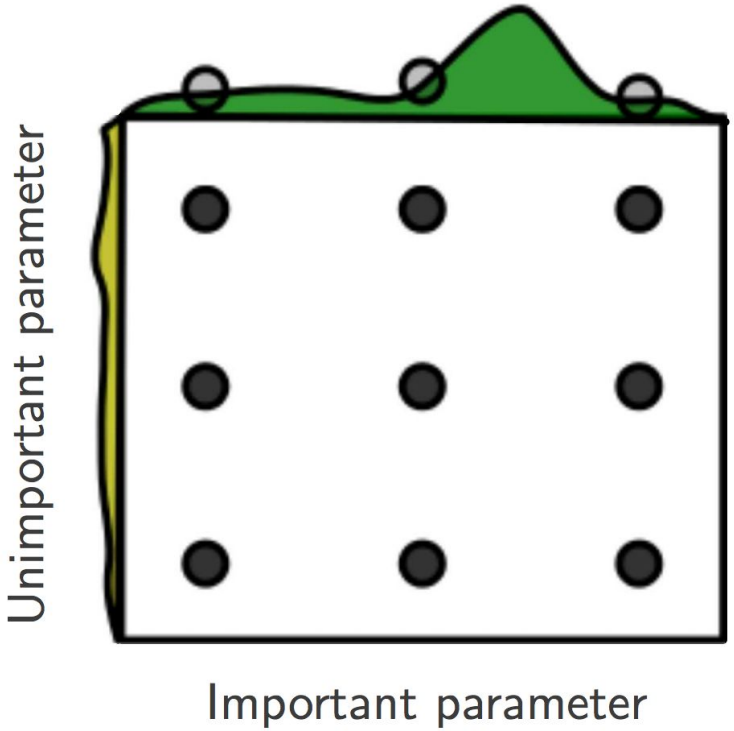

Hyperopt

We All Deal with Hyperparameters

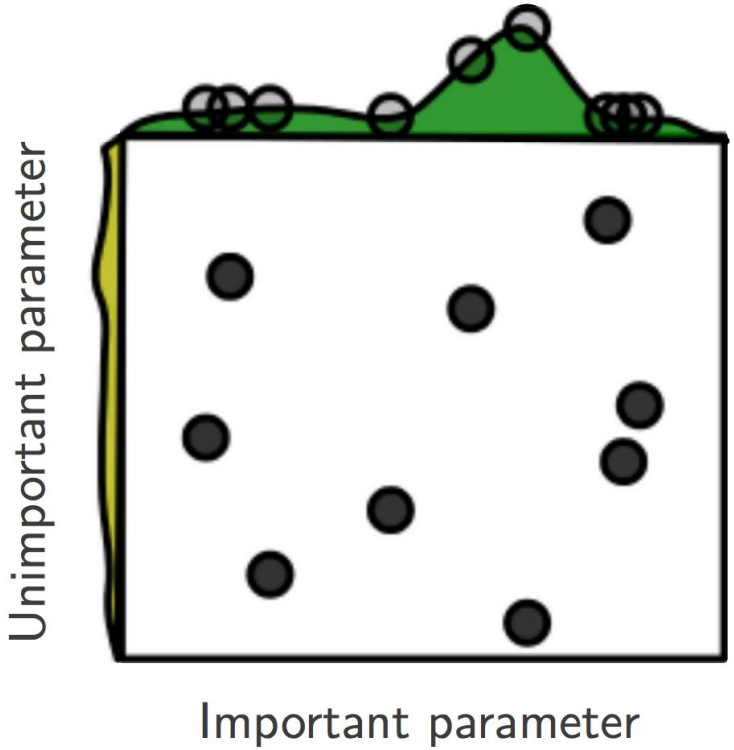
```
class sklearn.linear_model.LogisticRegression(penalty='l2', dual=False, tol=0.0001, C=1.0,
fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None,
solver='liblinear', max_iter=100, multi_class='ovr', verbose=0, warm_start=False, n_jobs=1)
```


Algorithm	Scikit-Learn	Hyperopt
Grid Search		
Random Search		
Tree of Parzen Estimators		

Grid Layout

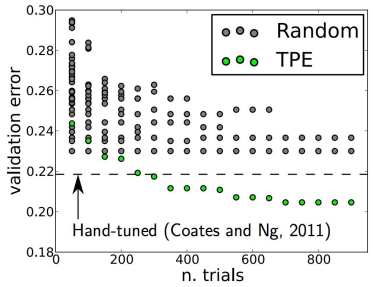


Random Layout



Hyperopt in Theory: numerical + categorical inputs

- Bayesian optimization using Tree-structured Parzen Estimators
- Everything is a Hyperparameter
  - Choice of algorithm
  - Preprocessing
    - Normalization
    - PCA
    - GMMs
- Parallel search framework w/ MongoDB (works well on a cluster)
  - fmin pulls jobs from mongo asynchronously - uses state w/in database



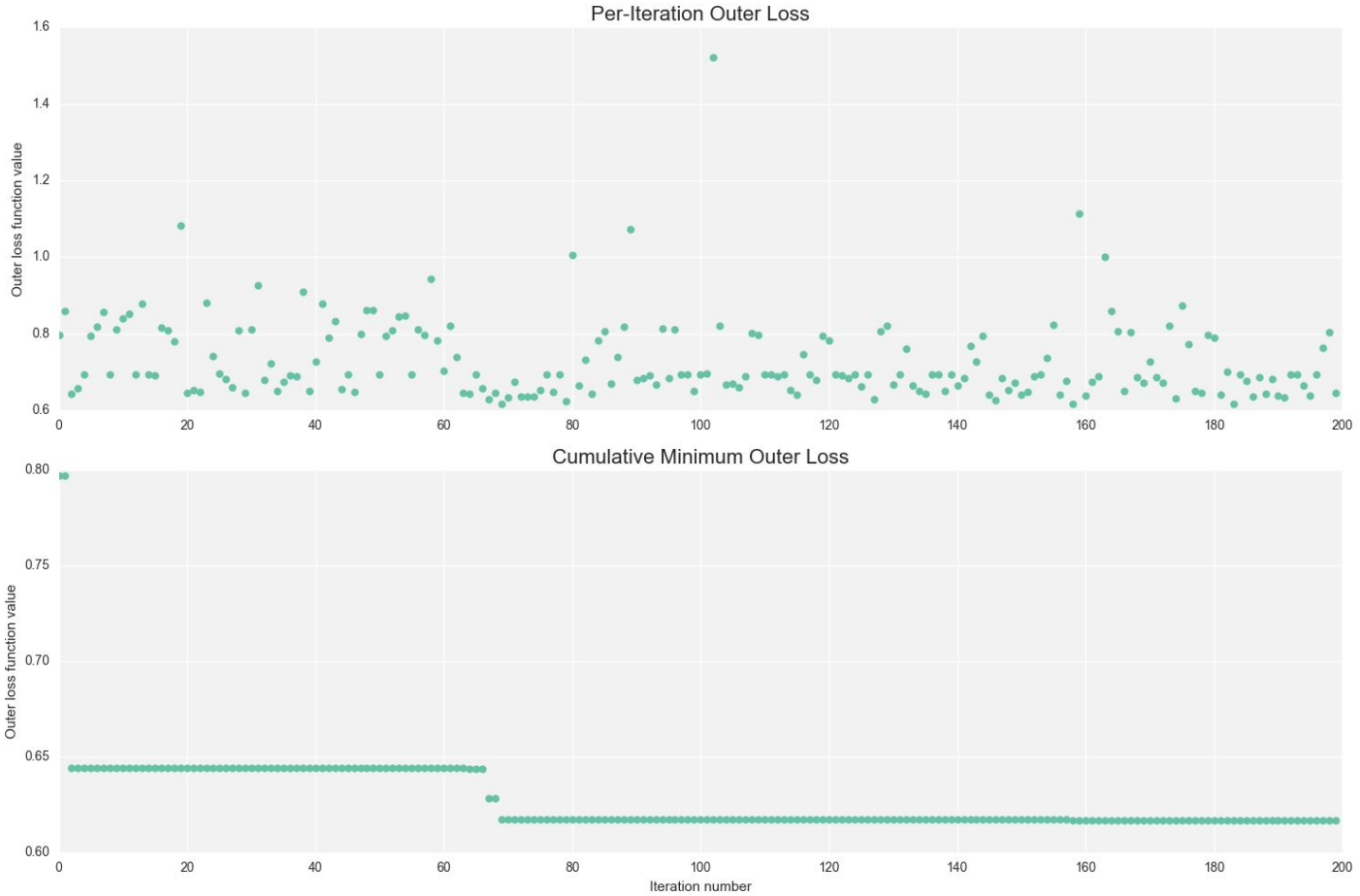
Method (# configs)	Test Acc. (%)
Hand-tuned	**79.1** ± .8
TPE (800)	**78.8** ± .8
Random (2K)	76.6 ± .8
Chance	10.0

Hyperopt in Practice

```
from hyperopt import hp, fmin, tpe, rand, space_eval

space = [hp.uniform('x', 0, 1), hp.normal('y', 0, 1)]

def f(args):
    x, y = args
    return x ** 2 + y ** 2

best = fmin(f, space, algo=tpe.suggest, max_evals=100)
print 'tpe:', best
```

Dealing with Large Search Spaces

```
def optimize(trials):
    space = {
             'n_estimators' : hp.quniform('n_estimators', 100, 1000, 1),
             'eta' : hp.quniform('eta', 0.025, 0.5, 0.025),
             'max_depth' : hp.quniform('max_depth', 1, 13, 1),
             'min_child_weight' : hp.quniform('min_child_weight', 1, 6, 1),
             'subsample' : hp.quniform('subsample', 0.5, 1, 0.05),
             'gamma' : hp.quniform('gamma', 0.5, 1, 0.05),
             'colsample_bytree' : hp.quniform('colsample_bytree', 0.5, 1, 0.05),
             'num_class' : 9,
             'eval_metric': 'mlogloss',
             'objective': 'multi:softprob',
             'nthread' : 6,
             'silent' : 1
             }
    best = fmin(score, space, algo=tpe.suggest, trials=trials, max_evals=250)
    print best
```


Per-Iteration Outer Loss
Outer loss function value
1.6
1.4
1.2
1.0
0.8
0.6
0
20
40
60
80
100
120
140
160
180
200
Cumulative Minimum Outer Loss
Outer loss function value
0.80
0.75
0.70
0.65
0.60
Iteration number

Conclusion

- Random Search is surprisingly good
- That said… TPE - choses values correlated with strong performance
  - Better for high dimensional spaces