**allwallfeatures\_pca300\_nov17.mtx"**

pca with 300 dimensions

ntrain = 250

nvalidation = 50

lambda\_list = list(range(5,150))

lasso

>>> test\_list

array([ 1., 1., 0., 1., 0., 1., 1., 1., 1., 0., 1., 1., 1.,

1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1.,

1., 1., 1., 1., 1., 1., 0., 0., 1., 0., 0., 1., 1.,

1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0.,

0., 1., 0., 1., 1., 1., 0., 1.])

>>> sum(test\_list)

46.0

>>> sum(test\_list) / test\_list.size

0.76666666666666672

>>> io.mmwrite("pca300", pca)

pca300 with lambda from 0 – 10

error on test set : 0.683333333333

pca with 100 components and lambda 50-150

words guessed correctly: 45.0

percentage guessed correctly: 0.75

[ 1. 1. 0. 1. 0. 0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1.

1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 1. 0.

0. 1. 1. 1. 0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 1.

0. 1. 1. 1. 0. 1.]

pca with 50 dimensions

words guessed correctly: 44.0

percentage guessed correctly: 0.733333333333

lambda 50-100

dimensions – 300

accuracy improves dramatically at ~12 features

we get a max accuracy of 0.8 at about 120 features

array([ 0.26666667, 0.28333333, 0.36666667, 0.38333333, 0.43333333,

0.46666667, 0.48333333, 0.5 , 0.53333333, 0.58333333,

0.61666667, 0.71666667, 0.71666667, 0.68333333, 0.68333333,

0.7 , 0.66666667, 0.68333333, 0.68333333, 0.68333333,

0.68333333, 0.66666667, 0.65 , 0.66666667, 0.66666667,

0.66666667, 0.66666667, 0.68333333, 0.68333333, 0.66666667,

0.65 , 0.66666667, 0.68333333, 0.7 , 0.7 ,

0.68333333, 0.68333333, 0.7 , 0.7 , 0.7 ,

0.7 , 0.68333333, 0.68333333, 0.7 , 0.7 ,

0.7 , 0.66666667, 0.68333333, 0.68333333, 0.68333333,

0.68333333, 0.66666667, 0.68333333, 0.68333333, 0.7 ,

0.71666667, 0.7 , 0.7 , 0.68333333, 0.68333333,

0.7 , 0.68333333, 0.7 , 0.7 , 0.71666667,

0.73333333, 0.71666667, 0.75 , 0.73333333, 0.73333333,

0.71666667, 0.73333333, 0.73333333, 0.73333333, 0.73333333,

0.73333333, 0.73333333, 0.71666667, 0.71666667, 0.7 ,

0.71666667, 0.71666667, 0.71666667, 0.71666667, 0.71666667,

0.71666667, 0.71666667, 0.73333333, 0.73333333, 0.73333333,

0.73333333, 0.73333333, 0.73333333, 0.71666667, 0.71666667,

0.71666667, 0.71666667, 0.71666667, 0.75 , 0.75 ,

0.75 , 0.75 , 0.75 , 0.75 , 0.75 ,

0.75 , 0.73333333, 0.73333333, 0.75 , 0.73333333,

0.75 , 0.76666667, 0.76666667, 0.76666667, 0.76666667,

0.76666667, 0.78333333, 0.8 , 0.76666667, 0.78333333,

0.78333333, 0.78333333, 0.78333333, 0.78333333, 0.78333333,

0.78333333, 0.78333333, 0.78333333, 0.78333333, 0.78333333,

0.78333333, 0.78333333, 0.78333333, 0.78333333, 0.76666667,

0.75 , 0.75 , 0.75 , 0.76666667, 0.76666667,

0.76666667, 0.75 , 0.75 , 0.78333333, 0.76666667,

0.76666667, 0.76666667, 0.73333333, 0.75 , 0.76666667,

0.76666667, 0.76666667, 0.75 , 0.76666667, 0.75 ,

0.76666667, 0.76666667, 0.76666667, 0.76666667, 0.76666667,

0.76666667, 0.76666667, 0.75 , 0.75 , 0.75 ,

0.73333333, 0.73333333, 0.73333333, 0.75 , 0.75 ,

0.75 , 0.75 , 0.75 , 0.75 , 0.75 ,

0.75 , 0.75 , 0.75 , 0.75 , 0.75 ,

0.75 , 0.75 , 0.75 , 0.75 , 0.75 ,

0.75 , 0.75 , 0.75 , 0.75 , 0.75 ,

0.75 , 0.73333333, 0.73333333, 0.73333333, 0.73333333,

0.73333333, 0.73333333, 0.73333333, 0.73333333, 0.75 ,

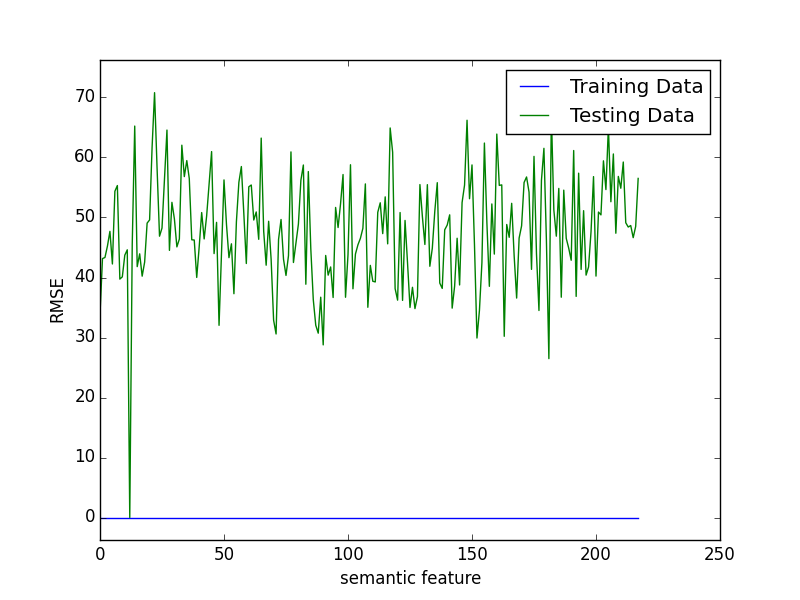
0.75 , 0.75 , 0.75 , 0.75 , 0.75 ,

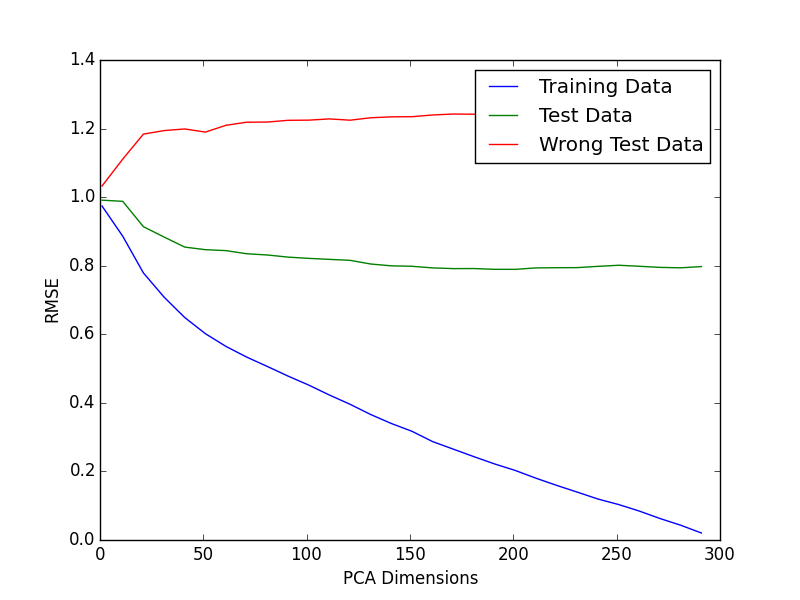
0.75 , 0.75 , 0.75 , 0.75 , 0.75 ,

0.75 , 0.75 , 0.75 , 0.75 , 0.75 ,

0.75 , 0.75 , 0.75 ])

root mean square error for each semantic feature with training and test dat





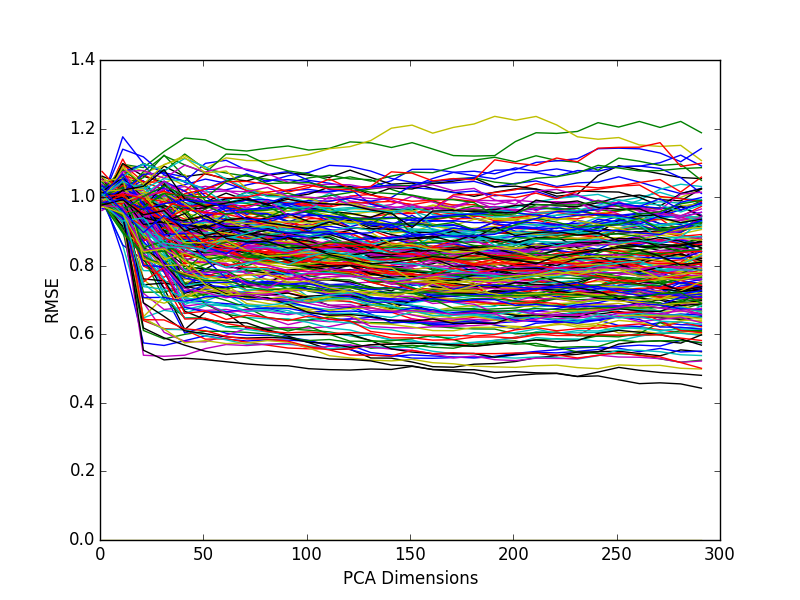
root mean square error calculated as

ypred = x.dot(w)

for each semantic feature

rmse = sqrt(1/n \* (ypred-y)^2)

total rmse = sum (rmse for each semantic feature) / num of semantic features



rmse per semantic feature

some stay high ~ 1

some drop.

need to find which ones!

dif = rmsetest\_matrix[0] - rmsetest\_matrix[29]

this should decrease with more pca dimensions.. so dif <0 is good.

if error increases it is bad!

GOOD SEMANTIC FEATURES:

Semantic Features that rmse decreased

(array([ 14, 21, 22, 27, 33, 35, 45, 65, 77, 101, 117, 118, 148,

150, 155, 160, 179, 182, 191, 205, 211]),)

worse semantic features :

>>> b = dif > 0.5

>>> b.nonzero()

(array([ 88, 90, 152, 163, 181]),)

features that have high rmse > 1

they don’t match with the ones above..

highrmse = rmsetest\_matrix[29]>1

highrmse.nonzero()

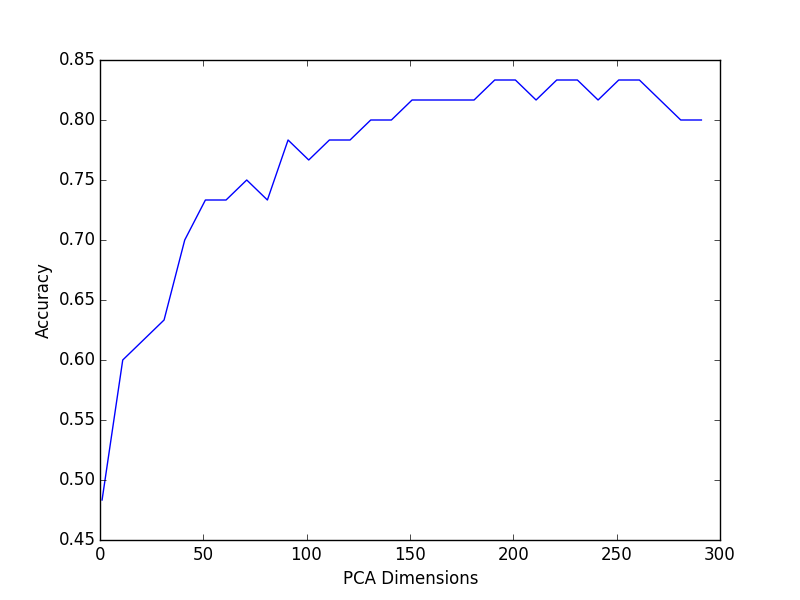
(array([ 14, 21, 22, 27, 33, 45, 65, 77, 117, 118, 148, 155, 160,

179, 182, 205]),)

# all this was done on test data. should probably be done on traingin data instead? or validation data!

#

% can I improve my prediction of getting rid of bad rmse features?

accuracy calculated as what percentage of 2 words guessed correctly by our model

pca:

[1, 11, 21, 31, 41, 51, 61, 71, 81, 91, 101, 111, 121, 131, 141, 151, 161, 171, 181, 191, 201, 211, 221, 231, 241, 251, 261, 271, 281, 291]

accuracy : [0.48333333333333334, 0.59999999999999998, 0.6166666666666667, 0.6333333333333333, 0.69999999999999996, 0.73333333333333328, 0.73333333333333328, 0.75, 0.73333333333333328, 0.78333333333333333, 0.76666666666666672, 0.78333333333333333, 0.78333333333333333, 0.80000000000000004, 0.80000000000000004, 0.81666666666666665, 0.81666666666666665, 0.81666666666666665, 0.81666666666666665, 0.83333333333333337, 0.83333333333333337, 0.81666666666666665, 0.83333333333333337, 0.83333333333333337, 0.81666666666666665, 0.83333333333333337, 0.83333333333333337, 0.81666666666666665, 0.80000000000000004, 0.80000000000000004]

Rank the rmse errors of all the words for each x datapoint and then find the rank of the right word!

ranks of the 60 words

[56, 44, 23, 41, 50, 17, 21, 1, 2, 31, 31, 7, 22, 56, 37, 6, 51, 46, 53, 46, 28, 56, 25, 35, 56, 26, 0, 30, 35, 19, 50, 3, 53, 57, 1, 26, 34, 43, 15, 40, 7, 32, 46, 55, 8, 55, 29, 44, 21, 39, 17, 57, 59, 54, 22, 44, 8, 46, 5, 53]

sum (rankarray <10)

11 words are ranked in the first 10 words… not very good!