Kaggle Competition Entry

Here are some quick predictors I whipped up to do a bit better than the homeworks on the Kaggle competition; in the end, I tried a few simple things and then make a stacked ensemble.

Warning:

(1) Don't use this as a guideline for your report, which should have a lot more discussion of how & why you tried these, and probably a lot more detailed exploration of how the performance progressed with each one (for example).

(2) Your project should involve more effort to tune or iterative improve on the models. These show a first cut only, without even an attempt to set the parameters of the learning and evaluate their fit. A good project should involve *not only* properly setting the paramters, but ideally some iterative attempts to improve each model (feature design or selection or similar modification) along with assessment of whether those attempts worked. (As an example, you can view the neural network or clustered logistic regression examples below as attempts to improve on logistic regression by increasing the model complexity, but this needs discussion.)

In [1]:
```
import numpy as np
np.random.seed(0)
import mltools as ml
import matplotlib.pyplot as plt   # use matplotlib for plotting with inl
%matplotlib inline

import mltools.linear
import mltools.linearC
import mltools.dtree
import mltools.cluster
import mltools.nnet
```

In [2]:
```
X = np.genfromtxt("../HW4/data/X_train.txt",delimiter=',')
Y = np.genfromtxt("../HW4/data/Y_train.txt",delimiter=',')
[Xt,Xv,Yt,Yv] = ml.splitData(X,Y,0.80)

Xe = np.genfromtxt('../HW4/data/X_test.txt',delimiter=',')

# Should rescale the data...
Xt,param = ml.transforms.rescale(Xt)
Xv,_ = ml.transforms.rescale(Xv,param)
Xe,_ = ml.transforms.rescale(Xe,param)
```

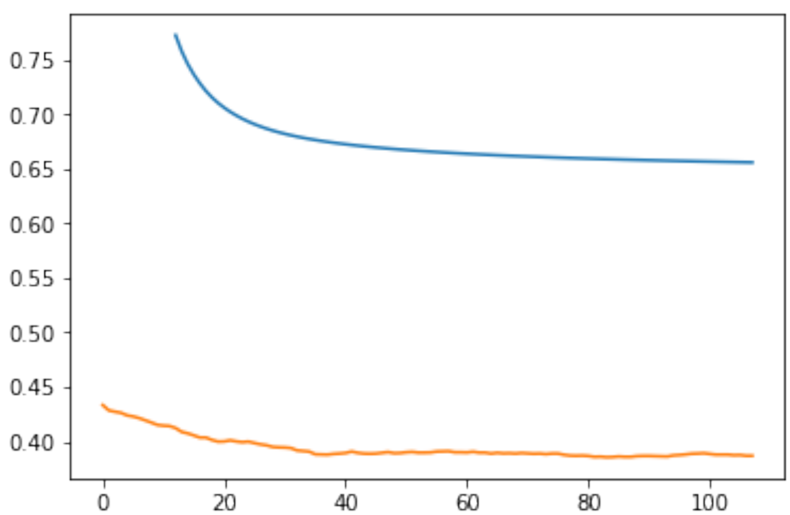
In [14]:
```
def toKaggle(filename,YeHat):
    fh = open(filename,'w')                    # open file for upload
    fh.write('ID,Predicted\n')                # output header line
    for i,yi in enumerate(YeHat.ravel()):
        fh.write('{},{}\n'.format(i,yi)) # output each prediction
    fh.close()                                 # close the file
```

In [4]:
```
class dummy(ml.classifier):
    def __init__(self,X,Y,P): self.P=P; self.classes=np.unique(Y);
    def predictSoft(self,X): return self.P
```

Basic Logistic Regression

In [5]:
```
from IPython import display
fig = plt.figure()
def myPlot(lc,X,Y,Js,J0):
    display.clear_output(wait=True); plt.cla();
    plt.plot(np.arange(len(Js)),Js,np.arange(len(J0)),J0)
    plt.show()
linc = ml.linearC.linearClassify(Xt,Yt, initStep=.1, reg=1e-3, plot=myPl
Pv0 = linc.predictSoft(Xv)
Pe0 = linc.predictSoft(Xe)

toKaggle('Pv0.csv',Pv0[:,1])
toKaggle('Pe0.csv',Pe0[:,1])
print "0: Linear Regress: AUC ~",linc.auc(Xv,Yv)
```




```
0: Linear Regress: AUC ~ 0.655878721859114
```

Decision Tree based methods

Bagged Random Forest

In [6]:
```
nUse = 50
M = Xt.shape[0]
Mv= Xv.shape[0]
Pv1 = np.zeros((Xv.shape[0],2))
Pe1 = np.zeros((Xe.shape[0],2))

np.random.seed(0)
for l in range(nUse):
    Xi,Yi = ml.bootstrapData(Xt,Yt, M)          # draw this member's ra
    tree = ml.dtree.treeClassify()              #   and train the model
    tree.train(Xi,Yi,minLeaf=10,maxDepth=30,nFeatures=40)
    Pv1 += tree.predictSoft(Xv)             #   validation data
    Pe1 += tree.predictSoft(Xe)             #   test data
    if (l & (l-1) == 0): print "Train {}; est AUC {}".format(l, dummy(Xv

Pv1 /= nUse
Pe1 /= nUse

toKaggle('Pv1.csv',Pv1[:,1])
toKaggle('Pe1.csv',Pe1[:,1])
print "1: Random forest ({} members): Est AUC: {}".format(nUse,dummy(Xv,
```

```
Train 0; est AUC 0.621823710966
Train 1; est AUC 0.655132534495
Train 2; est AUC 0.669537944808
Train 4; est AUC 0.687276688453
Train 8; est AUC 0.698667392883
Train 16; est AUC 0.722240377633
Train 32; est AUC 0.737412854031
1: Random forest (50 members): Est AUC: 0.743790849673
```

Gradient Boosted Trees

In [7]:
```
nUse= 100
mu = Yt.mean()
step = 0.5
Ft = np.zeros((Xt.shape[0],)) + np.log(mu/(1-mu))
Fv = np.zeros((Xv.shape[0],)) + np.log(mu/(1-mu))
Fe = np.zeros((Xe.shape[0],)) + np.log(mu/(1-mu))
def sigma(z): return 1./(1.+np.exp(-z))

Pv2 = np.zeros((Xv.shape[0],2)); Pv2[:,0]=1-mu; Pv2[:,1]=mu;
Pe2 = np.zeros((Xe.shape[0],2)); Pe2[:,0]=1-mu; Pe2[:,1]=mu;

np.random.seed(0)
for l in range(nUse):             # this is a lot faster than the baggin
    dJ = 1.*Yt - sigma(Ft)
    tree = ml.dtree.treeRegress(Xt,dJ, maxDepth=3)  # train and save pre
    Ft  += step*tree.predict(Xt)
    Fv  += step*tree.predict(Xv)
    Fe  += step*tree.predict(Xe)

    Pv2[:,1] = sigma(Fv); Pv2[:,0] = 1-Pv2[:,1]
    Pe2[:,1] = sigma(Fe); Pe2[:,0] = 1-Pe2[:,1]
    if (l & (l-1) == 0): print "Train {}; est AUC {}".format(l, dummy(Xv
    #print " {} trees: MSE ~ {}".format(l+1, ((Yv-Pv2)**2).mean())

toKaggle('Pv2.csv',Pv2[:,1])
toKaggle('Pe2.csv',Pe2[:,1])
print "2: GradBoost, {} trees: AUC ~ {}".format(nUse, dummy(Xv,Yv,Pv2).au
```

```
Train 0; est AUC 0.693312454611
Train 1; est AUC 0.697066993464
Train 2; est AUC 0.702275780683
Train 4; est AUC 0.705679920116
Train 8; est AUC 0.730291394336
Train 16; est AUC 0.743021968046
Train 32; est AUC 0.741257262164
Train 64; est AUC 0.744903776325
2: GradBoost, 100 trees: AUC ~ 0.742964778504
```

Clustering and "local" features

Suppose that we decide our linear classifier is *underfitting*, i.e., we want to add additional complexity to the model. (NOTE: This may not be true for your data!)

To do so, let's make a collection of "local" classifiers, i.e.,:

- Cluster our data in feature space
- For each cluster, learn a "local" linear predictor

In [8]:
```
nClust = 15
ssd = np.inf

np.random.seed(0)
for it in range(2):
    Zi,mui,ssdi = ml.cluster.kmeans(Xt[:10000,:41],K=nClust,init='k++')
    if ssdi < ssd:
        print ssdi
        Z,mu,ssd = Zi,mui,ssdi
Cluster_KNN = ml.knn.knnClassify(mu,np.array(range(nClust)),1)
XtC = ml.to1ofK( Cluster_KNN.predict(Xt[:,:41]) , np.array(range(nClust)
XvC = ml.to1ofK( Cluster_KNN.predict(Xv[:,:41]) , np.array(range(nClust)
XeC = ml.to1ofK( Cluster_KNN.predict(Xe[:,:41]) , np.array(range(nClust)
print XtC.sum(0)   # show cluster sizes...
```

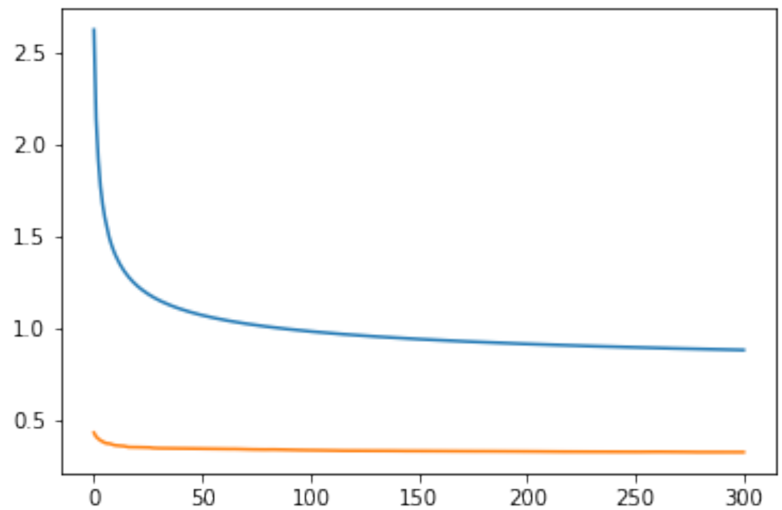
```
132714.8480207021
129868.43713290115
[ 370  253  273   26   49 1235    7   21  308 1923  640  548   54  229
    2]
```

One way to do this is to create local feature copies, which are zero if the data point is not in cluster c, and copy the original features if it is:

In [9]:
```
XtC2 = np.einsum('ij,ik->ijk',XtC,Xt).reshape((Xt.shape[0],Xt.shape[1]*n(
XvC2 = np.einsum('ij,ik->ijk',XvC,Xv).reshape((Xv.shape[0],Xv.shape[1]*n(
XeC2 = np.einsum('ij,ik->ijk',XeC,Xe).reshape((Xe.shape[0],Xe.shape[1]*n(

linc2 = ml.linearC.linearClassify(XtC2,Yt, initStep=.01, reg=1e-3, stopI1
Pv3 = linc2.predictSoft(XvC2)
Pe3 = linc2.predictSoft(XeC2)

toKaggle('Pv3.csv',Pv3[:,1])
toKaggle('Pe3.csv',Pe3[:,1])
print "3: Clustered LinClassify: AUC ~",linc2.auc(XvC2,Yv)
```




```
3: Clustered LinClassify: AUC ~ 0.6242302106027596
```

This didn't seem to work too well, but demonstrates the basic concept of a local model.

Simple Neural Network

In [10]:
```
nHidden = 500
nnet = ml.nnet.nnetClassify()
nnet.init_weights([Xt.shape[1],nHidden,2],'random',Xt,Yt);
nnet.train(Xt[:10000,:],Yt[:10000],stopTol=-1,initStep=1.,stopIter=10000(

Pv4 = nnet.predictSoft(Xv)
Pe4 = nnet.predictSoft(Xe)

toKaggle('Pv4.csv',Pv4[:,1])
toKaggle('Pe4.csv',Pe4[:,1])
print "4: Neural Net (500,1): AUC: ",nnet.auc(Xv,Yv)
```

```
it 2 : Jsur = 0.500833867903, J01 = 0.532670932974
it 4 : Jsur = 0.50218995715, J01 = 0.496968676322
it 8 : Jsur = 0.650583746627, J01 = 0.520040417649
it 16 : Jsur = 0.664054272748, J01 = 0.519703603907
it 32 : Jsur = 0.821853747772, J01 = 0.477770293028
it 64 : Jsur = 0.982262761249, J01 = 0.513640956551
it 128 : Jsur = 0.89525362095, J01 = 0.465139777703
it 256 : Jsur = 0.915684758302, J01 = 0.475581003705
it 512 : Jsur = 0.892150868516, J01 = 0.484506567868
it 1024 : Jsur = 0.767602923626, J01 = 0.469181542607
it 2048 : Jsur = 0.693209407303, J01 = 0.490232401482
it 4096 : Jsur = 0.514527044419, J01 = 0.414617716403
it 8192 : Jsur = 0.491140058489, J01 = 0.393735264399
it 16384 : Jsur = 0.443384874577, J01 = 0.352812394746
it 32768 : Jsur = 0.421344363097, J01 = 0.334792859549
it 65536 : Jsur = 0.40862254414, J01 = 0.316099696868
4: Neural Net (500,1): AUC:  0.6595352214960057
```

That doesn't seem like it worked too well (compare training err/auc to validation, and to other models), but maybe with some effort it could be improved.

Stacked

Reload the saved validation predictions and test data predictions from the previous learners:

In [19]:
```
Pv0 = np.genfromtxt('Pv0.csv',delimiter=',',skip_header=1)[:,1:2]
Pv1 = np.genfromtxt('Pv1.csv',delimiter=',',skip_header=1)[:,1:2]
Pv2 = np.genfromtxt('Pv2.csv',delimiter=',',skip_header=1)[:,1:2]
Pv3 = np.genfromtxt('Pv3.csv',delimiter=',',skip_header=1)[:,1:2]
Pv4 = np.genfromtxt('Pv4.csv',delimiter=',',skip_header=1)[:,1:2]

Pe0 = np.genfromtxt('Pe0.csv',delimiter=',',skip_header=1)[:,1:2]
Pe1 = np.genfromtxt('Pe1.csv',delimiter=',',skip_header=1)[:,1:2]
Pe2 = np.genfromtxt('Pe2.csv',delimiter=',',skip_header=1)[:,1:2]
Pe3 = np.genfromtxt('Pe3.csv',delimiter=',',skip_header=1)[:,1:2]
Pe4 = np.genfromtxt('Pe4.csv',delimiter=',',skip_header=1)[:,1:2]
```

Now, learn a simple linear combination of the models' outputs on the validation data:

In [21]:
```
Sv = np.hstack((Pv0,Pv1,Pv2,Pv3,Pv4))
stack = ml.linearC.linearClassify(Sv,Yv, reg=1e-3)
print "** Stacked AUC: ",stack.auc(Sv,Yv)

Se = np.hstack((Pe0,Pe1,Pe2,Pe3,Pe4))
PeS = stack.predictSoft(Se)
toKaggle('Ex_Stack.csv',PeS[:,1])
```

```
** Stacked AUC:  0.7516412490922295
```

So, I would estimate my AUC in the competition to be about this value.

This is very slightly optimistic, since it's estimated from the data I did the stacking with; but thats a few thousand data and only fitting 6 parameters, so it's probably not too far off. (The leaderboard reports 0.738 public and 0.749 private values for the stacked model.)

Note that, if one of your models is much better than the others, it is likely you will just end up with that model; so you would like for your stacked model to take in collections of (different) predictions that all obtain "reasonable" / approximately equivalent performance.