Machine Learning Project

Fall 2013

KDDCup 2004 Quantum physics data set

Team Name: The Perceptrons

Members:

Kartik Saxena
Student ID – **84568218**
UCINetID – **ksaxena**

Prithviraj Vasanth (self)
Student ID: 9224 7650
UCINetID - **pvasanth**

--

Learner 1: Linear Classifier (Perceptron) - Prithvi

Training method:

Take features, impute the missing values, and after shuffling, split into test and validation set.

Without feature selection:

1. Use train set to train a percept classifier on the features. → Obtain weights, theta.

Note: To consider using different values of alpha(step size and limits)

Learner: Linear Classifier (perceptron)

Scoring performance: Cost function - Loss function is Negative log-likelihood function on the sigmoid threshold on the linear response.

Learning : Adapt theta so as to minimize loss function (maximise cost function). Choosing appropriate alpha (step size of gradient descent).

Adapting may take a while for such a large data set, so must have a reasonable threshold value of steps.

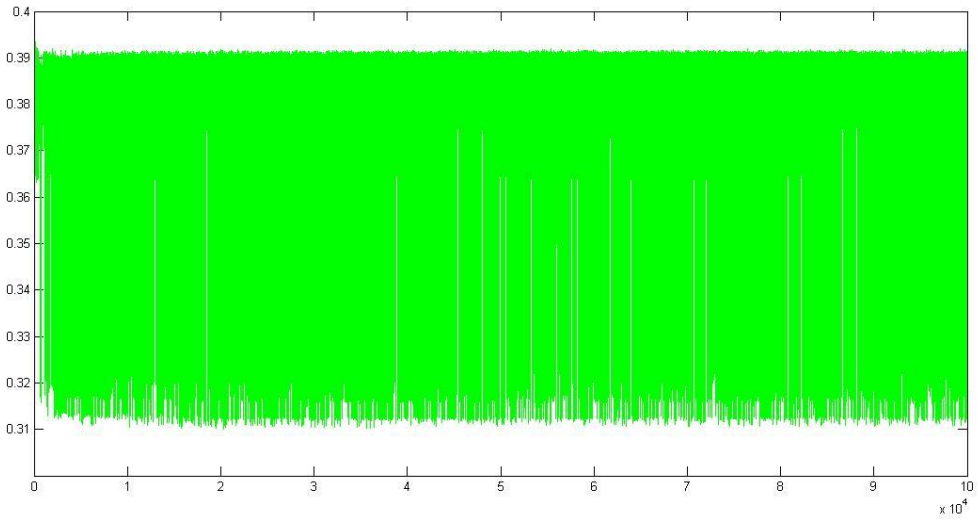
Note: We went with an online update method (better for a large data set to converge quickly when separable dataset.) however we noticed the data was pretty hard to separate. (And thus our motivation towards using an SVM reduced (as an SVM only tends to better the margin separation from the two classes (we felt the improvement would be minor) and our intuition)

Best result on trying across different values of num iterations and learning rate ‘alpha’ . The best error however on training was still poor, with respect to later

methods that we tried like decision trees and later decision forests./bagging. Due to the constant oscillation of error we figured that either the step size was too big or that the data has a lot of overlap between the two classes. We explored a little feature expansion on certain features by making them order 2. This improved the results a bit but still left us with a lot of uncertainty on which would give us more improvement. However, we did plan to use this percept classify learner along with other learners we got later, in the adaboost ensemble.

With Feature selection: We decided doing this by removing certain features based on how many zero values they had and felt those dimensions weren't moving the data around too much (rather not helping with the separation of the classes and therefore we felt was of less importance compared to the rest(at least our intuition)- we checked median value for zero in order not to consider it for training/test. The result we got on this was again not too impressive but marginally better than our earlier perceptron classifier (63.45% accuracy) We then wanted to explore dimensionality reduction techniques – PCA (explained in another section below as we combined PCA with a non-linear learning technique).

Percept Classify setup	**Test Error (sigmoid threshold mse)**
pc = perceptClassify(XA,YA,0.001,100000); ss	**err = 0.33**
pc = perceptClassify(XA,YA,0.0001,100000);	**err = 0.32**
pc = perceptClassify(XA,YA,0.001,1000);	**err=0.37**



Learner 2: Decision Trees and Bagging

We set out on this next learner that was a non-linear and something that tended to overfit to the data given infinite depth. However, given that such a learner tends to memorise and perform badly on the test, we performed a set of experiments with varying the Depth value keeping a fixed MinParent value. We decided on iterating over a set of log separated values for minParent [1 2 4 8 16 .. until 512] and then narrowed down in the region between 256 and 512 again in large jumps. We then arrived at an optimum result at minParent of 400. We also figured out that this was consistently the best result for a depth of 10. We then set these two parameters and tried to explore feature selection in a similar manner. The function had a provision for number of features to split on and we found the best feature count was 65 (picked at random – but should be pretty representative of the data).

Error value on the Validation set for the decision tree with the best of the above values came to about 0.31 and 0.29 on training data. We noticed that there is some potential in using decision trees and we thought, likely, if combined, as done in Bagging.

So we tried out our next class of learners which was an ensemble of tree learners with the above settings. We now had to decide on two things, what Nbag size to use and what Bootstrap sample size to use, 'Nuse' as described in the algorithm written below. We got a tabulation of errors computed on the validation set with differing Nbag values [1 5 10 20 50 100] and varying N' values [0.8N 0.85N 0.9N N].

We got the following error matrix for differing Nbag and N' values.

5 X 4 matrix of Nbag values and sample sizes respectively.

erTr =

0.2783	0.2770	0.2760	0.2749
0.2779	0.2759	0.2766	0.2762
0.2738	0.2738	0.2738	0.2727
0.2713	0.2711	0.2723	0.2697
0.2715	0.2704	0.2716	0.2697

erVa =

0.2897	0.2895	0.2896	0.2865
0.2850	0.2863	0.2882	0.2860
0.2842	0.2856	0.2830	0.2852

0.2814 0.2854 0.2853 0.2845

0.2835 0.2844 0.2850 0.2816

bestTCerr =

0.2814 0.2844 0.2830 0.2816

Comparing the best Test performance on the data, we decided on using an Nbag value of 50 and N' value of 0.8N and also Nbag value of 100 and N' value of N. We found results with both these values and noticed that the best performance on the Test Data was 0.29104.

Perceptron_cs273	7	143.0	0.70896	188.0	0.70875

Our error had shown considerable performance improvement on the Test set with that iteration.

Learner 3: GaussBayesClassifier(withPCA)

First applied GaussBayesClassifier to the high dimensional data and what was achieved was as close as a random guess (~0.49) and so we performed PCA reducing it to 10 dimensions. Here applying the GaussBayesClassifier on a dimensionally reduced (PCA reduced training+test data set) the error achieved was more meaningful. The accuracy achieved on a crossvalidated set was 63.45%. We kept this aside to have it as part of our ensemble.

Learner 4: KNN Classify (K=11 and greater. Too slow to run)

We tried out a KNN Classifier with a set of K values and we picked K=11 and we got a validation error of 63.14% which was better than the perceptron algorithm. However, this being a heavily non-parametric method, took a long time predicting on the validation set. The Training aspect of the learner was very quick.

Learner 5: Perceptron with PCA reduction and higher order feature expansion on a select few features of the lower dimensional subspace.

We applied the perceptClassifier on the lower dimensional subspace training set and let it optimize in terms of theta and we tried to improve the 58% dismal accuracy by making some of the features have a quadratic feature expansion done on. This reduced the error on training to **0.381** at best. We learnt that the PCA reduction nor feature expansion helped out in improving the error.

Learner 6: *PCA with a fairly non-complex decision tree on the lower dimension subspace.*

Error obtained on the cross-validated subspace training data set = 0.3043, which is a good improvement. However to begin with we were not sure how to go about the transformation of higher dimensional Test data set into the same lower dimensional subspace as the Training data was transformed to. This led us to think about combining the Test and Training data together and perform a PCA. This we thought would allow us to still train our data based on the

training data of the transformed whole. Then we planned to apply the learner on the remaining of the transformed lower dimensional test data. This however we feel is not completely fair (given a more real time prediction environment, but for such a static test set it seems acceptable a method) but in general we did see improvements.

Adaboost on the same data set over different learners. As of now we have tried the adaboost with the bagged tree classifier that we learnt, the linear classifiers and we arrived at a good Test accuracy of 70.94%. The errors were in the same ballpark as the above Nbagged result. This meant we needed a better mix of the learners.

Neural Networks:

This is the last method we tried and are still trying and trying to get a grip of. We downloaded the package from Deep Learning Matlab toolbox available on github and tried to experiment with the nn package for the vanilla neural network set. We started out by setting it up with the activation function being sigm and the batchsize of updates to weights set to a reasonable number. We however found the code not to be smoothly working from examples posted. We are continuing to try it out, as we happen to hear that Neural networks are among some of the better ranked Learners. This is an interesting area that we are still exploring.

Who did what:

We mostly worked at times when we both were free and we would spend time post class discussing a method and then try out methods on our own. To begin with pre-processing and ways of imputing data were explored by Prithvi and Kartik figured out the data submission part. The percept Classifier and the PCA dimensionality reductions were done by Prithviraj and the feature selection bit was done together. The Tree Classifier was a combined effort with the initial tree parameters determined by Kartik and the ideal Nbag and N' values for the tree parameters determined by Prithvi. We decided to split some other simpler methods from homework and otherwise to also tryout in a crossvalidated manner on the dataset – knnClassify and its K value exploration was done by Kartik and the Bayes', PCA with Decision trees, PCA with feature expanded to higher order perceptClassify was done by Prithvi. The analyses as to which learners to ensemble was a combined effort and we came up with a set of these learners called upon to vote. ANN was piece of our plan that we are understanding currently. Prithviraj downloaded the deep learning toolkit and started exploring the features. Adaboost is a a combination we both are trying together and ability to handle weights vector on train method for a classifier was written by Prithvi.

References: Class Lecture slides and hw Matlab libraries provided for the classifiers. The rest of the code were written by us as need for data exploration.

Code for bagging:

Bagging with Tree Classifiers:

Code to find good Nbag size and N' (sampling size)

```
Nbag = [5,10,25,50, 100];
[N,D] = size(Xt);
```

```
Nset = [ceil(N*0.8) ceil(N*0.85) ceil(N*0.90) N];
er = zeros(size(Nbag,2),size(Nset,2));
meanT_error = zeros(size(Nbag,2));
meanV_error = zeros(size(Nbag,2));
for j=1:size(Nbag,2);
    for k=1:size(Nset,2);
        Classifiers = cell(1,Nbag(j)); % Allocate space
        for i=1:Nbag(j);
            ind = ceil( N*rand(Nset(k),1) ); % Bootstrap sample data
            Xi = Xt(ind, :); Yi = Yt(ind, :); % Select those indices
            Classifiers{i} = treeClassify(Xi,Yi,2,7,0,65); % Train
        end;

        % Test data Xtest
        [Ntest,D] = size(Xv);
        pred = zeros(Ntest,Nbag(j)); % Allocate space
        for i=1:Nbag(j), % Apply each classifier
            pred(:,i)= predict(Classifiers{i},Xv);
        end;
        pred = (mean(pred,2) > 0.5); % Vote on output (0 vs 1)
        c = find(pred~=Yv);
        erV(j,k) = size(c,1)/Ntest;

        %For Training data:
        pred = zeros(Ntest,Nbag(j)); % Allocate space

        for i=1:Nbag(j), % Apply each classifier
            pred(:,i)= predict(Classifiers{i},Xt);
        end;
        pred = (mean(pred,2) > 0.5); % Vote on output (0 vs 1)
        c1 = find(pred~=Yt);

        erT(j,k) = size(c1,1)/N;
    end;
    meanT_er(j) = mean(erT(j,:),2);
    meanV_er(j) = mean(erV(j,:),2);
end;
erT
erV
```