bnn_mvp_model_selection

August 21, 2015

In [4]:
```
"""
The intent of this notebook is model selection and
evaluation for the MVP of our brainNN classifier.
"""
import sys
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from tornado import gen
from tornado.ioloop import IOLoop
import aimetrics as aim
import aimetrics.metrics as aim_metrics
import seaborn as sns
%matplotlib inline
```

In [5]:
```
X_trn_val = pd.read_csv('output/bnn-mvp/X_trn_val.csv', index_col=0)
y_trn_val = pd.read_csv('output/bnn-mvp/y_trn_val.csv', index_col=0)
X_test = pd.read_csv('output/bnn-mvp/X_test.csv', index_col=0)
y_test = pd.read_csv('output/bnn-mvp/y_test.csv', index_col=0)
labels = ['small_drone', 'person']
# create data storage variable
metrics = {}
```

In [6]:
```
from sklearn.cross_validation import  StratifiedKFold


# put together k-fold output
skf = StratifiedKFold(y_trn_val['small_drone'],  5)



@gen.coroutine
def get_default_metrics():
    indx = list(skf) # for debugging
    metrics['default'] = yield [
        aim_metrics.remote_classifier_metrics(
            'http://localhost:3002/',
            'bnn',
            X_trn_val.iloc[trn_ind].values,
            y_trn_val.iloc[trn_ind].values,
            X_trn_val.iloc[val_ind].values,
            y_trn_val.iloc[val_ind].values,
            labels
```

```
            )
            for trn_ind, val_ind in indx
        ]
    IOLoop.instance().add_callback(get_default_metrics)
```

In [ ]:
```
row_range = range(0, 6)
col_range = range(2,7)
rate_range = np.arange(0.1, 0.8, 0.1)
n_folds = 5
print("TESTING %d MODELS" % (len(row_range)*len(col_range)*len(rate_range)*n_folds)

@gen.coroutine
def get_param_grid_metrics():
    yield [get_param_metrics(nrows, ncols, learning_rate=rate) for nrows in row_range for ncols
    print("PARAMETER SEARCH COMPLETE")



@gen.coroutine
def get_param_metrics(n_hidden_rows, ncols, learning_rate=None):
    """Get the metrics for a particular parameter set.  Assumes a grid topo"""
    skf = StratifiedKFold(y_trn_val['small_drone'],  n_folds)
    params = {'hiddenLayers': [ncols] * n_hidden_rows}
    if learning_rate:
        params['learningRate'] = learning_rate
    key = 'x'.join((str(n_hidden_rows), str(ncols), str(learning_rate)))
    metrics[key] = yield [
        aim_metrics.remote_classifier_metrics(
            'http://localhost:3002/',
            'bnn',
            X_trn_val.iloc[trn_ind].values,
            y_trn_val.iloc[trn_ind].values,
            X_trn_val.iloc[val_ind].values,
            y_trn_val.iloc[val_ind].values,
            labels,
            model_params = params,
        )
        for trn_ind, val_ind in skf
    ]
    print("%s Complete" % key)

IOLoop.instance().add_callback(get_param_grid_metrics)
```

In [8]:
```
def get_score_report(key, score, agg):
    stats = pd.Series([r[score] for r in metrics[key]]).describe()
    return stats[agg]
```

In [13]:
```
scores = ['roc_auc', 'acc', 'f1_score']
aggs = ['mean', 'std']
report = pd.DataFrame({
        score + "_" + agg: pd.Series({
                key: get_score_report(key, score, agg) for key in metrics.keys()
        }) for score in scores for agg in aggs
})
report.sort("roc_auc_mean", ascending=False)
```

Out[13]:

	acc_mean	acc_std	f1_score_mean	f1_score_std	roc_auc_mean
1x4x0.7	0.720635	5.678903e-02	0.693801	5.954350e-02	0.775120
1x6x0.6	0.717460	6.085806e-02	0.691295	6.203974e-02	0.774402
1x5x0.7	0.717460	6.085806e-02	0.689795	6.313095e-02	0.773923
1x5x0.6	0.717460	6.085806e-02	0.691295	6.203974e-02	0.772727
1x4x0.6	0.717460	6.085806e-02	0.691295	6.203974e-02	0.772488
1x2x0.6	0.720635	6.106471e-02	0.697888	5.774796e-02	0.772488
1x4x0.5	0.717460	6.085806e-02	0.691295	6.203974e-02	0.772488
1x6x0.7	0.714286	5.611959e-02	0.687389	5.499315e-02	0.772249
2x6x0.7	0.720635	4.840619e-02	0.693770	5.010105e-02	0.771770
1x5x0.5	0.717460	6.085806e-02	0.691295	6.203974e-02	0.771770
1x3x0.7	0.720635	5.678903e-02	0.695033	5.867176e-02	0.771770
1x2x0.7	0.717460	6.085806e-02	0.692527	6.126658e-02	0.771770
1x3x0.5	0.720635	6.106471e-02	0.696656	5.870837e-02	0.771531
1x3x0.6	0.717460	6.085806e-02	0.692527	6.126658e-02	0.771531
1x6x0.5	0.717460	6.085806e-02	0.691295	6.203974e-02	0.771292
1x3x0.4	0.723810	6.602106e-02	0.700435	6.587214e-02	0.771053
2x6x0.5	0.720635	6.106471e-02	0.695156	6.002805e-02	0.770813
1x2x0.4	0.726984	6.188441e-02	0.704442	6.158310e-02	0.770574
2x6x0.6	0.723810	4.840619e-02	0.699131	4.554871e-02	0.770574
1x5x0.4	0.720635	6.602106e-02	0.695074	6.922460e-02	0.770335
1x2x0.5	0.720635	6.106471e-02	0.697888	5.774796e-02	0.770335
1x2x0.3	0.723810	6.002435e-02	0.701843	6.068017e-02	0.770335
1x4x0.4	0.723810	6.602106e-02	0.700435	6.587214e-02	0.770096
1x5x0.3	0.723810	6.602106e-02	0.700435	6.587214e-02	0.770096
1x3x0.3	0.730159	5.832118e-02	0.708316	5.845061e-02	0.770096
default	0.723810	6.106471e-02	0.701711	6.237541e-02	0.770096
1x6x0.3	0.723810	6.309399e-02	0.698949	6.713256e-02	0.769976
2x4x0.6	0.720635	3.984095e-02	0.692722	3.324309e-02	0.769856
2x5x0.7	0.720635	5.566882e-02	0.687041	6.041036e-02	0.769856
2x5x0.3	0.717460	6.388766e-02	0.695462	6.597280e-02	0.769856
...	...	...	...	...	...
5x4x0.1	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
5x4x0.2	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x5x0.1	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
5x4x0.3	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
5x4x0.4	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
5x4x0.5	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
5x4x0.6	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x5x0.5	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x4x0.2	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x4x0.7	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x3x0.2	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
3x2x0.3	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x2x0.1	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x2x0.2	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x2x0.3	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x2x0.4	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x2x0.6	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x2x0.7	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x3x0.1	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x3x0.3	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000

4x4x0.6	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x3x0.4	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x3x0.5	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x3x0.6	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x3x0.7	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x4x0.1	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x4x0.3	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x4x0.4	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x4x0.5	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000
4x2x0.5	0.698413	1.490116e-08	0.574395	7.450581e-09	0.500000

	roc_auc_std
1x4x0.7	0.026029
1x6x0.6	0.024508
1x5x0.7	0.024962
1x5x0.6	0.027629
1x4x0.6	0.029170
1x2x0.6	0.027699
1x4x0.5	0.026199
1x6x0.7	0.028653
2x6x0.7	0.027621
1x5x0.5	0.028198
1x3x0.7	0.027164
1x2x0.7	0.028096
1x3x0.5	0.028002
1x3x0.6	0.028256
1x6x0.5	0.026979
1x3x0.4	0.028616
2x6x0.5	0.026769
1x2x0.4	0.028638
2x6x0.6	0.026699
1x5x0.4	0.028783
1x2x0.5	0.028218
1x2x0.3	0.029043
1x4x0.4	0.027686
1x5x0.3	0.027439
1x3x0.3	0.028924
default	0.027429
1x6x0.3	0.029207
2x4x0.6	0.029075
2x5x0.7	0.026769
2x5x0.3	0.025885
...	...
5x4x0.1	0.000000
5x4x0.2	0.000000
4x5x0.1	0.000000
5x4x0.3	0.000000
5x4x0.4	0.000000
5x4x0.5	0.000000
5x4x0.6	0.000000
4x5x0.5	0.000000
4x4x0.2	0.000000
4x4x0.7	0.000000
4x3x0.2	0.000000

```
3x2x0.3     0.000000
4x2x0.1     0.000000
4x2x0.2     0.000000
4x2x0.3     0.000000
4x2x0.4     0.000000
4x2x0.6     0.000000
4x2x0.7     0.000000
4x3x0.1     0.000000
4x3x0.3     0.000000
4x4x0.6     0.000000
4x3x0.4     0.000000
4x3x0.5     0.000000
4x3x0.6     0.000000
4x3x0.7     0.000000
4x4x0.1     0.000000
4x4x0.3     0.000000
4x4x0.4     0.000000
4x4x0.5     0.000000
4x2x0.5     0.000000

[182 rows x 6 columns]
```

In [14]:
```
import json
with open("../output/bnn-mvp/param_metrics.json", 'w') as f:
    json.dump(metrics, f)
report.to_csv("../output/bnn-mvp/param_metrics_summary.csv")
```

In [11]:
```
"""
Conclusion:

The 1-layer NN's almost categorically outperformed the others
in roc_auc. Of those, 1x4x0.7 was the best performer by a very
small margin.  Therefore, it will be the selected model for our
MVP.  However, I want to confirm this with an increased search
space this evening, in order to rule out somewhat larger
dimensions.

Selected Params:
{
    hiddenLayers: [4],
    learningRate: 0.7
}
"""
pass
```

In [ ]:

In [ ]: