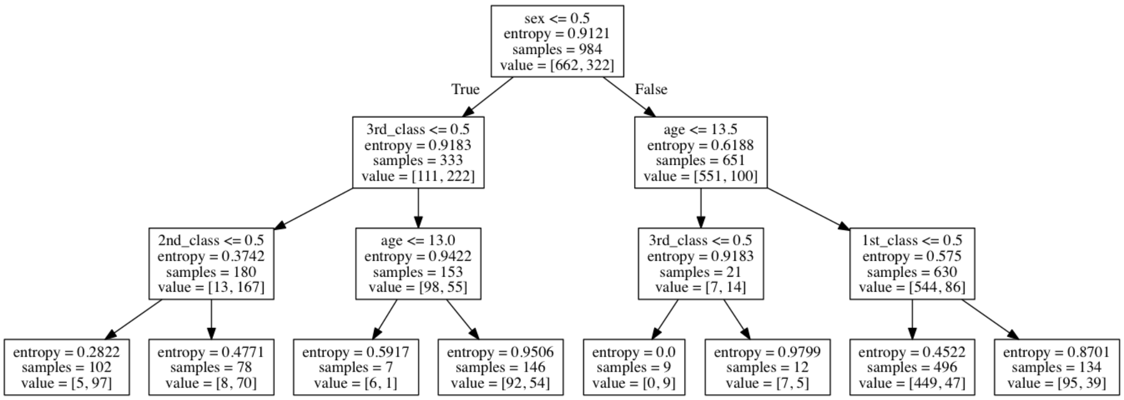
In [10]:
```
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(titanic_X, titanic_y, test_size=0
```

```
/Users/youngseoklee/anaconda/lib/python2.7/site-packages/sklearn/cross_validation.py:44: Depre
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [11]:
```
# Decision tree generation
from sklearn import tree
clf = tree.DecisionTreeClassifier(criterion='entropy', max_depth=3,min_samples_leaf=5)
clf = clf.fit(X_train,y_train)
```

In [13]:
```python
# Decision tree visualization
import StringIO
dot_data = StringIO.StringIO()
tree.export_graphviz(clf, out_file=dot_data, feature_names=['age','sex','1st_class','2
graph = pydot.graph_from_dot_data(dot_data.getvalue())
graph.write_png('titanic.png')
from IPython.core.display import Image
Image(filename='titanic.png')
```

Out[13]:



In [14]:
```python
from sklearn import metrics
def measure_performance(X,y,clf, show_accuracy=True, show_classification_report=True,
    y_pred=clf.predict(X)
    if show_accuracy:
        print "Accuracy:{0:.3f}".format(metrics.accuracy_score(y,y_pred)),"\n"

    if show_classification_report:
        print "Classification report"
        print metrics.classification_report(y,y_pred),"\n"

    if show_confusion_matrix:
        print "Confusion matrix"
        print metrics.confusion_matrix(y,y_pred),"\n"

measure_performance(X_train,y_train,clf, show_classification_report=False, show_confus
```

```
Accuracy:0.838
```

In [16]:
```python
from sklearn.cross_validation import cross_val_score, LeaveOneOut
from scipy.stats import sem
```

```
def loo_cv(X_train,y_train,clf):
    # Perform Leave-One-Out cross validation
    # We are preforming 1313 classifications!
    loo = LeaveOneOut(X_train[:].shape[0])
    scores=np.zeros(X_train[:].shape[0])
    for train_index,test_index in loo:
        X_train_cv, X_test_cv= X_train[train_index], X_train[test_index]
        y_train_cv, y_test_cv= y_train[train_index], y_train[test_index]
        clf = clf.fit(X_train_cv,y_train_cv)
        y_pred=clf.predict(X_test_cv)
        scores[test_index]=metrics.accuracy_score(y_test_cv.astype(int), y_pred.astype
    print ("Mean score: {0:.3f} (+/-{1:.3f})").format(np.mean(scores), sem(scores))
```

```
In [17]: loo_cv(X_train, y_train,clf)
```

```
Mean score: 0.837 (+/-0.012)
```

```
In [18]: from sklearn.ensemble import RandomForestClassifier
         clf = RandomForestClassifier(n_estimators=10,random_state=33)
         clf = clf.fit(X_train,y_train)
         loo_cv(X_train,y_train,clf)
```

```
Mean score: 0.817 (+/-0.012)
```

```
In [19]: clf_dt=tree.DecisionTreeClassifier(criterion='entropy', max_depth=3,min_samples_leaf=
         clf_dt.fit(X_train,y_train)
         measure_performance(X_test,y_test,clf_dt)
```

```
Accuracy:0.793

Classification report
             precision    recall  f1-score   support

        0.0       0.77      0.96      0.85       202
        1.0       0.88      0.54      0.67       127

avg / total       0.81      0.79      0.78       329


Confusion matrix
[[193   9]
 [ 59  68]]
```

```
In [ ]:
```