Project 3: Multiclass and Linear Models

UIC CS 412, Spring 2018

If you have discussed this assignment with anyone, please state their name(s) here: [NAMES]. Keep in mind the expectations set in the Academic Honesty part of the syllabus.

There are two parts to this project. The first is on multiclass reductions. The second is on linear models and gradient descent. There is also a third part which gives you an opportunity for extra credit.

This assignment is adapted from the github materials for A Course in Machine Learning (https://github.com/hal3/ciml).

Due Date

This assignment is due at 11:59pm Thursday, March 15th.

Files You'll Edit

`multiclass.py`: The multiclass classification implementation you need to complete.

`gd.py`: The gradient descent file you need to edit.

`quizbowl.py`: Multiclass evaluation of the quiz bowl dataset (optional).

`predictions.txt`: This file is automatically generated as part of Part 3 (optional).

Files you might want to look at

`binary.py`: Our generic interface for binary classifiers (actually works for regression and other types of classification, too).

`datasets.py`: Where a handful of test data sets are stored.

`util.py`: A handful of useful utility functions: these will undoubtedly be helpful to you, so take a look!

`runClassifier.py`: A few wrappers for doing useful things with classifiers, like training them, generating learning curves, etc.

`mlGraphics.py`: A few useful plotting commands

`data/*`: All of the datasets we'll use.

What to Submit

You will hand in all of the python files listed above as a single zip file **h3.zip** on Gradescope under *Homework 3*. The programming part constitutes 60% of the grade for this homework. You also need to answer the questions denoted by **WU#** (and a kitten) in this notebook which are the other 40% of your homework grade. When you are done, you should export **hw3.ipynb** with your answers as a PDF file **hw3WrittenPart.pdf**, upload the PDF file to Gradescope under *Homework 3 - Written Part*, and tag each question on Gradescope.

Your entire homework will be considered late if any of these parts are submitted late.

Autograding

Your code will be autograded for technical correctness. Please **do not** change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder. We have provided two simple test cases that you can try your code on, see `run_tests_simple.py`. As usual, you should create more test cases to make sure your code runs correctly.

Part 1: Multiclass Classification *[30% impl, 20% writeup]*

In this section, you will explore the differences between three multiclass-to-binary reductions: one-versus-all (OVA), all-versus-all (AVA), and a tree-based reduction (TREE). The evaluation will be on different datasets from `datasets.py`.

The classification task we'll work with is wine classification. The dataset was downloaded from allwines.com. Your job is to predict the type of wine, given the description of the wine. There are two tasks: WineData has 20 different wines, WineDataSmall is just the first five of those (sorted roughly by frequency). You can find the names of the wines both in WineData.labels as well as the file wines.names.

To start out, let's import everything and train decision "stumps" (aka depth=1 decision trees) on the large data set:

In [1]:
```
from sklearn.tree import DecisionTreeClassifier
import multiclass
import util
from datasets import *
import importlib

h = multiclass.OVA(20, lambda: DecisionTreeClassifier(max_depth=1))
h.train(WineData.X, WineData.Y)
P = h.predictAll(WineData.Xte)
mean(P == WineData.Yte)
# 0.29499072356215211
```

```
training classifier for 0 versus rest
training classifier for 1 versus rest
training classifier for 2 versus rest
training classifier for 3 versus rest
training classifier for 4 versus rest
training classifier for 5 versus rest
training classifier for 6 versus rest
training classifier for 7 versus rest
training classifier for 8 versus rest
training classifier for 9 versus rest
training classifier for 10 versus rest
training classifier for 11 versus rest
training classifier for 12 versus rest
training classifier for 13 versus rest
training classifier for 14 versus rest
training classifier for 15 versus rest
training classifier for 16 versus rest
training classifier for 17 versus rest
training classifier for 18 versus rest
training classifier for 19 versus rest
```

Out[1]:
```
0.2949907235621521
```

That means 29% accuracy on this task. The most frequent class is:

In [10]:
```
print(mode(WineData.Y))
# 1
print(WineData.labels[1])
# Cabernet-Sauvignon
```

```
1
Cabernet-Sauvignon
```

And if you were to always predict label 1, you would get the following accuracy:

In [11]:
```
mean(WineData.Yte == 1)
# 0.17254174397031541
```

Out[11]:
```
0.1725417439703154
```

So we're doing a bit (12%) better than that using decision stumps.

The default implementation of OVA uses decision tree confidence (probability of prediction) to weigh the votes. You can switch to zero/one predictions to see the effect:

In [12]:
```
P = h.predictAll(WineData.Xte, useZeroOne=True)
mean(P == WineData.Yte)
# 0.19109461966604824
```

Out[12]:
```
0.19109461966604824
```

As you can see, this is markedly worse.

Switching to the smaller data set for a minute, we can train, say, depth 3 decision trees:

In [13]:
```
h = multiclass.OVA(5, lambda: DecisionTreeClassifier(max_depth=3))
h.train(WineDataSmall.X, WineDataSmall.Y)
P = h.predictAll(WineDataSmall.Xte)
print(mean(P == WineDataSmall.Yte))
# 0.590809628009
print(mean(WineDataSmall.Yte == 1))
# 0.407002188184
```

```
training classifier for 0 versus rest
training classifier for 1 versus rest
training classifier for 2 versus rest
training classifier for 3 versus rest
training classifier for 4 versus rest
0.5908096280087527
0.40700218818380746
```

So using depth 3 trees we get an accuracy of about 60% (this number varies a bit), versus a baseline of 41%. That's not too terrible, but not great.

We can look at what this classifier is doing.

In [4]:
```
print(WineDataSmall.labels[0])
#'Sauvignon-Blanc'
util.showTree(h.f[0], WineDataSmall.words)
#citrus?
#-N-> lime?
#|    -N-> gooseberry?
#|    |    -N-> class 0 (356.0 for class 0, 10.0 for class 1)
#|    |    -Y-> class 1 (0.0 for class 0, 4.0 for class 1)
#|    -Y-> apple?
#|    |    -N-> class 1 (1.0 for class 0, 15.0 for class 1)
#|    |    -Y-> class 0 (2.0 for class 0, 0.0 for class 1)
#-Y-> grapefruit?
#|    -N-> flavors?
#|    |    -N-> class 1 (4.0 for class 0, 12.0 for class 1)
#|    |    -Y-> class 0 (11.0 for class 0, 5.0 for class 1)
#|    -Y-> opens?
#|    |    -N-> class 1 (0.0 for class 0, 14.0 for class 1)
#|    |    -Y-> class 0 (1.0 for class 0, 0.0 for class 1)
```

```
Sauvignon-Blanc
citrus?
-N-> lime?
|    -N-> gooseberry?
|    |    -N-> class 0  (356.0 for class 0, 10.0 for class 1)
|    |    -Y-> class 1  (0.0 for class 0, 4.0 for class 1)
|    -Y-> hint?
|    |    -N-> class 1  (1.0 for class 0, 15.0 for class 1)
|    |    -Y-> class 0  (2.0 for class 0, 0.0 for class 1)
-Y-> grapefruit?
|    -N-> flavors?
|    |    -N-> class 1  (4.0 for class 0, 12.0 for class 1)
|    |    -Y-> class 0  (11.0 for class 0, 5.0 for class 1)
|    -Y-> extremely?
|    |    -N-> class 1  (0.0 for class 0, 14.0 for class 1)
|    |    -Y-> class 0  (1.0 for class 0, 0.0 for class 1)
```

This should show the tree that's associated with predicting label 0 (which is stored in h.f[0]). The 1s mean "likely to be Sauvignon-Blanc" and the 0s mean "likely not to be".

Now, go in and complete the AVA implementation in `multiclass.py`. You should be able to train an AVA model on the small data set by:

In [2]:
```
h = multiclass.AVA(5, lambda: DecisionTreeClassifier(max_depth=3))
h.train(WineDataSmall.X, WineDataSmall.Y)
P = h.predictAll(WineDataSmall.Xte)
print(mean(WineDataSmall.Yte == 1))
```

```
training classifier for 1 versus 0
training classifier for 2 versus 0
training classifier for 2 versus 1
training classifier for 3 versus 0
training classifier for 3 versus 1
training classifier for 3 versus 2
training classifier for 4 versus 0
training classifier for 4 versus 1
training classifier for 4 versus 2
training classifier for 4 versus 3
0.40700218818380746
```

Next, you must implement a tree-based reduction in `multiclass.py`. Most of train is given to you, but predict you must do all on your own. There is a tree class to help you:

In [24]:
```
t = multiclass.makeBalancedTree(range(5))
print(t)
# [[0 1]] [2 [3 4]]]
print(t.isLeaf)
# False
print(t.getRight())
# [2 [3 4]]
print(t.getRight().getLeft())
# 2
print(t.getRight().getLeft().isLeaf)
# True
```

```
[[0 1] [2 [3 4]]]
False
[2 [3 4]]
2
True
```

You should be able to train a MCTree model by:

In [16]:
```
h = multiclass.MCTree(t, lambda: DecisionTreeClassifier(max_depth=3))
h.train(WineDataSmall.X, WineDataSmall.Y)
P = h.predictAll(WineDataSmall.Xte)
```

```
training classifier for [0, 1] versus [2, 3, 4]
training classifier for [0] versus [1]
training classifier for [2] versus [3, 4]
training classifier for [3] versus [4]
```

WU1 (10%):

Answer A, B, C for both OVA and AVA.

(A) What words are most indicative of being Sauvignon-Blanc? Which words are most indicative of not being Sauvignon-Blanc? What about Pinot-Noir (label==2)?

(B) Train depth 3 decision trees on the full WineData task (with 20 labels). What accuracy do you get? How long does this take (in seconds)? One of my least favorite wines is Viognier -- what words are indicative of this?

(C) Compare the accuracy using zero-one predictions versus using confidence. How much difference does it make?

In [2]:
```
# from datetime import datetime

import multiclass
# WU1 CODE HERE
#A
#For OVA
h = multiclass.OVA(5, lambda: DecisionTreeClassifier(max_depth=3))
h.train(WineDataSmall.X, WineDataSmall.Y)
P = h.predictAll(WineDataSmall.Xte)
util.showTree(h.f[0], WineDataSmall.words)
#words indicative of Suvignon-blanc: lime,citrus,apple
#words not indicative of Suvignon-blanc:Flavors,grapefruit,gooseberry

#For AVA
h = multiclass.AVA(5, lambda: DecisionTreeClassifier(max_depth=3))
h.train(WineDataSmall.X, WineDataSmall.Y)
P = h.predictAll(WineDataSmall.Xte)
#words indicative of Suvignon-blanc: cassis
#words not indicative of Suvignon-blanc:
#util.showTree(h.f[0][2], WineDataSmall.words)

#'Pinot-Noir '

#For OVA using given above function:

h = multiclass.OVA(5, lambda: DecisionTreeClassifier(max_depth=3))
h.train(WineData.X, WineData.Y)
P = h.predictAll(WineData.Xte)
util.showTree(h.f[2], WineDataSmall.words)
#Hence,words indicative of Pinot-Noir  are:carneros,tea
#and words not indicative of Pinot-Noir are:purple,cassis



#'Pinot-Noir ' for ava
#increasing depth to 5 to get more clarity as to which are like Pinot
Noir
h = multiclass.AVA(5, lambda: DecisionTreeClassifier(max_depth=5))
h.train(WineDataSmall.X, WineDataSmall.Y)
P = h.predictAll(WineDataSmall.Xte)

util.showTree(h.f[2][1], WineDataSmall.words)

#words indicative of Pinot-Noir  are:cassis,plum,smooth,aromas,chocola
te,tobacco
#words not indicative of Pinot-Noir are:salmon,100,acidity,pleasing,pu
re,duck,acidity
```

```
training classifier for 0 versus rest
```

```
training classifier for 1 versus rest
training classifier for 2 versus rest
training classifier for 3 versus rest
training classifier for 4 versus rest
citrus?
-N-> lime?
|    -N-> gooseberry?
|    |    -N-> class 0  (356.0 for class 0, 10.0 for class 1)
|    |    -Y-> class 1  (0.0 for class 0, 4.0 for class 1)
|    -Y-> apple?
|    |    -N-> class 1  (1.0 for class 0, 15.0 for class 1)
|    |    -Y-> class 0  (2.0 for class 0, 0.0 for class 1)
-Y-> grapefruit?
|    -N-> flavors?
|    |    -N-> class 1  (4.0 for class 0, 12.0 for class 1)
|    |    -Y-> class 0  (11.0 for class 0, 5.0 for class 1)
|    -Y-> 2010?
|    |    -N-> class 1  (0.0 for class 0, 14.0 for class 1)
|    |    -Y-> class 0  (1.0 for class 0, 0.0 for class 1)
training classifier for 1 versus 0
training classifier for 2 versus 0
training classifier for 2 versus 1
training classifier for 3 versus 0
training classifier for 3 versus 1
training classifier for 3 versus 2
training classifier for 4 versus 0
training classifier for 4 versus 1
training classifier for 4 versus 2
training classifier for 4 versus 3
training classifier for 0 versus rest
training classifier for 1 versus rest
training classifier for 2 versus rest
training classifier for 3 versus rest
training classifier for 4 versus rest
cherry?
-N-> cherries?
|    -N-> mushroom?
|    |    -N-> class 0  (693.0 for class 0, 44.0 for class 1)
|    |    -Y-> class 0  (5.0 for class 0, 4.0 for class 1)
|    -Y-> tea?
|    |    -N-> class 0  (59.0 for class 0, 21.0 for class 1)
|    |    -Y-> class 1  (1.0 for class 0, 5.0 for class 1)
-Y-> nutmeg?
|    -N-> underlying?
|    |    -N-> class 0  (174.0 for class 0, 56.0 for class 1)
|    |    -Y-> class 1  (0.0 for class 0, 5.0 for class 1)
|    -Y-> complemented?
|    |    -N-> class 1  (0.0 for class 0, 9.0 for class 1)
|    |    -Y-> class 0  (1.0 for class 0, 0.0 for class 1)
training classifier for 1 versus 0
```

```
training classifier for 2 versus 0
training classifier for 2 versus 1
training classifier for 3 versus 0
training classifier for 3 versus 1
training classifier for 3 versus 2
training classifier for 4 versus 0
training classifier for 4 versus 1
training classifier for 4 versus 2
training classifier for 4 versus 3
cassis?
-N-> acidity?
|    -N-> salmon?
|    |    -N-> chocolate?
|    |    |    -N-> 10?
|    |    |    |    -N-> class 0        (87.0 for class 0, 87.0 for
class 1)
|    |    |    |    -Y-> class 1        (0.0 for class 0, 11.0 for c
lass 1)
|    |    |    -Y-> supple?
|    |    |    |    -N-> class 1        (2.0 for class 0, 31.0 for c
lass 1)
|    |    |    |    -Y-> class 0        (3.0 for class 0, 0.0 for cl
ass 1)
|    |    -Y-> class 0  (11.0 for class 0, 0.0 for class 1)
|    -Y-> tannins?
|    |    -N-> class 0  (22.0 for class 0, 0.0 for class 1)
|    |    -Y-> aromas?
|    |    |    -N-> pure?
|    |    |    |    -N-> class 1        (1.0 for class 0, 8.0 for cl
ass 1)
|    |    |    |    -Y-> class 0        (2.0 for class 0, 0.0 for cl
ass 1)
|    |    |    -Y-> plum?
|    |    |    |    -N-> class 0        (12.0 for class 0, 1.0 for c
lass 1)
|    |    |    |    -Y-> class 1        (0.0 for class 0, 2.0 for cl
ass 1)
-Y-> tea?
|    -N-> 100?
|    |    -N-> pleasing?
|    |    |    -N-> class 1     (0.0 for class 0, 46.0 for class 1)
|    |    |    -Y-> raspberry?
|    |    |    |    -N-> class 0        (1.0 for class 0, 0.0 for cl
ass 1)
|    |    |    |    -Y-> class 1        (0.0 for class 0, 1.0 for cl
ass 1)
|    |    -Y-> class 0  (1.0 for class 0, 0.0 for class 1)
|    -Y-> class 0       (2.0 for class 0, 0.0 for class 1)
```

In [64]:
```
from datetime import datetime


#B
#'Viognier '
#for ova using label Viogner
start_time = datetime.now()
h = multiclass.OVA(20, lambda: DecisionTreeClassifier(max_depth=3))
h.train(WineData.X, WineData.Y)
P = h.predictAll(WineData.Xte)
util.showTree(h.f[17], WineData.words)
dt = datetime.now() - start_time
ms = (dt.days * 24 * 60 * 60 + dt.seconds) * 1000 + dt.microseconds /
1000.0
print('Time elapsed for OVA for Viognier with 20 labels is = {0} ms'.f
ormat(ms))
#words indicative of Viognier are: Pineapple,Tannin,cinnamon,candied,m
ilk

#for AVA and using label Viogner

start_time = datetime.now()
h = multiclass.AVA(20, lambda: DecisionTreeClassifier(max_depth=5))
h.train(WineData.X, WineData.Y)
P = h.predictAll(WineData.Xte)

dt = datetime.now() - start_time
ms = (dt.days * 24 * 60 * 60 + dt.seconds) * 1000 + dt.microseconds /
1000.0
print('Time elapsed for AVA for Viognier with 20 labels is = {0} ms'.f
ormat(ms))
util.showTree(h.f[17][1], WineData.words)

#words indicative of Viognier are: Pear,peaches,peach
```

```
training classifier for 0 versus rest
training classifier for 1 versus rest
training classifier for 2 versus rest
training classifier for 3 versus rest
training classifier for 4 versus rest
training classifier for 5 versus rest
training classifier for 6 versus rest
training classifier for 7 versus rest
training classifier for 8 versus rest
training classifier for 9 versus rest
training classifier for 10 versus rest
training classifier for 11 versus rest
training classifier for 12 versus rest
training classifier for 13 versus rest
```

```
training classifier for 14 versus rest
training classifier for 15 versus rest
training classifier for 16 versus rest
training classifier for 17 versus rest
training classifier for 18 versus rest
training classifier for 19 versus rest
peaches?
-N-> nectarine?
|    -N-> chilled?
|    |    -N-> class 0  (1036.0 for class 0, 1.0 for class 1)
|    |    -Y-> class 0  (6.0 for class 0, 1.0 for class 1)
|    -Y-> lychee?
|    |    -N-> class 0  (13.0 for class 0, 1.0 for class 1)
|    |    -Y-> class 1  (0.0 for class 0, 1.0 for class 1)
-Y-> milk?
|    -N-> fully?
|    |    -N-> class 0  (14.0 for class 0, 0.0 for class 1)
|    |    -Y-> class 1  (0.0 for class 0, 1.0 for class 1)
|    -Y-> class 1       (0.0 for class 0, 3.0 for class 1)
Time elapsed for OVA for Viognier with 20 labels is = 1375.232 ms
training classifier for 1 versus 0
training classifier for 2 versus 0
training classifier for 2 versus 1
training classifier for 3 versus 0
training classifier for 3 versus 1
training classifier for 3 versus 2
training classifier for 4 versus 0
training classifier for 4 versus 1
training classifier for 4 versus 2
training classifier for 4 versus 3
training classifier for 5 versus 0
training classifier for 5 versus 1
training classifier for 5 versus 2
training classifier for 5 versus 3
training classifier for 5 versus 4
training classifier for 6 versus 0
training classifier for 6 versus 1
training classifier for 6 versus 2
training classifier for 6 versus 3
training classifier for 6 versus 4
training classifier for 6 versus 5
training classifier for 7 versus 0
training classifier for 7 versus 1
training classifier for 7 versus 2
training classifier for 7 versus 3
training classifier for 7 versus 4
training classifier for 7 versus 5
training classifier for 7 versus 6
training classifier for 8 versus 0
training classifier for 8 versus 1
```

```
training classifier for 8 versus 2
training classifier for 8 versus 3
training classifier for 8 versus 4
training classifier for 8 versus 5
training classifier for 8 versus 6
training classifier for 8 versus 7
training classifier for 9 versus 0
training classifier for 9 versus 1
training classifier for 9 versus 2
training classifier for 9 versus 3
training classifier for 9 versus 4
training classifier for 9 versus 5
training classifier for 9 versus 6
training classifier for 9 versus 7
training classifier for 9 versus 8
training classifier for 10 versus 0
training classifier for 10 versus 1
training classifier for 10 versus 2
training classifier for 10 versus 3
training classifier for 10 versus 4
training classifier for 10 versus 5
training classifier for 10 versus 6
training classifier for 10 versus 7
training classifier for 10 versus 8
training classifier for 10 versus 9
training classifier for 11 versus 0
training classifier for 11 versus 1
training classifier for 11 versus 2
training classifier for 11 versus 3
training classifier for 11 versus 4
training classifier for 11 versus 5
training classifier for 11 versus 6
training classifier for 11 versus 7
training classifier for 11 versus 8
training classifier for 11 versus 9
training classifier for 11 versus 10
training classifier for 12 versus 0
training classifier for 12 versus 1
training classifier for 12 versus 2
training classifier for 12 versus 3
training classifier for 12 versus 4
training classifier for 12 versus 5
training classifier for 12 versus 6
training classifier for 12 versus 7
training classifier for 12 versus 8
training classifier for 12 versus 9
training classifier for 12 versus 10
training classifier for 12 versus 11
training classifier for 13 versus 0
training classifier for 13 versus 1
```

```
training classifier for 13 versus 2
training classifier for 13 versus 3
training classifier for 13 versus 4
training classifier for 13 versus 5
training classifier for 13 versus 6
training classifier for 13 versus 7
training classifier for 13 versus 8
training classifier for 13 versus 9
training classifier for 13 versus 10
training classifier for 13 versus 11
training classifier for 13 versus 12
training classifier for 14 versus 0
training classifier for 14 versus 1
training classifier for 14 versus 2
training classifier for 14 versus 3
training classifier for 14 versus 4
training classifier for 14 versus 5
training classifier for 14 versus 6
training classifier for 14 versus 7
training classifier for 14 versus 8
training classifier for 14 versus 9
training classifier for 14 versus 10
training classifier for 14 versus 11
training classifier for 14 versus 12
training classifier for 14 versus 13
training classifier for 15 versus 0
training classifier for 15 versus 1
training classifier for 15 versus 2
training classifier for 15 versus 3
training classifier for 15 versus 4
training classifier for 15 versus 5
training classifier for 15 versus 6
training classifier for 15 versus 7
training classifier for 15 versus 8
training classifier for 15 versus 9
training classifier for 15 versus 10
training classifier for 15 versus 11
training classifier for 15 versus 12
training classifier for 15 versus 13
training classifier for 15 versus 14
training classifier for 16 versus 0
training classifier for 16 versus 1
training classifier for 16 versus 2
training classifier for 16 versus 3
training classifier for 16 versus 4
training classifier for 16 versus 5
training classifier for 16 versus 6
training classifier for 16 versus 7
training classifier for 16 versus 8
training classifier for 16 versus 9
```

```
training classifier for 16 versus 10
training classifier for 16 versus 11
training classifier for 16 versus 12
training classifier for 16 versus 13
training classifier for 16 versus 14
training classifier for 16 versus 15
training classifier for 17 versus 0
training classifier for 17 versus 1
training classifier for 17 versus 2
training classifier for 17 versus 3
training classifier for 17 versus 4
training classifier for 17 versus 5
training classifier for 17 versus 6
training classifier for 17 versus 7
training classifier for 17 versus 8
training classifier for 17 versus 9
training classifier for 17 versus 10
training classifier for 17 versus 11
training classifier for 17 versus 12
training classifier for 17 versus 13
training classifier for 17 versus 14
training classifier for 17 versus 15
training classifier for 17 versus 16
training classifier for 18 versus 0
training classifier for 18 versus 1
training classifier for 18 versus 2
training classifier for 18 versus 3
training classifier for 18 versus 4
training classifier for 18 versus 5
training classifier for 18 versus 6
training classifier for 18 versus 7
training classifier for 18 versus 8
training classifier for 18 versus 9
training classifier for 18 versus 10
training classifier for 18 versus 11
training classifier for 18 versus 12
training classifier for 18 versus 13
training classifier for 18 versus 14
training classifier for 18 versus 15
training classifier for 18 versus 16
training classifier for 18 versus 17
training classifier for 19 versus 0
training classifier for 19 versus 1
training classifier for 19 versus 2
training classifier for 19 versus 3
training classifier for 19 versus 4
training classifier for 19 versus 5
training classifier for 19 versus 6
training classifier for 19 versus 7
training classifier for 19 versus 8
```

```
training classifier for 19 versus 9
training classifier for 19 versus 10
training classifier for 19 versus 11
training classifier for 19 versus 12
training classifier for 19 versus 13
training classifier for 19 versus 14
training classifier for 19 versus 15
training classifier for 19 versus 16
training classifier for 19 versus 17
training classifier for 19 versus 18
Time elapsed for AVA for Viognier with 20 labels is = 10654.155 ms
peach?
-N-> peaches?
|    -N-> pear?
|    |    -N-> class 1  (0.0 for class 0, 187.0 for class 1)
|    |    -Y-> class 0  (1.0 for class 0, 0.0 for class 1)
|    -Y-> class 0       (3.0 for class 0, 0.0 for class 1)
-Y-> class 0    (4.0 for class 0, 0.0 for class 1)
```

In [26]:
```
#C
#FOR OVA:

#USING CONFIDENCE
h = multiclass.OVA(20, lambda: DecisionTreeClassifier(max_depth=3))
h.train(WineData.X, WineData.Y)
P = h.predictAll(WineData.Xte)
print("FOr OVA confidence:")
print(mean(P == WineData.Yte))
#For OVA probability of prediction:


#USING ZERONE
P = h.predictAll(WineData.Xte, useZeroOne=True)
print("FOr OVA zeroOne confidence is:")
print(mean(P == WineData.Yte))

#for OVA the confidence is:

#For AVA:
#USING CONFIDENCE
start_time = datetime.now()
h = multiclass.AVA(20, lambda: DecisionTreeClassifier(max_depth=3))
h.train(WineData.X, WineData.Y)
P = h.predictAll(WineData.Xte)
dt = datetime.now() - start_time
ms = (dt.days * 24 * 60 * 60 + dt.seconds) * 1000 + dt.microseconds /
1000.0
print('Time elapsed for AVA  with 20 labels is = {0} ms'.format(ms))
print("FOr AVA confidence:")
print(mean(P == WineData.Yte))

#For AVA with 20 labels Confidence is:

#USING ZERONE
P = h.predictAll(WineData.Xte, useZeroOne=True)
print("FOr AVA zeroONe confidence:")
print(mean(P == WineData.Yte))
#ZeroOne for AVA accuracy:
```

```
training classifier for 0 versus rest
training classifier for 1 versus rest
training classifier for 2 versus rest
training classifier for 3 versus rest
training classifier for 4 versus rest
training classifier for 5 versus rest
training classifier for 6 versus rest
training classifier for 7 versus rest
```

```
training classifier for 8 versus rest
training classifier for 9 versus rest
training classifier for 10 versus rest
training classifier for 11 versus rest
training classifier for 12 versus rest
training classifier for 13 versus rest
training classifier for 14 versus rest
training classifier for 15 versus rest
training classifier for 16 versus rest
training classifier for 17 versus rest
training classifier for 18 versus rest
training classifier for 19 versus rest
FOr OVA confidence:
0.3738404452690167
FOr OVA zeroOne confidence is:
0.24860853432282004
training classifier for 1 versus 0
training classifier for 2 versus 0
training classifier for 2 versus 1
training classifier for 3 versus 0
training classifier for 3 versus 1
training classifier for 3 versus 2
training classifier for 4 versus 0
training classifier for 4 versus 1
training classifier for 4 versus 2
training classifier for 4 versus 3
training classifier for 5 versus 0
training classifier for 5 versus 1
training classifier for 5 versus 2
training classifier for 5 versus 3
training classifier for 5 versus 4
training classifier for 6 versus 0
training classifier for 6 versus 1
training classifier for 6 versus 2
training classifier for 6 versus 3
training classifier for 6 versus 4
training classifier for 6 versus 5
training classifier for 7 versus 0
training classifier for 7 versus 1
training classifier for 7 versus 2
training classifier for 7 versus 3
training classifier for 7 versus 4
training classifier for 7 versus 5
training classifier for 7 versus 6
training classifier for 8 versus 0
training classifier for 8 versus 1
training classifier for 8 versus 2
training classifier for 8 versus 3
training classifier for 8 versus 4
training classifier for 8 versus 5
```

```
training classifier for 8 versus 6
training classifier for 8 versus 7
training classifier for 9 versus 0
training classifier for 9 versus 1
training classifier for 9 versus 2
training classifier for 9 versus 3
training classifier for 9 versus 4
training classifier for 9 versus 5
training classifier for 9 versus 6
training classifier for 9 versus 7
training classifier for 9 versus 8
training classifier for 10 versus 0
training classifier for 10 versus 1
training classifier for 10 versus 2
training classifier for 10 versus 3
training classifier for 10 versus 4
training classifier for 10 versus 5
training classifier for 10 versus 6
training classifier for 10 versus 7
training classifier for 10 versus 8
training classifier for 10 versus 9
training classifier for 11 versus 0
training classifier for 11 versus 1
training classifier for 11 versus 2
training classifier for 11 versus 3
training classifier for 11 versus 4
training classifier for 11 versus 5
training classifier for 11 versus 6
training classifier for 11 versus 7
training classifier for 11 versus 8
training classifier for 11 versus 9
training classifier for 11 versus 10
training classifier for 12 versus 0
training classifier for 12 versus 1
training classifier for 12 versus 2
training classifier for 12 versus 3
training classifier for 12 versus 4
training classifier for 12 versus 5
training classifier for 12 versus 6
training classifier for 12 versus 7
training classifier for 12 versus 8
training classifier for 12 versus 9
training classifier for 12 versus 10
training classifier for 12 versus 11
training classifier for 13 versus 0
training classifier for 13 versus 1
training classifier for 13 versus 2
training classifier for 13 versus 3
training classifier for 13 versus 4
training classifier for 13 versus 5
```

```
training classifier for 13 versus 6
training classifier for 13 versus 7
training classifier for 13 versus 8
training classifier for 13 versus 9
training classifier for 13 versus 10
training classifier for 13 versus 11
training classifier for 13 versus 12
training classifier for 14 versus 0
training classifier for 14 versus 1
training classifier for 14 versus 2
training classifier for 14 versus 3
training classifier for 14 versus 4
training classifier for 14 versus 5
training classifier for 14 versus 6
training classifier for 14 versus 7
training classifier for 14 versus 8
training classifier for 14 versus 9
training classifier for 14 versus 10
training classifier for 14 versus 11
training classifier for 14 versus 12
training classifier for 14 versus 13
training classifier for 15 versus 0
training classifier for 15 versus 1
training classifier for 15 versus 2
training classifier for 15 versus 3
training classifier for 15 versus 4
training classifier for 15 versus 5
training classifier for 15 versus 6
training classifier for 15 versus 7
training classifier for 15 versus 8
training classifier for 15 versus 9
training classifier for 15 versus 10
training classifier for 15 versus 11
training classifier for 15 versus 12
training classifier for 15 versus 13
training classifier for 15 versus 14
training classifier for 16 versus 0
training classifier for 16 versus 1
training classifier for 16 versus 2
training classifier for 16 versus 3
training classifier for 16 versus 4
training classifier for 16 versus 5
training classifier for 16 versus 6
training classifier for 16 versus 7
training classifier for 16 versus 8
training classifier for 16 versus 9
training classifier for 16 versus 10
training classifier for 16 versus 11
training classifier for 16 versus 12
training classifier for 16 versus 13
```

```
training classifier for 16 versus 14
training classifier for 16 versus 15
training classifier for 17 versus 0
training classifier for 17 versus 1
training classifier for 17 versus 2
training classifier for 17 versus 3
training classifier for 17 versus 4
training classifier for 17 versus 5
training classifier for 17 versus 6
training classifier for 17 versus 7
training classifier for 17 versus 8
training classifier for 17 versus 9
training classifier for 17 versus 10
training classifier for 17 versus 11
training classifier for 17 versus 12
training classifier for 17 versus 13
training classifier for 17 versus 14
training classifier for 17 versus 15
training classifier for 17 versus 16
training classifier for 18 versus 0
training classifier for 18 versus 1
training classifier for 18 versus 2
training classifier for 18 versus 3
training classifier for 18 versus 4
training classifier for 18 versus 5
training classifier for 18 versus 6
training classifier for 18 versus 7
training classifier for 18 versus 8
training classifier for 18 versus 9
training classifier for 18 versus 10
training classifier for 18 versus 11
training classifier for 18 versus 12
training classifier for 18 versus 13
training classifier for 18 versus 14
training classifier for 18 versus 15
training classifier for 18 versus 16
training classifier for 18 versus 17
training classifier for 19 versus 0
training classifier for 19 versus 1
training classifier for 19 versus 2
training classifier for 19 versus 3
training classifier for 19 versus 4
training classifier for 19 versus 5
training classifier for 19 versus 6
training classifier for 19 versus 7
training classifier for 19 versus 8
training classifier for 19 versus 9
training classifier for 19 versus 10
training classifier for 19 versus 11
training classifier for 19 versus 12
```

```
training classifier for 19 versus 13
training classifier for 19 versus 14
training classifier for 19 versus 15
training classifier for 19 versus 16
training classifier for 19 versus 17
training classifier for 19 versus 18
Time elapsed for AVA  with 20 labels is = 9775.177 ms
FOr AVA confidence:
0.06029684601113173
FOr AVA zeroONe confidence:
0.1725417439703154
```

[WU1 ANSWER HERE]

WU2 (10%):

Using decision trees of constant depth for each classifier (but you choose it as well as you can!), train AVA, OVA and Tree (using balanced trees) for the wine data. Which does best and why?

In [26]:
```
# WU2 CODE HERE
h = multiclass.AVA(5, lambda: DecisionTreeClassifier(max_depth=2))
h.train(WineData.X, WineData.Y)
P = h.predictAll(WineData.Xte)
print(mean(P == WineData.Yte))

h = multiclass.OVA(5, lambda: DecisionTreeClassifier(max_depth=2))
h.train(WineData.X, WineData.Y)
P = h.predictAll(WineData.Xte)
print(mean(P == WineData.Yte))

h = multiclass.MCTree(t, lambda: DecisionTreeClassifier(max_depth=2))
h.train(WineData.X, WineData.Y)
P = h.predictAll(WineData.Xte)
print(mean(P != WineData.Yte))



#Best is OVA because the less the classification is less so accuracy i
s better.OVA has less number of classifier and
#less error as we train with less number of classifier.In AVA there ar
e more classsifier as we compare with all
# n with all n.In MCTRee we balanced tree so more balanced tree will h
ave more error at each depth.Each classification
#compararision are O(log(k)).Every level has more average e as level i
ncreases.
```

```
training classifier for 1 versus 0
training classifier for 2 versus 0
training classifier for 2 versus 1
training classifier for 3 versus 0
training classifier for 3 versus 1
training classifier for 3 versus 2
training classifier for 4 versus 0
training classifier for 4 versus 1
training classifier for 4 versus 2
training classifier for 4 versus 3
0.11224489795918367
training classifier for 0 versus rest
training classifier for 1 versus rest
training classifier for 2 versus rest
training classifier for 3 versus rest
training classifier for 4 versus rest
0.23654916512059368
training classifier for [0, 1] versus [2, 3, 4]
training classifier for [0] versus [1]
training classifier for [2] versus [3, 4]
training classifier for [3] versus [4]
0.764378478664193
```

[WU2 ANSWER HERE]

WU-EC1 ExtraCredit (10%):

Build a better tree (any way you want) other than the balanced binary tree. Fill in your code for this in `getMyTreeForWine`, which defaults to a balanced tree. It should get at least 5% lower absolute error to get the extra credit. Describe what you did.

[YOUR WU-EC1 ANSWER HERE]

In [46]:
```
t = multiclass.getMyTreeForWine(range(19))
h = multiclass.MCTree(t, lambda: DecisionTreeClassifier(max_depth=2))
h.train(WineData.X, WineData.Y)
P = h.predictAll(WineData.Xte)
print(mean(P != WineData.Yte))


#made the split more informative in the getMyTreeForWine method
#so when the split is more infomative we classify the points to get most information gain
#improving my split intelligence helped in improving the absolute error and helps features classify correctly
#so here splitting randomly in 4 for the WineData make samples easily classified hence can be labelled using the split
#feature which gives most informative split(we can also use train_test_split)
```

```
training classifier for [0, 1, 2, 3] versus [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]
training classifier for [0, 1] versus [2, 3]
training classifier for [0] versus [1]
training classifier for [2] versus [3]
training classifier for [4, 5, 6, 7, 8, 9, 10] versus [11, 12, 13, 14, 15, 16, 17, 18]
training classifier for [4, 5, 6] versus [7, 8, 9, 10]
training classifier for [4] versus [5, 6]
training classifier for [5] versus [6]
training classifier for [7, 8] versus [9, 10]
training classifier for [7] versus [8]
training classifier for [9] versus [10]
training classifier for [11, 12, 13, 14] versus [15, 16, 17, 18]
training classifier for [11, 12] versus [13, 14]
training classifier for [11] versus [12]
training classifier for [13] versus [14]
training classifier for [15, 16] versus [17, 18]
training classifier for [15] versus [16]
training classifier for [17] versus [18]
0.7087198515769945
```

Part 2: Gradient Descent and Linear Classification *[30% impl, 20% writeup]*

To get started with linear models, we will implement a generic gradient descent method. This should go in `gd.py`, which contains a single (short) function: `gd`. This takes five parameters: the function we're optimizing, it's gradient, an initial position, a number of iterations to run, and an initial step size.

In each iteration of gradient descent, we will compute the gradient and take a step in that direction, with step size `eta`. We will have an *adaptive* step size, where `eta` is computed as `stepSize` divided by the square root of the iteration number (counting from one).

Once you have an implementation running, we can check it on a simple example of minimizing the function `x^2`:

In [4]:
```
import gd
gd.gd(lambda x: x**2, lambda x: 2*x, 10, 10, 0.2)
#(1.0034641051795872, array([ 100.        ,   36.        ,   18.515324
7 ,   10.95094653,
#          7.00860578,    4.72540613,    3.30810578,    2.38344246,
#          1.75697198,    1.31968118,    1.00694021]))
```

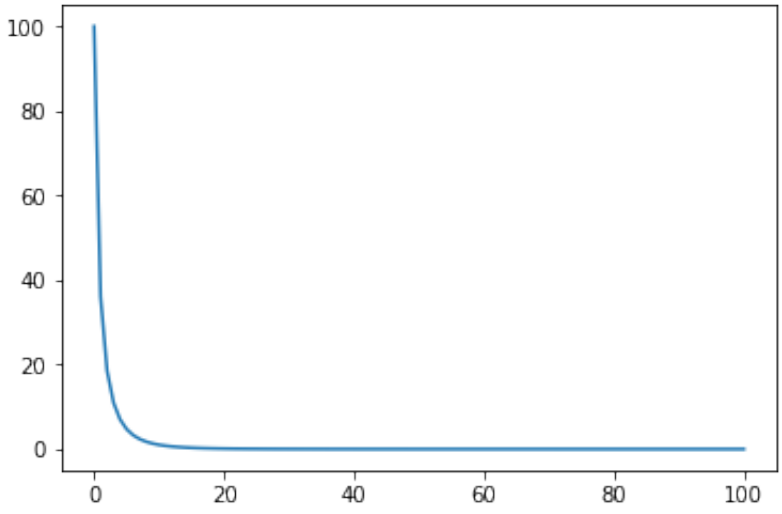
Out[4]:
```
(1.0034641051795872,
 array([100.        ,  36.        ,  18.5153247 ,  10.95094653,
          7.00860578,   4.72540613,   3.30810578,   2.38344246,
          1.75697198,   1.31968118,   1.00694021]))
```

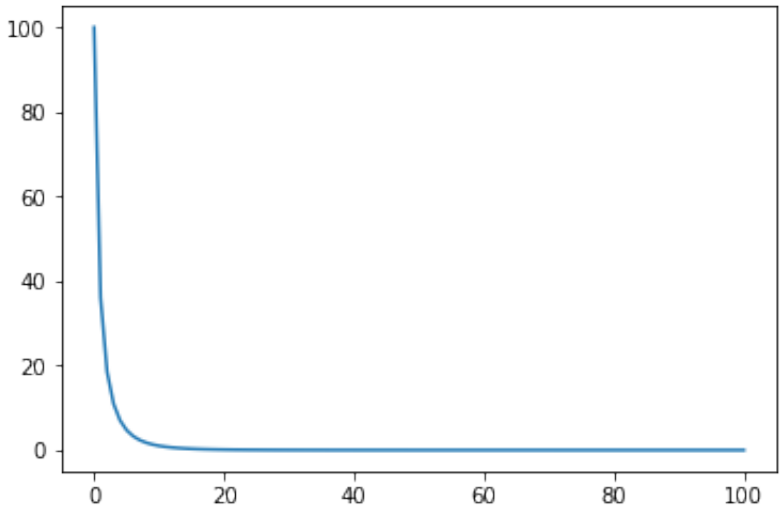
You can see that the "solution" found is about 1, which is not great (it should be zero!), but it's better than the initial value of ten! If yours is going up rather than going down, you probably have a sign error somewhere!

We can let it run longer and plot the trajectory:

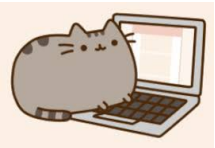
In [70]:
```
from matplotlib.pyplot import *
#def gd(func, grad, x0, numIter, stepSize)
x, trajectory = gd.gd(lambda x: x**2, lambda x: 2*x, 10, 100, 0.2)
print(x)
# 0.003645900464603937
plot(trajectory)
show(False)
```

```
0.003645900464603937
```





It's now found a value close to zero and you can see that the objective is decreasing by looking at the plot.



WU3 (5%):

Find a few values of step size where it converges and a few values where it diverges. Where does the threshold seem to be?

[Your WU3 answer here]

In [7]:
```
#WU3 code here
import numpy as np
#converges at  0.25
x, trajectory = gd.gd(lambda x: np.linalg.norm(x)**2, lambda x: 2*x, n
p.array([10,5]), 100, 0.25)
print(x)


#diverges at 7.7
x, trajectory = gd.gd(lambda x: np.linalg.norm(x)**2, lambda x: 2*x, n
p.array([10,5]), 100,7.7)
print(x)




#threshold at 6.7
x, trajectory = gd.gd(lambda x: np.linalg.norm(x)**2, lambda x: 2*x, n
p.array([10,5]), 100, 6.7)
print(x)
```

```
[0.00041783 0.00020892]
[2.36728875e+14 1.18364438e+14]
[81.61748427 40.80874214]
```

WU4 (10%):

Come up with a *non-convex* univariate optimization problem. Plot the function you're trying to minimize and show two runs of `gd`, one where it gets caught in a local minimum and one where it manages to make it to a global minimum. (Use different starting points to accomplish this.)

If you implemented it well, this should work in multiple dimensions, too:

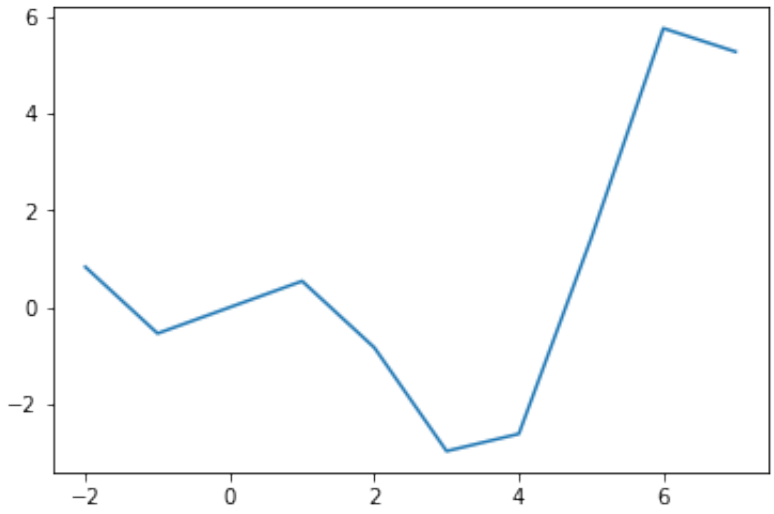
In [66]:
```
#WU4 code here
import matplotlib.pyplot as plt

import numpy as np
import gd
import math as mt
#local minimum it get's caught in local
x, trajectory = gd.gd(lambda x: np.linalg.norm(x*np.cos(x)), lambda x:
np.cos(x)-x*np.sin(x), -2, 100, 0.5)
print("Printing the location of local minima in terms of x axis")
print(x)


#global minimum
x, trajectory = gd.gd(lambda x: np.linalg.norm(x*np.cos(x)), lambda x:
np.cos(x)-x*np.sin(x),2, 100,0.5)
print("Printing the coordination of local minima in terms of x axis")
print(x)

#plotting the function
x = np.array(range(-2,8))
y = eval('x*np.cos(x)')
plt.plot(x, y)
plt.show()
```

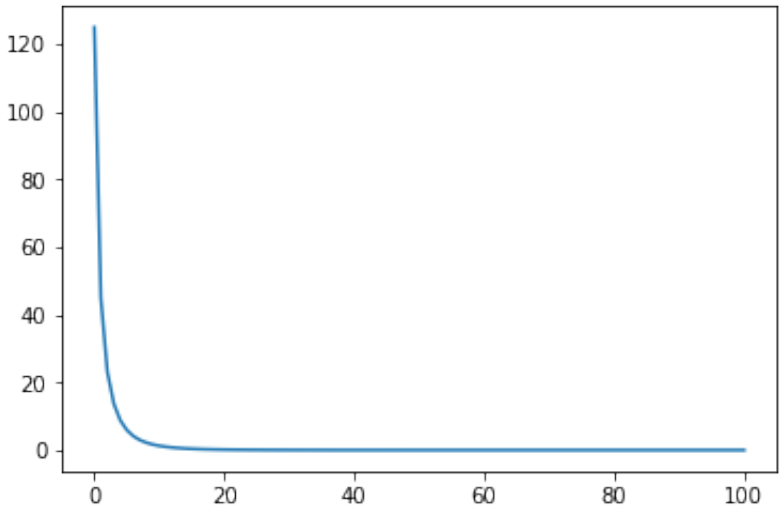
```
Printing the location of local minima in terms of x axis
-0.860333589030166
Printing the coordination of local minima in terms of x axis
3.4256184594817274
```



In [71]:
```
from matplotlib.pyplot import *
import numpy as np
x, trajectory = gd.gd(lambda x: np.linalg.norm(x)**2, lambda x: 2*x, n
p.array([10,5]), 100, 0.2)
print(x)
# array([ 0.0036459 ,  0.00182295])
plot(trajectory)
```

```
[0.0036459  0.00182295]
```

Out[71]:
```
[<matplotlib.lines.Line2D at 0x10fbb1d90>]
```



Our generic linear classifier implementation is in `linear.py`. The way this works is as follows. We have an interface `LossFunction` that we want to minimize. This must be able to compute the loss for a pair `Y` and `Yhat` where, the former is the truth and the latter are the predictions. It must also be able to compute a gradient when additionally given the data `X`. This should be all you need for these.

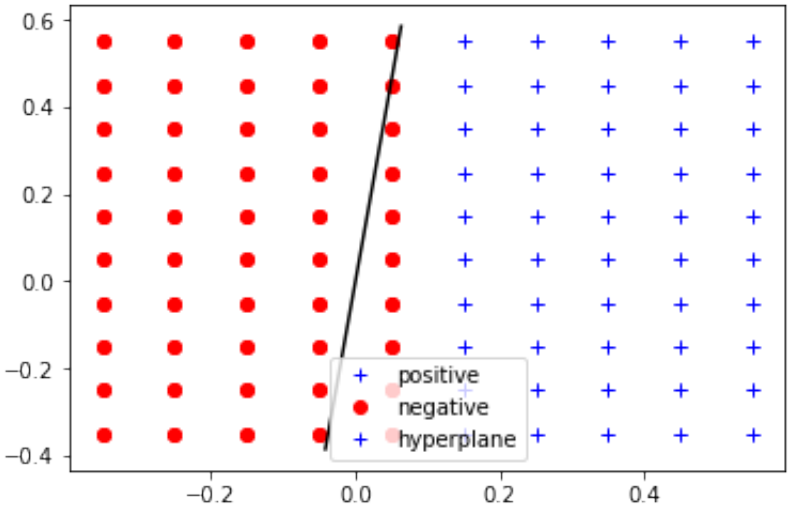
There are three loss function stubs: `SquaredLoss` (which is implemented for you!), `LogisticLoss` and `HingeLoss` (both of which you'll have to implement. My suggestion is to hold off implementing the other two until you have the linear classifier working.

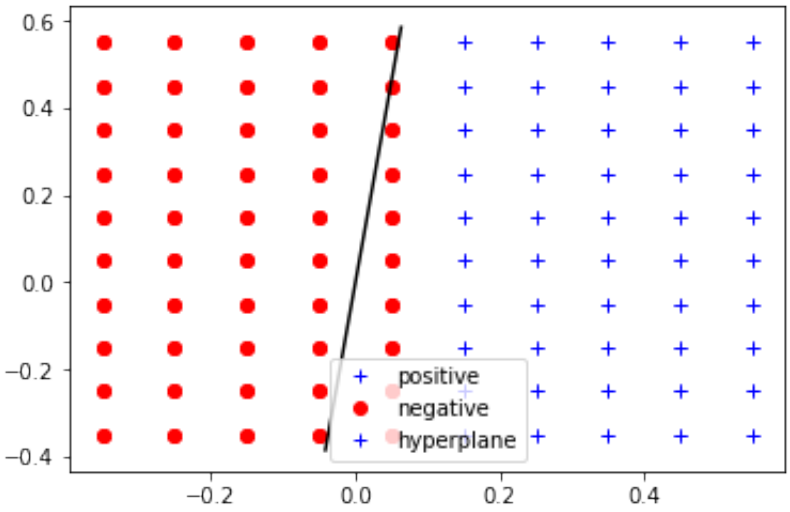
The `LinearClassifier` class is a stub implemention of a generic linear classifier with an l2 regularizer. It is *unbiased* so all you have to take care of are the weights. Your implementation should go in `train`, which has a handful of stubs. The idea is to just pass appropriate functions to `gd` and have it do all the work. See the comments inline in the code for more information.

Once you've implemented the function evaluation and gradient, we can test this. We'll begin with a very simple 2D example data set so that we can plot the solutions. We'll also start with *no regularizer* to help you figure out where errors might be if you have them. (You'll have to import `mlGraphics` to make this work.)

In [2]:
```
import linear
import datasets
import mlGraphics
import runClassifier
from matplotlib.pyplot import *
f = linear.LinearClassifier({'lossFunction': linear.SquaredLoss(), 'la
mbda': 0, 'numIter': 100, 'stepSize': 0.5})
runClassifier.trainTestSet(f, datasets.TwoDAxisAligned)
# Training accuracy 0.91, test accuracy 0.86
print(f)
# w=array([ 2.73466371, -0.29563932])
mlGraphics.plotLinearClassifier(f, datasets.TwoDAxisAligned.X, dataset
s.TwoDAxisAligned.Y)
show(False)
```

```
Training accuracy 0.91, test accuracy 0.86
w=array([ 2.73466371, -0.29563932])
```





Note that even though this data is clearly linearly separable, the *unbiased* classifier is unable to perfectly separate it.

If we change the regularizer, we'll get a slightly different solution:

In [8]:
```
import linear
import runClassifier
import datasets

f = linear.LinearClassifier({'lossFunction': linear.SquaredLoss(), 'la
mbda': 10, 'numIter': 100, 'stepSize': 0.5})

runClassifier.trainTestSet(f, datasets.TwoDAxisAligned)
# Training accuracy 0.9, test accuracy 0.86
print(f)

# w=array([ 1.30221546, -0.06764756])
```

```
Training accuracy 0.9, test accuracy 0.86
w=array([ 1.30221546, -0.06764756])
```

As expected, the weights are *smaller*.

Now, we can try different loss functions. Implement logistic loss and hinge loss. Here are some simple test cases:

In [7]:
```
f = linear.LinearClassifier({'lossFunction': linear.LogisticLoss(), 'l
ambda': 10, 'numIter': 100, 'stepSize': 0.5})
runClassifier.trainTestSet(f, datasets.TwoDDiagonal)
# Training accuracy 0.99, test accuracy 0.86
print(f)
# w=array([ 0.29809083,  1.01287561])

f = linear.LinearClassifier({'lossFunction': linear.HingeLoss(), 'lamb
da': 1, 'numIter': 100, 'stepSize': 0.5})
runClassifier.trainTestSet(f, datasets.TwoDDiagonal)
# Training accuracy 0.98, test accuracy 0.86
print(f)
# w=array([ 1.17110065,  4.67288657])
```

```
Training accuracy 0.99, test accuracy 0.86
w=array([0.29809083, 1.01287561])
Training accuracy 0.98, test accuracy 0.86
w=array([1.17110065, 4.67288657])
```

WU5 (5%):

For each of the loss functions, train a model on the binary version of the wine data (called WineDataBinary) and evaluate it on the test data. You should use lambda=1 in all cases. Which works best? For that best model, look at the learned weights. Find the *words* corresponding to the weights with the greatest positive value and those with the greatest negative value (this is like LAB3). Hint: look at WineDataBinary.words to get the id-to-word mapping. List the top 5 positive and top 5 negative and explain.

[Your WU5 answer here]

In [56]:
```
#WU5 code here
import linear
import util
import numpy as np
import runClassifier
import datasets
f = linear.LinearClassifier({'lossFunction': linear.SquaredLoss(), 'la
mbda': 1, 'numIter': 100, 'stepSize': 0.5})
runClassifier.trainTestSet(f, datasets.WineDataBinary)



# Training accuracy

f = linear.LinearClassifier({'lossFunction': linear.LogisticLoss(), 'l
ambda': 1, 'numIter': 100, 'stepSize': 0.5})
runClassifier.trainTestSet(f, datasets.WineDataBinary)
# Training accuracy

p=f.getWeigths()
ind = p.argsort()[-5:][::-1]
ind1= p.argsort()[:5]

print("Top 5 positive words from WineDataBinary are:")
for i in ind:
    print(datasets.WineDataBinary.words[i])
print("Top 5 negative words from WineDataBinary are:")
for j in ind1:
    print(datasets.WineDataBinary.words[j])

f = linear.LinearClassifier({'lossFunction': linear.HingeLoss(), 'lamb
da': 1, 'numIter': 100, 'stepSize': 0.5})
runClassifier.trainTestSet(f, datasets.WineDataBinary)
# Training accuracy


#Logistic loss works best as it gives better test accuracy.
#Giving a high training accuracy and testing accuracy
#Here we have weight vector based for WineDataBinary using LinearClass
ifier then we get the index of those words
#as we have to find labels so we go in the WordsBinary Datasets get th
e values with that top 5 weight or bottom 5
#weight correspondingly
```

```
Training accuracy 0.242914979757, test accuracy 0.313653136531
Training accuracy 0.995951417004, test accuracy 0.974169741697
Top 5 positive words from WineDataBinary are:
citrus
crisp
lime
acidity
tropical
Top 5 negative words from WineDataBinary are:
tannins
black
dark
cherry
blackberry
Training accuracy 0.753036437247, test accuracy 0.686346863469
```

Part 3: Classification with Many Classes *[0% -- up to 15% extra credit]*

Finally, we'll do multiclass classification using Scikit-learn functionality. You can find the documentation here: http://scikit-learn.org/stable/modules/multiclass.html (http://scikit-learn.org/stable/modules/multiclass.html).

Quiz bowl is a game in which two teams compete head-to-head to answer questions from different areas of knowledge. It lets players interrupt the reading of a question when they know the answer. The goal here is to see how well a classifier performs in predicting the `Answer` of a question when a different portion of the question is revealed.

Here's an example question from the development data:

```
206824,dev,History,Alan Turing,"This man and Donald Bayley created a secur
e voice communications machine called ""Delilah"". ||| The Chinese Room Ex
periment was developed by John Searle in response to one of this man's nam
esake tests. ||| He showed that the halting problem was undecidable. ||| H
e devised a bombe with Gordon Welchman that found the settings of an Enigm
a machine. ||| One of this man's eponymous machines which can perform any
computing task is his namesake ""complete."" Name this man, whose eponymou
s test is used to determine if a machine can exhibit behavior indistinguis
hable from that of a human."
```

The more of the question you get, the easier the problem becomes.

The default code below just runs OVA and AVA on top of a linear SVM (it might take a few seconds):

In [3]:
```
import sklearn.metrics
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsRestClassifier, OneVsOneClassifier
from numpy import *
import datasets
import imp

imp.reload(datasets)

if not datasets.Quizbowl.loaded:
    datasets.loadQuizbowl()

print('\n\nRUNNING ON EASY DATA\n')

print('training ova')
X = datasets.QuizbowlSmall.X
Y = datasets.QuizbowlSmall.Y
ova = OneVsOneClassifier(LinearSVC(random_state=0)).fit(X, Y)
print('predicting ova')
ovaDevPred = ova.predict(datasets.QuizbowlSmall.Xde)
print('error = {0}'.format(mean(ovaDevPred != datasets.QuizbowlSmall.Y
de)))

print('training ava')
ava = OneVsRestClassifier(LinearSVC(random_state=0)).fit(X, Y)
print('predicting ava')
avaDevPred = ava.predict(datasets.QuizbowlSmall.Xde)
print('error = {0}'.format(mean(avaDevPred != datasets.QuizbowlSmall.Y
de)))

print('\n\nRUNNING ON HARD DATA\n')

print('training ova')
X = datasets.QuizbowlHardSmall.X
Y = datasets.QuizbowlHardSmall.Y
ova = OneVsOneClassifier(LinearSVC(random_state=0)).fit(X, Y)
print('predicting ova')
ovaDevPred = ova.predict(datasets.QuizbowlHardSmall.Xde)
print('error = {0}'.format(mean(ovaDevPred != datasets.QuizbowlHardSma
ll.Yde)))

print('training ava')
ava = OneVsRestClassifier(LinearSVC(random_state=0)).fit(X, Y)
print('predicting ava')
avaDevPred = ava.predict(datasets.QuizbowlHardSmall.Xde)
print('error = {0}'.format(mean(avaDevPred != datasets.QuizbowlHardSma
ll.Yde)))

savetxt('predictions.txt', avaDevPred)
```

```
Loading Quizbowl dataset...

total labels: 2370
unique features: 8416
total training examples: 8845

Loading QuizbowlSmall dataset...

total labels: 31
unique features: 8416
total training examples: 361

Loading QuizbowlHard dataset...

total labels: 2370
unique features: 4132
total training examples: 8845

Loading QuizbowlHardSmall dataset...

total labels: 31
unique features: 4132
total training examples: 361


RUNNING ON EASY DATA

training ova
predicting ova
error = 0.293413173653
training ava
predicting ava
error = 0.218562874251


RUNNING ON HARD DATA

training ova
predicting ova
error = 0.595808383234
training ava
predicting ava
error = 0.553892215569
```

When you run the code above, you should see some statistics of the loaded datasets and the following error rates on two of the datasets `QuizbowlSmall` and `QuizbowlHardSmall` using OVA and AVA:

```
RUNNING ON EASY DATA

training ova
predicting ova
error = 0.293413
training ava
predicting ava
error = 0.218563


RUNNING ON HARD DATA

training ova
predicting ova
error = 0.595808
training ava
predicting ava
error = 0.553892
```

This is running on a shrunken version of the data (that only contains answers that occur at least 20 times in the data).

The first ("easy") version is when you get to see the entire question. The second ("hard") version is when you only get to use the first two sentences. It's clearly significantly harder to answer!

Your task is to achieve the lowest possible error on the development set for `QuizbowlSmall` and `QuizbowlHardSmall`. You will get 5% extra credit for getting lower error (by at least absolute 1%) on *either* dataset than the errors presented above (21.86% for `QuizbowlSmall` and 55.39% for `QuizbowlHardSmall`).

You're free to use the training data in any way you want, but you must include your code in `quizbowl.py`, submit your predictions file(s), and a writeup here that says what you did, in order to receive the extra credit. The script `quizbowl.py` includes a command in the last line that saves predictions to a text file `predictions.txt`. You need to edit this line to rename the file to either `predictionsQuizbowlSmall.txt` or `predictionsQuizbowlHardSmall.txt` dependent on the dataset: that's what you upload for the EC.

WU-EC2 (5%):

[YOUR WU-EC2 WRITEUP HERE]

In [8]:
```
#I got 16% in AVA for small 50.5 for AVA in case of Hard
```

```
import sklearn.metrics
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsRestClassifier, OneVsOneClassifier
from numpy import *
from sklearn.ensemble import RandomForestClassifier
import datasets
import imp

imp.reload(datasets)

if not datasets.Quizbowl.loaded:
    datasets.loadQuizbowl()

print('\n\nRUNNING ON EASY DATA\n')

print('training ova')
X = datasets.QuizbowlSmall.X
Y = datasets.QuizbowlSmall.Y
ova = OneVsOneClassifier(LinearSVC(random_state=0)).fit(X, Y)
print('predicting ova')
ovaDevPred = ova.predict(datasets.QuizbowlSmall.Xde)
print('error = {0}'.format(mean(ovaDevPred != datasets.QuizbowlSmall.Y
de)))
#Ref:http://blog.yhat.com/posts/random-forests-in-python.html
#n_estimators the more it is the better it will do oob score helped me
reach to the estimators in each case
#jobs to run in parallel to fit and predict was -1  to set it as numbe
r of cores in processor
#So this makes a better classification model
print('training ava')
ava = OneVsRestClassifier(RandomForestClassifier(n_estimators=1000,
                                    oob_score=True, n_jobs=-1, verbose
=1)).fit(X, Y)
print('predicting ava')
avaDevPred = ava.predict(datasets.QuizbowlSmall.Xde)
print('error = {0}'.format(mean(avaDevPred != datasets.QuizbowlSmall.Y
de)))
savetxt('predictionsQuizbowlSmall.txt', avaDevPred)
print('\n\nRUNNING ON HARD DATA\n')

print('training ova')
X = datasets.QuizbowlHardSmall.X
Y = datasets.QuizbowlHardSmall.Y
ova = OneVsOneClassifier(LinearSVC(random_state=0)).fit(X, Y)
print('predicting ova')
ovaDevPred = ova.predict(datasets.QuizbowlHardSmall.Xde)

print('training ava')
```

```
ava = OneVsRestClassifier(RandomForestClassifier(n_estimators=1000,
                                    oob_score=True, n_jobs=-1, verbose
=1)).fit(X, Y)
print('predicting ava')
avaDevPred = ava.predict(datasets.QuizbowlHardSmall.Xde)
print('error = {0}'.format(mean(avaDevPred != datasets.QuizbowlHardSma
ll.Yde)))

savetxt('predictionsQuizbowlHardSmall.txt', avaDevPred)
```

```
Loading Quizbowl dataset...

total labels: 2370
unique features: 8416
total training examples: 8845

Loading QuizbowlSmall dataset...

total labels: 31
unique features: 8416
total training examples: 361

Loading QuizbowlHard dataset...

total labels: 2370
unique features: 4132
total training examples: 8845

Loading QuizbowlHardSmall dataset...

total labels: 31
unique features: 4132
total training examples: 361


RUNNING ON EASY DATA

training ova
predicting ova
error = 0.293413173653
training ava

[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
```

```
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
```

```
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 100 out of 100 | elapsed:    0.1s finish
ed
```

```
predicting ava
```

```
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
```

```
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.1s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.1s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
```

```
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 100 out of 100 | elapsed:    0.0s finishe
d
```

```
error = 0.185628742515


RUNNING ON HARD DATA

training ova
predicting ova
training ava
```

```
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.5s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.8s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    1.1s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    1.0s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    0.9s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
```

```
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.5s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.8s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    1.0s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    0.8s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.5s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.8s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    1.0s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    0.8s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    0.9s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    0.9s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.5s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.8s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    1.0s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    0.8s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.5s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.9s
```

```
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    1.2s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.5s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.9s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    1.1s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    1.0s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    1.0s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.5s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.9s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    1.0s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.5s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.9s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    1.0s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.5s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.9s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    1.1s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    0.9s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    0.9s fini
shed
```

```
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.6s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    0.8s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    0.9s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    0.8s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    0.9s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.6s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    0.8s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.6s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    0.8s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.5s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.9s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    1.0s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.5s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.8s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    1.1s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
```

```
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    0.9s fini
shed
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:    0.2s
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:    0.4s
[Parallel(n_jobs=-1)]: Done 792 tasks      | elapsed:    0.8s
[Parallel(n_jobs=-1)]: Done 1000 out of 1000 | elapsed:    0.9s fini
shed
```

```
predicting ava
```

```
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.3s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.4s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.3s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.3s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.3s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.3s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.3s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
```

```
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.3s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.4s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.3s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.4s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.3s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.3s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.3s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.3s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.4s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.3s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.4s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.3s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.2s
```

```
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.3s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.3s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.4s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.3s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.4s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.3s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.3s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.3s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.3s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.4s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.3s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.4s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.4s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.4s finis
hed
```

```
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.3s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.3s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.3s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.3s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.3s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.3s finis
hed
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.0s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.1s
```

```
error = 0.523952095808
```

```
[Parallel(n_jobs=4)]: Done 442 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 792 tasks      | elapsed:    0.2s
[Parallel(n_jobs=4)]: Done 1000 out of 1000 | elapsed:    0.3s finis
hed
```

Additionally, you can get extra credit for providing the lowest-error solution on the full versions of the easy and hard problems, `Quizbowl` and `QuizbowlHard` in comparison to your classmates' solutions. There will be two separate (hidden) leaderboards for each of these two datasets. You will receive 5% if your solution is the best for the respective dataset (first place), 3% for second place and 1% for third. We will reveal the top three scores for each dataset after the submission period is over, and you are welcome to compete in both. Note that this problem is much harder due to the larger number of class labels. A simple majority label classifier has an error of 99.89%.

You're free to use the training data in any way you want, but you must include your code in `quizbowl.py`, submit your predictions file(s) (`predictionsQuizbowl.txt` and/or `predictionsQuizbowlHard.txt`), and a writeup here that says what you did, in order to receive the extra credit.

WU-EC3 (up to 10%):

[YOUR WU-EC3 WRITEUP HERE]

In [ ]:
```
import sklearn.metrics
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsRestClassifier, OneVsOneClassifier
from numpy import *
from sklearn.ensemble import RandomForestClassifier
import datasets
import imp

imp.reload(datasets)

if not datasets.Quizbowl.loaded:
    datasets.loadQuizbowl()

print('\n\nRUNNING ON EASY DATA\n')

print('training ova')
X = datasets.Quizbowl.X
Y = datasets.Quizbowl.Y
ova = OneVsOneClassifier(LinearSVC(random_state=0)).fit(X, Y)
print('predicting ova')
ovaDevPred = ova.predict(datasets.Quizbowl.Xde)
print('error = {0}'.format(mean(ovaDevPred != datasets.Quizbowl.Yde)))
#Ref:http://blog.yhat.com/posts/random-forests-in-python.html
```

```
#n_estimators the more it is the better it will do oob score helped me
reach to the estimators in each case
#jobs to run in parallel to fit and predict was -1  to set it as numbe
r of cores in processor
#So this makes a better classification model
print('training ava')
ava = OneVsRestClassifier(RandomForestClassifier(n_estimators=1000,
                                    oob_score=True, n_jobs=-1, verbose
=5)).fit(X, Y)
print('predicting ava')
avaDevPred = ava.predict(datasets.Quizbowl.Xde)
print('error = {0}'.format(mean(avaDevPred != datasets.Quizbowl.Yde)))
savetxt('predictionsQuizbowl.txt', avaDevPred)
print('\n\nRUNNING ON HARD DATA\n')

print('training ova')
X = datasets.QuizbowlHard.X
Y = datasets.QuizbowlHard.Y
ova = OneVsOneClassifier(LinearSVC(random_state=0)).fit(X, Y)
print('predicting ova')
ovaDevPred = ova.predict(datasets.QuizbowlHard.Xde)

print('training ava')
ava = OneVsRestClassifier(RandomForestClassifier(n_estimators=1000,
                                    oob_score=True, n_jobs=-1, verbose
=5)).fit(X, Y)
print('predicting ava')
avaDevPred = ava.predict(datasets.QuizbowlHard.Xde)
print('error = {0}'.format(mean(avaDevPred != datasets.QuizbowlHard.Yd
e)))

savetxt('predictionsQuizbowlHard.txt', avaDevPred)
```

In [ ]:

```
#WU EC 3 writeup here:


# I used a better classifier than the Support vector classifier I used
RandomForestClassifier to improve accuracy.

#It comes into use to make predicition for categories with multiple po
ssible outcomes here
#i found in QuizData that was helpful.It does'nt require much tuning a
s we see in Support vectors.So I used it and
#and improved accuracy
```