HTM Image Classification

Darbin Reyes

Motivation

Hierarchical temporal memory (HTM) is a learning and inference algorithm developed by the company Numenta. This algorithm is unique and interesting because its developers stress the use of biologically accurate neuroscience in its design. For example, evidence suggests that the human brain possesses a common cortical algorithm. That is, the neocortex of the brain, which houses much of the learning and memory facilities, utilizes the same algorithm to solve different problems across different modalities (audio, vision, somatosensory perception etc.). Moreover, there is a hierarchical structure to the way neurons are connected. The general structure observed is that large regions exist close to the sensory inputs and regions above that are smaller in size (contain less neurons). Another interesting observation is that specific network connections and region size are very different within and between species. This suggests that although hierarchy is important, the exact manner in which neurons connect within the hierarchy is not important. These, and a slew of other interesting observations from neuroscience are being used to guide the design of the HTM algorithm. My main goal is to learn how the HTM algorithm works. In order to do this as part of the course project requirement, I have chosen to explore and attempt to improve an existing application of HTMs. This application entails classification of simple images spanning 48 categories.

The Classification Problem

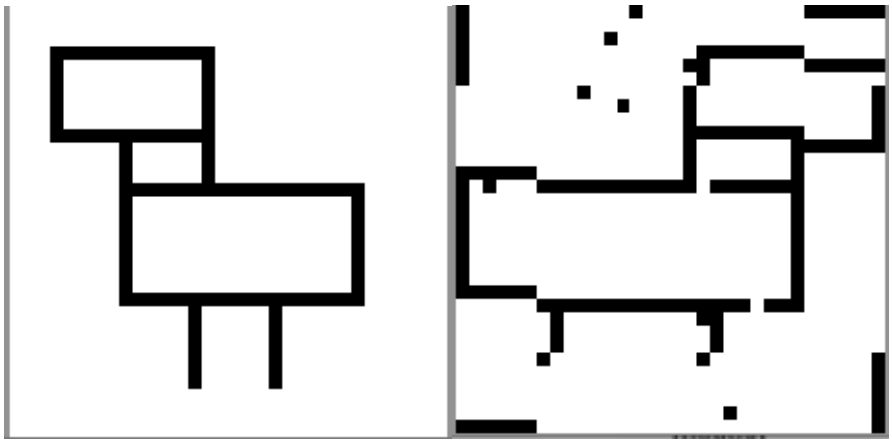


Figure 1: Left, a dog category training image. Right, a dog category test image.

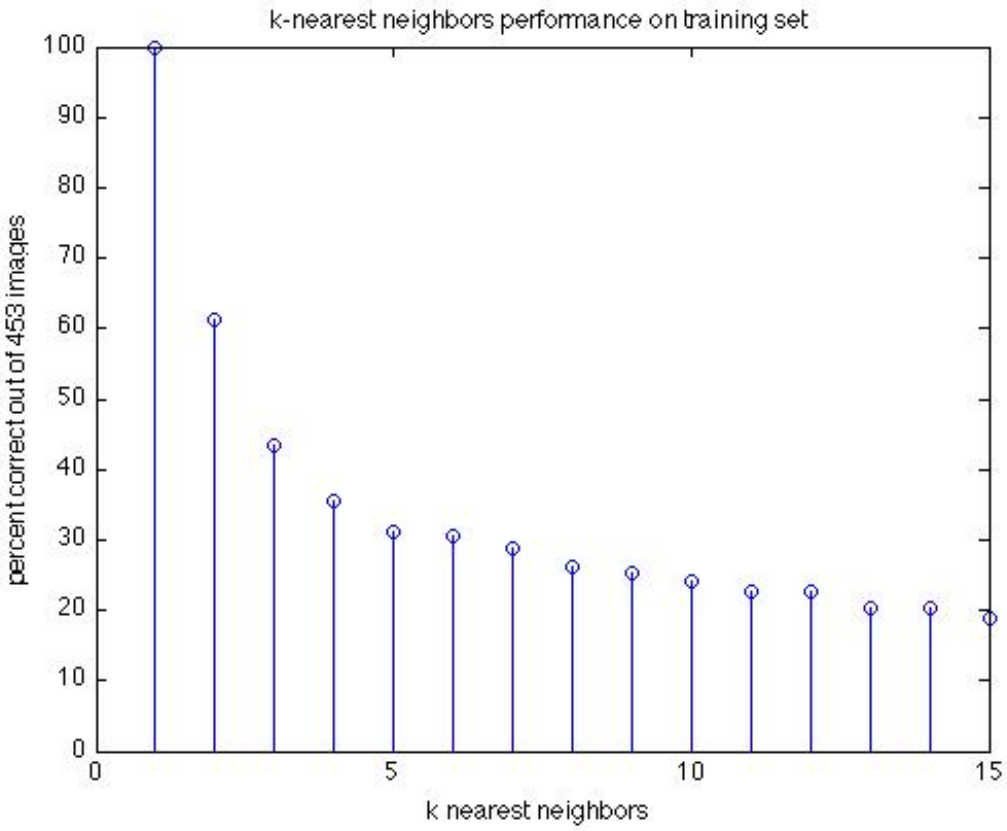
The problem at hand is to classify images. The images are binary 32x32 pixels and span 48 categories. An example image is shown above in figure 1. The left image shows a typical “dog” category image used for training. The right image is the same category but distorted used for testing. In general, the training data consists of scaled and translated versions of a standard category image, and the testing images are noisy distorted versions of the training images. In its current state, the HTM

algorithm can achieve 66 percent accuracy on 8941 test images. Given this, my goal will be to attempt to improve the algorithms accuracy by changing different aspects of the algorithm. Specifically, I will explore the effect of changing how the algorithm quantizes the training data in space and time. The algorithm makes assumptions about the spatial and temporal structure of the data that can perhaps be improved. My explorations will utilize the original training and test data to maintain a consistent benchmark.

Baseline Algorithm Comparisons

To compare the HTM algorithm with more standard techniques I ran K-nearest neighbors, logistic regression, and Support Vector Machines on the classification problem mentioned above. In each case, performance was very good on the training set but performance on the test set was not good, it remained below 5% in most cases. Each algorithm was run in Matlab, refer to the appendix for the corresponding code.

The performance of k nearest neighbors on the algorithm is shown in figure 2 below.



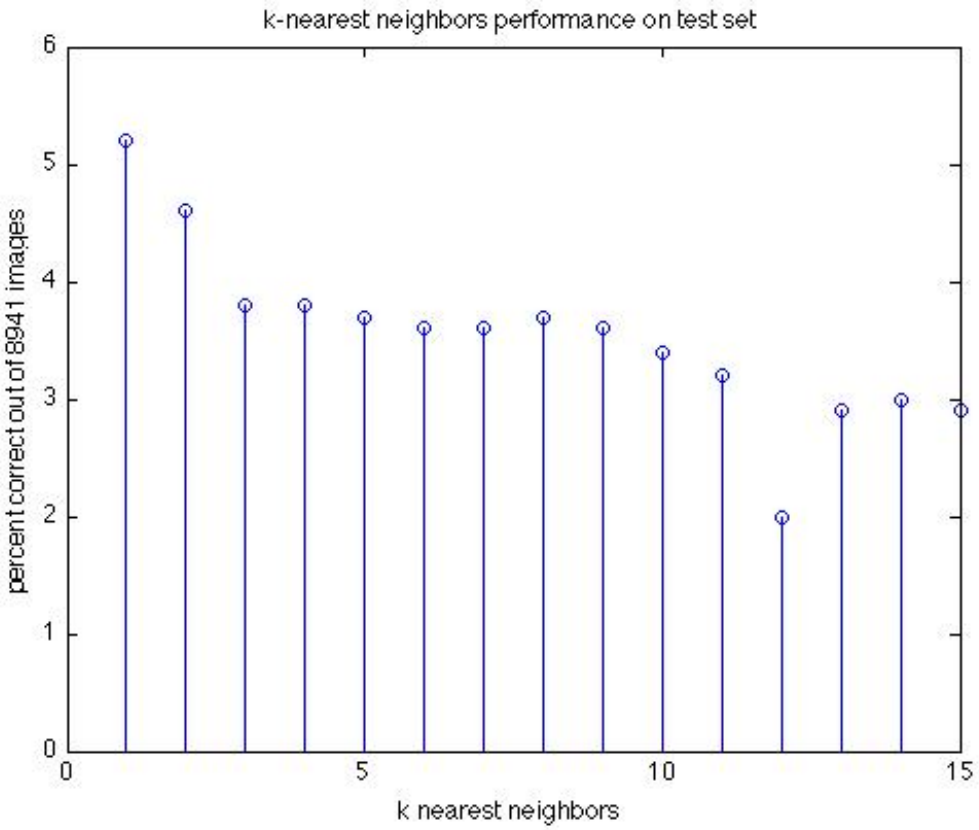
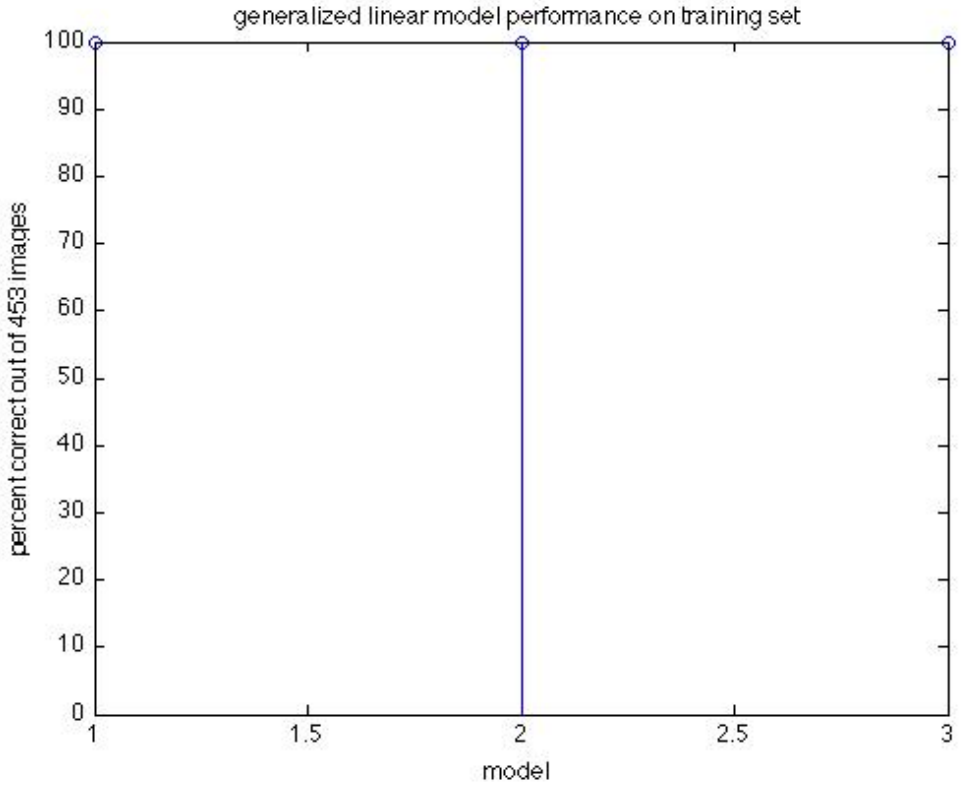


Figure 2: Top, performance of k nearest neighbors on the training set. Bottom, performance of k nearest neighbors on the test set.

On the training set, with one neighbor 100% accuracy is achieved however a decreasing accuracy is observed with an increasing number of neighbors. The decreased accuracy with more neighbors was unexpected but can be explained by considering the fact that the k- nearest neighbor algorithm is not invariant to scales and translations. As such, it is possible for two images of the same category images to be very different by the k-nearest neighbor measure. For instance, by simply translating an image, k-nearest neighbor result in a maximal hamming distance.

Similarly, performance of k-nearest neighbors on the test set was maximal for a single neighbor. Accuracy is reduced by about a factor of ten compared to the training set.

The glmfit() Matlab function was used to perform logistic regression on the classification problem. In addition to logistic regression, two other generalized linear models were run using glmfit(). In the figure, model 1 refers to glmfit() parameter distr='binomial', link='logit', model 2 refers to glmfit() parameter distr='binomial', link='logit', model 3 refers to glmfit() parameter distr='binomial', link='logit'. Accuracy on the training set was %100 for each model. However, on the test set accuracy was not much better than chance, around %2 for each.


generalized linear model performance on training set
percent correct out of 453 images
model
100
90
80
70
60
50
40
30
20
10
0
1
1.5
2
2.5
3

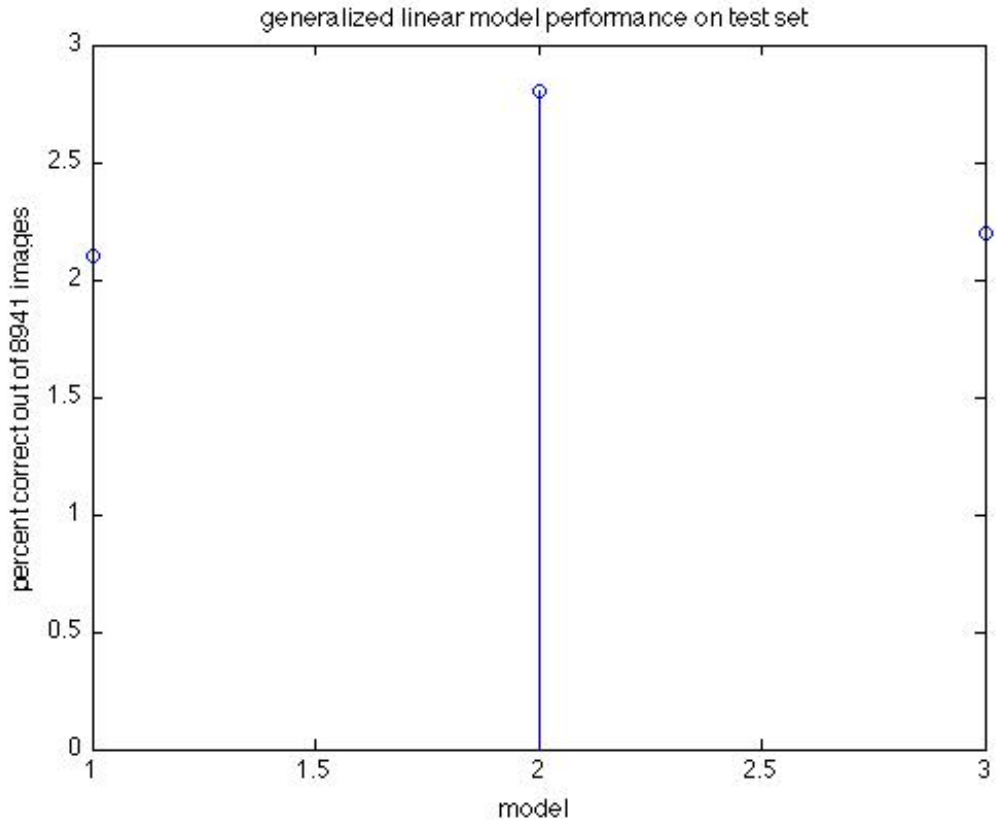

generalized linear model performance on test set
percent correct out of 8941 images
model
3
2.5
2
1.5
1
0.5
0
1
1.5
2
2.5
3

Figure 3: Top, generalized linear model performance on training set. Bottom, generalized linear model performance on test set.

The classification problem was test with the multi-class support vector machine implementation provided at: http://svmlight.joachims.org/. Both a linear and polynomial kernel were tested however the tests with a polynomial kernel were limited due to computation time constraints. For the linear kernel, several values of the "c" parameter were tested. The "c" parameter specifies the trade off between the training error and margin. Figure 4 shows the linear kernel performance. On the training set, %100 accuracy was achieved with c=5000. In general, the accuracy increased with c value. On the test set, accuracy peaked at about %7.5.

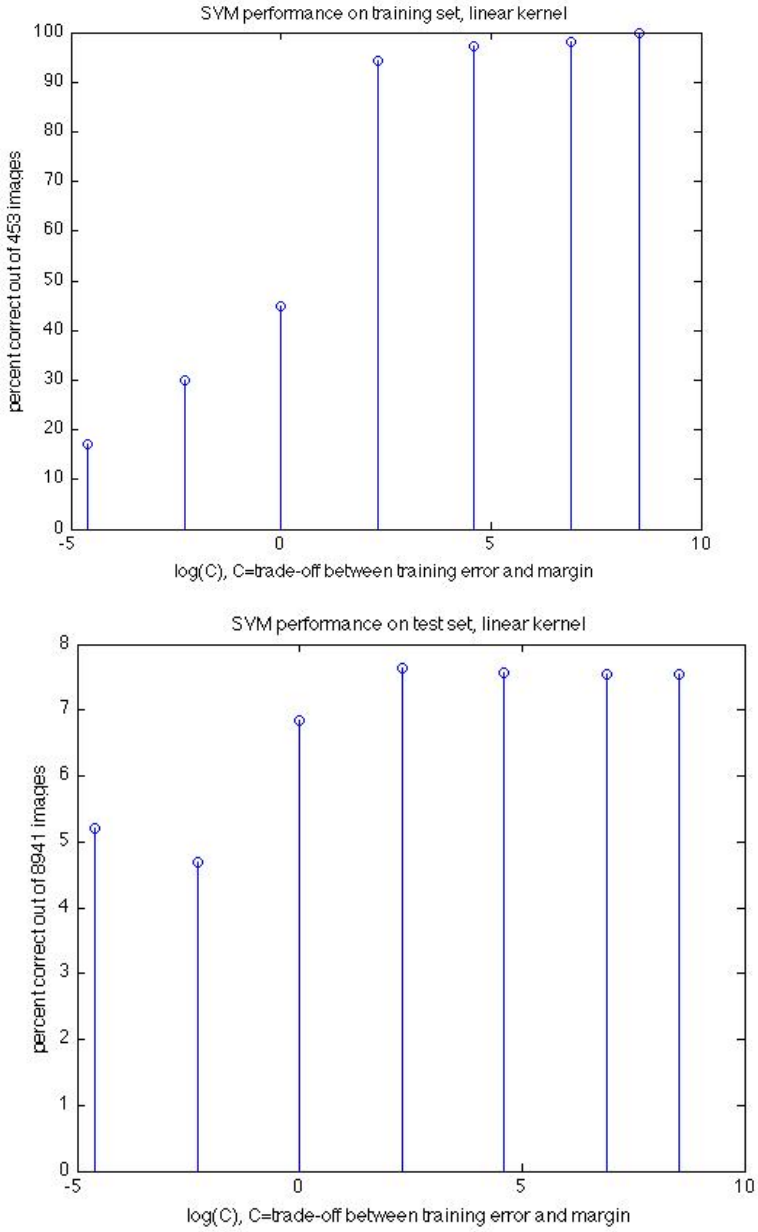


Figure 4: Top, support vector machine linear kernel performance on training set. Bottom, support vector machine linear kernel performance on test set.

Running the support vector machine polynomial kernel took several hours. As a result, only a single c value was tested, c=.01. On the training set, %100 accuracy was achieved. On the test set, %8.6 accuracy was achieved. This outperformed all linear kernel tests.

Testing HTM algorithm variants

To understand the tests I did, it is necessary to understand the structure and function of HTM nodes in a network. A general description of this is provided in the following section. Table below illustrates the tests I performed an HTM network for solving the classification problem. To facilitate these tests I implemented a version of the HTM algorithm in java. My implementation is as described in:

Zeta1 Algorithms Reference Version 1.5

In the table below, SP stands for spatial pooler, TP stands for temporal pooler. For the spatial pooler, a 0 entry indicates Gaussian learning was used, dot learning was used otherwise. For the temporal pooler, a 0 entry indicates sum learning was used, max learning was used otherwise. The level number indicates which level in the network, in total there are three levels of nodes. The top node performs the final classification and always uses the max algorithm. For a description of these algorithms please refer to the reference above.

	Level 1 SP	**Level 1 TP**	**Level 2 SP**	**Level 2 TP**	**Level 3 TP**	**Level 3 Mapper**
Test 1	**0**	**0**	**0**	**0**	**0**	**0**
Test 2	**0**	**1**	**0**	**0**	**0**	**0**
Test 3	**0**	**1**	**1**	**0**	**0**	**0**
Test 4	**0**	**1**	**1**	**1**	**0**	**0**
Test 5	**0**	**1**	**1**	**1**	**1**	**0**
Test 6	**0**	**0**	**1**	**1**	**0**	**0**
Test 7	**0**	**0**	**1**	**1**	**1**	**0**

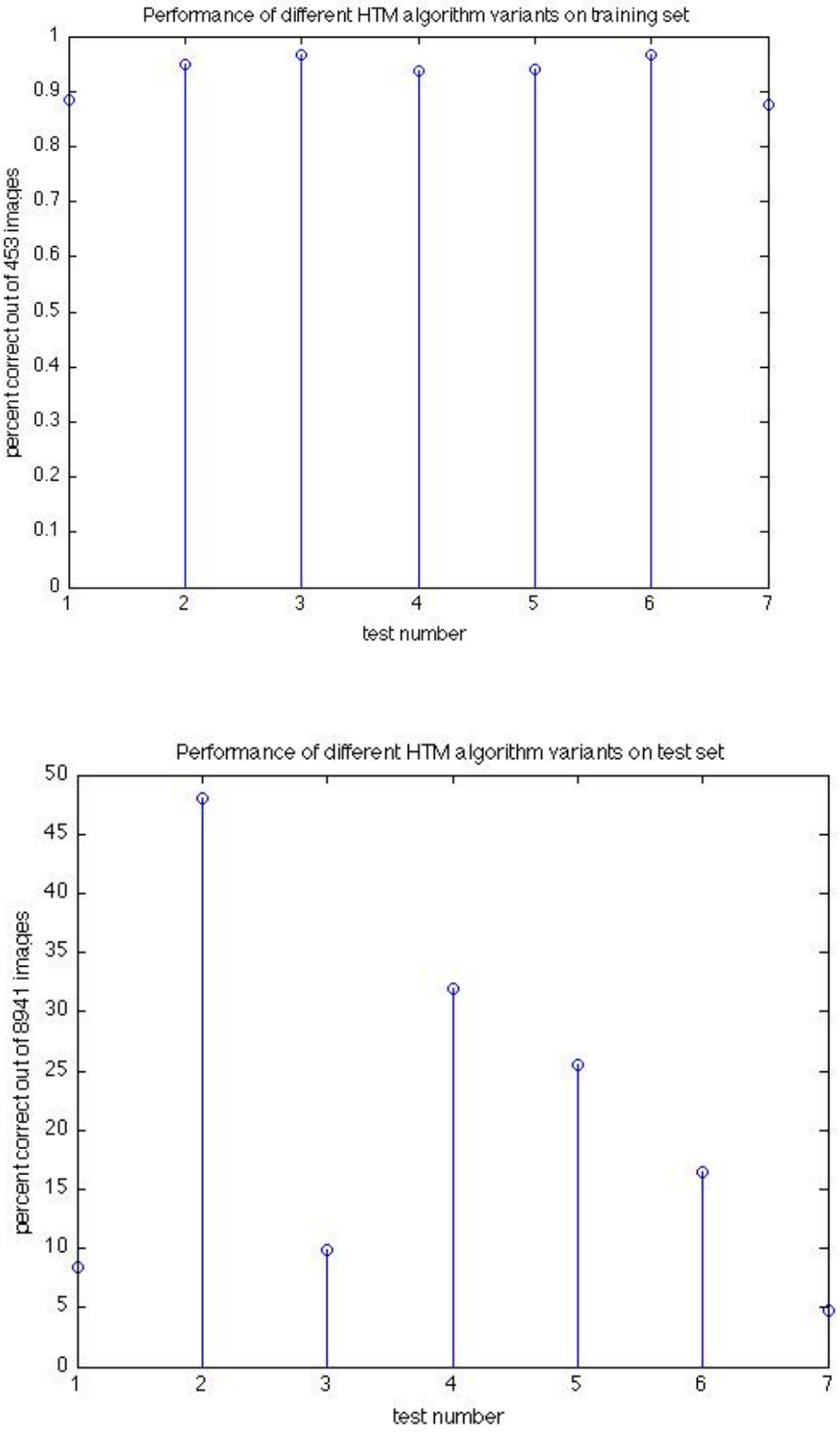


Figure 5: Accuracy of HTM algorithm variants.

Conclusion

With the HTM configuration specified in test 2, I was able to achieve about %50 accuracy, which is significantly better than the baseline algorithms. Unfortunately, I was unable to achieve the %66 percent accuracy achieved by the original implementation. This is in part due to the fact that I was unable to identify all the parameters used in the original implementation to achieve %66 accuracy. In addition, the algorithm was difficult to implement, so my implementation may contain minor bugs. However, I did gain a much deeper understanding of how the HTM algorithm works by actually implementing it. With a working implementation at my disposal, I will be able to pursue more interesting classification problems.

Appendix

The HTM Learning Algorithm

At a high level, the HTM learning algorithm consists of several nodes connected together in a hierarchical manner. A typical network has many nodes at the bottom, where training data is received, and a decreasing number of nodes at higher levels. This is illustrated in figure 2.

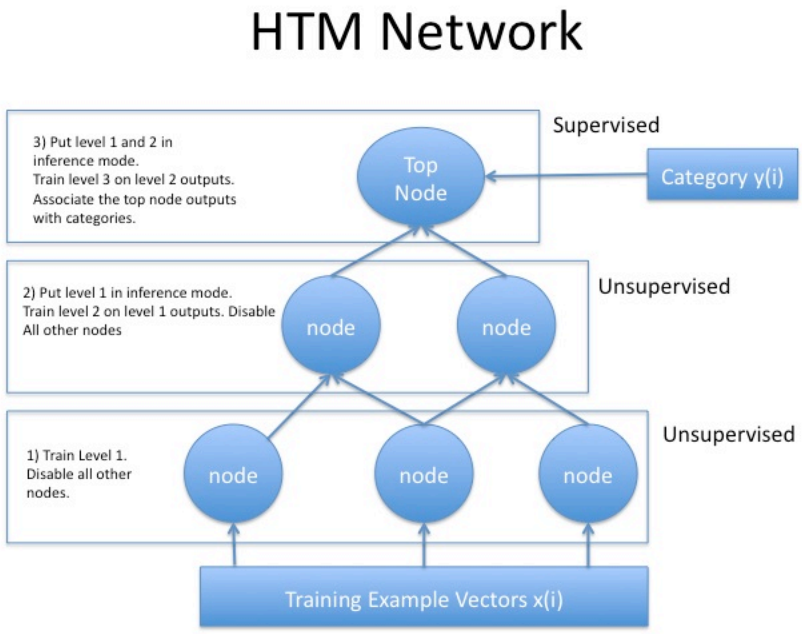


Figure 2.

There are essentially two types of nodes in the network, “Zeta1TopNodes”, which are always at the top of the network, and “Zeta1Nodes”, which make up the rest of the network. (From here on, I’ll refer to these as TopNodes and Nodes, respectively). Each node is can be in either of two modes, learning or inference mode. HTM networks are trained **one** level at a time from bottom to top. As figure 2 indicates, at the start of network training, the lowest level is placed into learning mode until it has seen all the training data. During this time, all other nodes are disabled. The next level is then trained on the inferences of the first level as it is given training data. This process is repeated until the TopNode is reached. The TopNode is the only node in the network which performs supervised learning, all other nodes perform unsupervised learning. When the TopNode is learning, all other nodes are

performing inference. For each training input at the lowest level, the TopNode is given a category to associate its internal representations of the data with.

Internals of a Node

The main internal components of the node are what the authors call "spatial" and "temporal poolers." The inputs to a node are vectors from its children (or training data if it is a bottom node). The operation of the each pooler is different during learning and inference.

Spatial Pooler

In learning mode, the inputs from each child are concatenated and given to the internal spatial pooler. The exact operation of the spatial pooler can vary (you can choose from several algorithms) but all have the same purpose, to map inputs to a discrete number of bins the authors call "coincidences." Coincidences are intended to represent the real-world causes of the data. For instance, a real-world cause could be an edge of some object in an image. This process significantly reduces the dimension of the input data. At the end of a node's learning, the coincidences (which are vectors themselves) are concatenated to form the "coincidence matrix." In addition, the spatial pooler maintains a count of the number of times it observed each particular coincidence. This count is passed to the temporal pooler, discussed later.

In inference mode, the spatial pooler takes an input and generates a belief distribution over the coincidences it observed during learning. The authors stress that this belief distribution is not an actual probability but only a general measure of the likelihood that the input has the same cause as one of its learned coincidences.

Temporal Pooler

During learning, the temporal pooler groups together coincidences that are frequently adjacent to each other in time. To do this, the temporal pooler is told what coincidence is observed at each time step of learning. The length of sequences to be remembered before a new sequence starts is a node parameter that must be specified. At the end of a sequence length, a "time adjacency matrix" is updated. The time adjacency matrix stores an integer proportional to the number of times a transition was observed between pairs of coincidences. The increment to the matrix decreases linearly for coincidences that occur further back in time. The learning process essentially ends with the formation of the time adjacency matrix. However, just before the node switches to inference mode, the temporal pooler generates groups of coincidences. These groups are formed using the time adjacency matrix and coincidence frequency from the spatial pooler (mentioned earlier).

In inference mode, the temporal pooler receives belief distributions over coincidences from the spatial pooler. The temporal pooler uses this, the time adjacency matrix, and weight matrix, to produce a distribution over groups of coincidences.

Operation of a TopNode

The TopNode only contains a spatial pooler, no temporal pooler.

When learning, a TopNode receives a coincidence from its spatial pooler and a category from the data. This information is used to form a "mapping matrix" that counts the number of times a particular coincidence was associated with a category.

When performing inference, the TopNode uses the belief distribution over coincidences from its spatial pooler and mapping matrix to produce a distribution over categories.

Java HTM Code

```
public class Zeta1Node {
	public SpatialPooler sp=null;
	public TemporalPooler tp=null;

	Zeta1Node(int SpatialPoolerAlgorithm,
			double MaxDistance,double sigma,int transitionMemory,
			boolean symetricTime,int maxGroupSize, boolean detectBlanks,
			int topNeighbors,int temporalPoolerAlgorithm,boolean
overLappingGroups){

		this.sp=new SpatialPooler(SpatialPoolerAlgorithm,MaxDistance,sigma);
		this.tp=new TemporalPooler(transitionMemory,
maxGroupSize,topNeighbors,temporalPoolerAlgorithm,overLappingGroups);
	}

	public void trainSpatialPooler(double[] in,int numChildren){
		sp.processInputVector(in,numChildren);
	}

	public void setTemporalPoolerCounts(){
		tp.setCounts(sp.counts);
	}

	public void trainTemporalPooler(double[] in,int numChildren){
		int c;
		c=sp.getCoincidenceIndex(in, numChildren);
		tp.updateTAM(c);
	}

	public void setInferenceMode(){
		tp.formGroups();
	}

	public double[] inference(double[] in){
		double [] sp_out=new double[sp.counts.length];
		sp.computeOutputDistribution(in, sp_out);
		return tp.processInput(sp_out);
	}
}

public class Zeta1Node {
	public SpatialPooler sp=null;
	public TemporalPooler tp=null;
```

```
        Zeta1Node(int SpatialPoolerAlgorithm,
                        double MaxDistance,double sigma,int transitionMemory,
                        boolean symetricTime,int maxGroupSize, boolean detectBlanks,
                        int topNeighbors,int temporalPoolerAlgorithm,boolean
overLappingGroups){

                this.sp=new SpatialPooler(SpatialPoolerAlgorithm,MaxDistance,sigma);
                this.tp=new TemporalPooler(transitionMemory,
maxGroupSize,topNeighbors,temporalPoolerAlgorithm,overLappingGroups);
        }

        public void trainSpatialPooler(double[] in,int numChildren){
                sp.processInputVector(in,numChildren);
        }

        public void setTemporalPoolerCounts(){
                tp.setCounts(sp.counts);
        }

        public void trainTemporalPooler(double[] in,int numChildren){
                int c;
                c=sp.getCoincidenceIndex(in, numChildren);
                tp.updateTAM(c);
        }

        public void setInferenceMode(){
                tp.formGroups();
        }

        public double[] inference(double[] in){
                double [] sp_out=new double[sp.counts.length];
                sp.computeOutputDistribution(in, sp_out);
                return tp.processInput(sp_out);
        }
}
////////////////////////////////////////////////////////////////


public class SpatialPooler {

        public int spatialPoolerAlgorithm=0;
        public double MaxDistance=1.0; //only used with the gaussian spatial pooler
        public double sigma=1.0; //only used with the gaussian spatial pooler
        public MyMatrix W; //the coincidence matrix
        public int[] counts;

        public SpatialPooler(int spatialPoolerAlgorithm,double MaxDistance, double
sigma){
                this.spatialPoolerAlgorithm=spatialPoolerAlgorithm;
                this.MaxDistance=MaxDistance;
                this.sigma=sigma;
        }

        public void setSpatialPoolerAlgorithm(int a){
                this.spatialPoolerAlgorithm=a;
        }

        public void processInputVector(double[] inputVector,int numChildren){
                if(spatialPoolerAlgorithm==0) //gauss
```

```
                gaussianLearning(inputVector);
        else if(spatialPoolerAlgorithm==1) //dot
                DotOrProductLearning(inputVector,numChildren);
        else if(spatialPoolerAlgorithm==2) //product
                DotOrProductLearning(inputVector,numChildren);
        else if(spatialPoolerAlgorithm==3) //max dot
                maxDotOrProductLearning(inputVector,numChildren);
        else if(spatialPoolerAlgorithm==4) //max product
                maxDotOrProductLearning(inputVector,numChildren);
        else
                System.out.println("algo error");

}

public void gaussianLearning(double[] inputVector){
        if(W==null){ //this is the first coincidence
                W=new MyMatrix(inputVector,true);
                counts=new int[1];
                counts[0]=1;
                return;
        }

        boolean novelVector=true;
        int numRows=W.getNumRows();
        double[] thisCoincidence;
        double dist;
        for(int j=0;j<numRows;j++){
                thisCoincidence=W.getRow(j);
                dist=getDistance(thisCoincidence,inputVector);
                if(dist<=MaxDistance){
                        counts[j] +=1;
                        novelVector=false;
                        break;
                }
        }

        if(novelVector){
                W.appendRow(inputVector);

                int[] tmpCounts=counts.clone();
                counts=new int[tmpCounts.length+1];

                for(int i=0;i<tmpCounts.length;i++)
                        counts[i]=tmpCounts[i];

                counts[tmpCounts.length]=1;
        }

}

public void DotOrProductLearning(double[] inputVector,int numChildren){
        double[] updated=new double[inputVector.length];

        winnerTakeAllUpdate(inputVector,numChildren,updated);


        if(W==null){ //this is the first coincidence
                W=new MyMatrix(updated,true);
                counts=new int[1];
                counts[0]=1;
```

```
                return;
        }

        boolean novelVector=true;
        int numRows=W.getNumRows();
        double[] thisCoincidence;

        for(int j=0;j<numRows;j++){
                thisCoincidence=W.getRow(j);
                if(CheckEqual(thisCoincidence,updated)){
                        counts[j] +=1;
                        novelVector=false;
                        break;
                }
        }

        if(novelVector){
                W.appendRow(updated);

                int[] tmpCounts=counts.clone();
                counts=new int[tmpCounts.length+1];

                for(int i=0;i<tmpCounts.length;i++)
                        counts[i]=tmpCounts[i];

                counts[tmpCounts.length]=1;
        }

}


public void maxDotOrProductLearning(double[] inputVector,int numChildren){
        double[] updated=new double[inputVector.length];

        winnerTakeAllUpdate(inputVector,numChildren,updated);


        if(W==null){ //this is the first coincidence
                W=new MyMatrix(updated,true);
                counts=new int[1];
                counts[0]=1;
                return;
        }

        boolean novelVector=true;
        int numRows=W.getNumRows();
        double[] thisCoincidence;

        for(int j=0;j<numRows;j++){
                thisCoincidence=W.getRow(j);
                if(hammingDistance(thisCoincidence,updated)<=MaxDistance){
                        counts[j] +=1;
                        novelVector=false;
                        break;
                }
        }

        if(novelVector){
                W.appendRow(updated);
```

```
                        int[] tmpCounts=counts.clone();
                        counts=new int[tmpCounts.length+1];

                        for(int i=0;i<tmpCounts.length;i++)
                                counts[i]=tmpCounts[i];

                        counts[tmpCounts.length]=1;
                }

        }
        public void winnerTakeAllUpdate(double[] input, int numChildren,double[]
output){
                int childInputLength=input.length/numChildren;
                double[][] childInput=new double[numChildren][childInputLength];

                for(int j=0; j<numChildren;j++){
                        for(int i=0; i<childInputLength; i++){
                                childInput[j][i]=input[j*childInputLength+i];
                        }
                }

                int maxInd;
                for(int j=0; j<numChildren;j++){
                        maxInd=maxIndex(childInput[j]);
                        for(int i=0; i<childInputLength; i++){
                                if(i==maxInd)
                                        output[j*childInputLength+i]=1.0;
                                else
                                        output[j*childInputLength+i]=0.0;

                        }
                }

        }

        public int maxIndex(double[] input){
                int ind=0;
                double max=Double.MIN_VALUE;

                for(int i=0; i<input.length;i++){
                        if(input[i]>max){
                                max=input[i];
                                ind=i;
                        }
                }

                return ind;
        }

        public double getDistance(double[] a,double[] b){
                double sqrdDistance=0.0;
                double diff;
                for(int i=0; i<a.length;i++){
                        diff=a[i]-b[i];
                        sqrdDistance+=diff*diff;
                }

                return sqrdDistance;

        }
```

```
public void setMaxDistance(double d){
	this.MaxDistance=d;
}

public void setSigma(double s){
	this.sigma=s;
}

public String toString() {
	String s="Coincidences Matrix: \n";

	s+=toString(W.m);



	s+="Counts: \n";

	for(int i=0;i<counts.length;i++){
		s+=counts[i]+ " ";
	}

	return s;
}

public String toString(double[][] m) {
	//print matrix
	String s="";
	for(int i=0;i<m.length;i++){
		for(int j=0; j<m[0].length;j++){
			s+=m[i][j] + " ";
		}
		s+="\n";
	}

      return s;
}

public boolean CheckEqual(double[] a,double[] b){
	for(int i=0; i<a.length;i++){
		if(a[i] !=b[i])
			return false;
	}

	return true;
}

public double hammingDistance(double[] a,double[] b){
	double ret=0.0;
	for(int i=0; i<a.length;i++){
		if(a[i] !=b[i])
			ret+=1.0;
	}

	return ret;
}

public void computeOutputDistribution(double[] input, double[] output){

	if(spatialPoolerAlgorithm==0) //gauss
```

```
                gaussOutputDistribution(input,output);
        else if(spatialPoolerAlgorithm==1) //dot
                dotOutputDistribution(input,output);
        else if(spatialPoolerAlgorithm==2) //product
                productOutputDistribution(input,output);
        else if(spatialPoolerAlgorithm==3) //max dot
                dotOutputDistribution(input,output);
        else if(spatialPoolerAlgorithm==4) //max product
                productOutputDistribution(input,output);
        else
                System.out.println("algo error");

}

public void gaussOutputDistribution(double[] input, double[] output){
        double power;

        for(int i=0; i<output.length;i++)
                output[i]=0.0;

        for(int i=0; i<output.length;i++){
                power=-getDistance(input,W.getRow(i));
                power=power/(2.0*sigma*sigma);
                output[i]=Math.exp(power);
        }
}

public void dotOutputDistribution(double[] input, double[] output){
        double[] thisCoincidence;

        for(int i=0; i<output.length;i++)
                output[i]=0.0;

        for(int i=0; i<output.length;i++){ //each coincidence
                thisCoincidence=W.getRow(i);
                for(int j=0; j<thisCoincidence.length;j++){
                        if(thisCoincidence[j]==1.0)
                                output[i]+=input[j];
                }
        }
}

public void productOutputDistribution(double[] input, double[] output){
        double[] thisCoincidence;

        for(int i=0; i<output.length;i++)
                output[i]=1.0;

        for(int i=0; i<output.length;i++){ //each coincidence
                thisCoincidence=W.getRow(i);
                for(int j=0; j<thisCoincidence.length;j++){
                        if(thisCoincidence[j]==1.0)
                                output[i]*=input[j];
                }
        }
}

public int getCoincidenceIndex(double[] input,int numChildren){
        if(spatialPoolerAlgorithm==0)
                return gaussCIndex(input);
```

```
                else
                        return dotOrProductCIndex(input,numChildren);
        }


        public int gaussCIndex(double[] input){

                int numRows=W.getNumRows();
                double[] thisCoincidence;
                double dist;

                for(int j=0;j<numRows;j++){
                        thisCoincidence=W.getRow(j);
                        dist=getDistance(thisCoincidence,input);
                        if(dist<=MaxDistance){
                                return j;
                        }
                }

                return -1;
        }

        public int dotOrProductCIndex(double[] input, int numChildren){
                double[] updated=new double[input.length];
                winnerTakeAllUpdate(input,numChildren,updated);

                int numRows=W.getNumRows();
                double[] thisCoincidence;

                for(int j=0;j<numRows;j++){
                        thisCoincidence=W.getRow(j);
                        if(CheckEqual(thisCoincidence,updated)){
                                return j;
                        }
                }

                return -1;
        }


}
import java.util.Arrays;


public class TemporalPooler {

        public int transitionMemory=4;
        public int maxGroupSize=10;
        public int topNeighbors=2;
        public int numCoincidences=4;
        public int temporalPoolerAlgorithm=0;

        public boolean symetricTime=false;
        public boolean detectBlanks=false;
        public boolean overLappingGroups=false;

        public int[][] groups=null;
        public int[] counts=null;
        public double[][] weightsMatrix=null;
        public int[] mem=null; //the transition memory
```

```
        public int[][] TAM=null; //time adjacency matrix
        public int memFillState=0;

        public TemporalPooler(int transitionMemory,
                        int maxGroupSize,int topNeighbors,int
temporalPoolerAlgorithm,boolean overLappingGroups){

                this.transitionMemory=transitionMemory;
                mem=new int[transitionMemory];
                this.maxGroupSize=maxGroupSize;
                this.topNeighbors=topNeighbors;
                this.temporalPoolerAlgorithm=temporalPoolerAlgorithm;
                this.overLappingGroups=overLappingGroups;

        }


        public void setTAM(int[][] TAM){this.TAM=TAM.clone();}

        public void setCounts(int[] counts){
                this.counts=new int[counts.length];
                this.numCoincidences=counts.length;
                copy(counts,this.counts);
                this.TAM=new int[numCoincidences][numCoincidences];
        }
        public void setTransitionMemory(int t){ this.transitionMemory=t;}

        public void updateTAM(int coincidenceIndex){
                /*
                For each of these past coincidences, the pooler
                increments the value in the time-adjacency matrix
                corresponding to a transition from the past coincidence
                to the current coincidence. The amount by which
                the entries are incremented falls off linearly
                with time.
                */
                //System.out.println("mem before: " +toString(mem));
                if(memFillState< this.transitionMemory) {
                        updateTMem(coincidenceIndex);
                        //System.out.println("mem coincidence: "+coincidenceIndex);
                        //System.out.println("mem after: " +toString(mem));
                        return;
                }

                for(int i=0;i<mem.length;i++){
                        TAM[coincidenceIndex][mem[i]]+=transitionMemory-i;
                }

                //System.out.println("TAM: \n" +toString(TAM));
                updateTMem(coincidenceIndex);
        //      System.out.println("mem coincidence: "+coincidenceIndex);
        //      System.out.println("mem after: " +toString(mem));
        }

        public void updateTMem(int coincidenceIndex){
                //push a new coincidence onto the coincidence memory,
                //throw away the oldest coincidence.

                if(memFillState< this.transitionMemory)
                        this.memFillState++;
```

```
                int[] tmp=new int[mem.length];
                for(int i=1; i<mem.length;i++){
                        tmp[i]=mem[i-1];
                }
                tmp[0]=coincidenceIndex;
                mem=tmp;

        }

        public void formGroups(){
                //forms groups from the TAM and counts matrices.


                int[] counts=this.counts.clone();
                int mostFrequentCoincidence;

            int[][] TAMcopy=new int[TAM.length][TAM.length];
            copy(TAM,TAMcopy);
            int[] group;
            int[] countsCopy=new int[counts.length];
            copy(counts,countsCopy);

            //Finds the most frequent coincidence that is not yet part of a group.
            //The most frequent coincidence is the one with the highest corresponding
value
            //in the counts vector (maintained by the spatial pooler).
            while(!doneFormingGroups()){
                group=null;
                mostFrequentCoincidence=maxIndex(countsCopy);

        group=findTopNeighbors(mostFrequentCoincidence,TAMcopy,group,false,false);
                //System.out.println("formed: "+toString(group));
                updateCounts(group,countsCopy);
                addGroup(group);
            }

            if(overLappingGroups)
                growGroups();

            createWeightsMatrix();

        }

        public int[] findTopNeighbors(int coincidence,int[][] updatedTAM,int[]
group,boolean recursive,boolean grow){
                //Picks the coincidences that are the most-connected to this coincidence.
                //System.out.println("root: "+ coincidence);
                group=addCoincidence(group,coincidence,grow);
                updatedTAM[coincidence][coincidence]=Integer.MIN_VALUE; //prevent adding
self to group
                int[] connectedCoincidences;
                int mostConnected;
                for(int i=0; i<topNeighbors;i++){
                        //The pooler finds the most-connected coincidences by
                        //finding the highest values in the row of the time-adjacency
                        //matrix that corresponds to the current coincidence

                        connectedCoincidences=updatedTAM[coincidence];
                        //System.out.println("connected:
```

```
"+toString(connectedCoincidences));
                        mostConnected=maxIndex(connectedCoincidences);
                        //System.out.println("most connected: "+mostConnected);
                        if(connectedCoincidences[mostConnected]>0)
                                updatedTAM[coincidence][mostConnected]=Integer.MIN_VALUE;

                        if(!isInGroup(mostConnected,group,grow) ){
                                group=addCoincidence(group,mostConnected,grow);
                                if(recursive)

        group=findTopNeighbors(mostConnected,updatedTAM,group,true,false);
                        }
                }

                if(!grow){
                        if(!recursive){
                                //Then recursively computes step 2 on their neighbors,
                                //and so on, until no new coincidences are added.
                                int leng=group.length;
                                for(int i=1; i<leng;i++){

        group=findTopNeighbors(group[i],updatedTAM,group,true,false);
                                }
                        }
                }
                return group;
        }

        public int maxIndex(int[] input){
                int ind=0;
                int max=Integer.MIN_VALUE;

                for(int i=0; i<input.length;i++){
                        if(input[i]>max){
                                max=input[i];
                                ind=i;
                        }
                }

                return ind;
        }

        public int maxIndex(double[] input){
                int ind=0;
                double max=Double.MIN_VALUE;

                for(int i=0; i<input.length;i++){
                        if(input[i]>max){
                                max=input[i];
                                ind=i;
                        }
                }

                return ind;
        }

        public int[] addCoincidence(int[] group,int coincidence,boolean growing){
                //add coincidence to the group, return new group

                if(isInGroup(coincidence,group,growing)) //coincidence is already in the
```

```
group, dont add it to the group
                        return group;

                if(group==null){
                        group=new int[1];
                        group[0]=coincidence;
                        return group;
                }
                else{
                        if(group.length+1<=maxGroupSize){

                                int[] tmp=new int[group.length+1];
                                for(int i=0;i<group.length;i++){
                                        tmp[i]=group[i];
                                }

                                tmp[group.length]=coincidence;
                                group=tmp;
                                return group;

                        }
                        else{
                                System.err.println("exceeded max group size");
                                return group;
                        }
                }
        }

        public boolean isInGroup(int coincidence, int[] newGroup,boolean growing){
                //return true if the coincidence is in a group
                if(!growing){
                        if(groups!=null){ //is the coincidence is a group that was
previously created?
                                for(int i=0; i<groups.length;i++){
                                        for(int j=0;j<groups[i].length;j++){
                                                if(groups[i][j]==coincidence)
                                                        return true;
                                        }
                                }
                        }
                }

                if(newGroup !=null){ //is the coincidence in the group currently being
formed?
                        for(int j=0;j<newGroup.length;j++){
                                if(newGroup[j]==coincidence)
                                        return true;
                        }
                }

                return false;
        }

        public boolean doneFormingGroups(){
                //return true if all coincidences have been added to a group

                int sum=0;
                if(groups==null)
                        return false;
```

```
		for(int i=0;i<groups.length;i++)
			sum+=groups[i].length;

		if(sum==numCoincidences)
			return true;
		else
			return false;
	}

	public void addGroup(int[] group){
		//add a group to the groups list
		if(group==null){
			System.err.println("attempting to add empty group");
			return;
		}

		if(groups==null){
			groups=new int[1][group.length];
			groups[0]=group;
		}
		else{
			int[][] tmp=new int[groups.length+1][];
			for(int i=0; i<groups.length;i++){
				tmp[i]=groups[i];
			}
			tmp[groups.length]=group;
			groups=tmp;
		}
	}

	public void updateCounts(int[] group,int[] counts){
		//mark coincidences in the counts vector that have been added to group[]

		if(group==null)
			return;

		for(int i=0;i<group.length;i++){
			counts[group[i]]=Integer.MIN_VALUE;
		}
	}

	public void growGroups(){
		System.out.println("growing");
		int[][] TAMcopy=new int[TAM.length][TAM.length];

		System.out.println("old group: "+toString(TAM));
		for(int i=0; i<groups.length;i++){
			System.out.println("old group: "+toString(groups[i]));
			copy(TAM,TAMcopy);
			int leng=groups[i].length;
			for(int j=0; j<leng;j++){
				groups[i]=this.findTopNeighbors(groups[i][j], TAMcopy,
groups[i], false, true);
			}
		}
	}

	public void createWeightsMatrix(){
		weightsMatrix=new double[groups.length][counts.length];
		double[] sums=new double[groups.length];
```

```
        for(int i=0; i<groups.length;i++){
                for(int j=0; j<groups[i].length;j++){
                        sums[i]+=counts[groups[i][j]];
                }
        }

        for(int i=0; i<groups.length;i++){
                for(int j=0; j<groups[i].length;j++){

weightsMatrix[i][groups[i][j]]=counts[groups[i][j]]/sums[i];
                }
        }

}

//useful for debugging
public String toString() {
        //print groups list
        String s="Time Adjacency Matrix: \n";

        s+=toString(TAM);

        s+="Groups: \n";

        s+=toString(groups);

        s+="Trasition Memory: \n";

        s+=toString(mem);

    return s;
}

public String toString(int[][] m) {
        //print matrix
        if(m==null)
                return "null \n";

        String s="";
        for(int i=0;i<m.length;i++){
                for(int j=0; j<m[i].length;j++){
                        s+=m[i][j] + " ";
                }
                s+="\n";
        }

      return s;
}

public double[] processInput(double[] in){
        if(temporalPoolerAlgorithm==0)
                return maxProp(in);
        else if(temporalPoolerAlgorithm==1)
                return sumProp(in);

        return in;
}

public double[] maxProp(double[] in){
```

```
        double[][] inPartition=new double[groups.length][];

        for(int i=0; i<groups.length;i++){
                inPartition[i]=new double[groups[i].length];
                for(int j=0; j<groups[i].length;j++){
                        inPartition[i][j]=in[groups[i][j]];
                }
        }

        double[] out=new double[groups.length];

        for(int i=0; i<out.length;i++){
                out[i]=inPartition[i][maxIndex(inPartition[i])];
        }

        return out;
}

public double[] sumProp(double[] in){
        double[] out=new double[groups.length];

        for(int i=0; i<weightsMatrix.length;i++){
                for(int j=0; j<weightsMatrix[i].length;j++){
                        out[i]+=weightsMatrix[i][j]*in[j];
                }
        }

        return out;
}

public String toString(double[][] m) {
        //print matrix
        if(m==null)
                return "null";

        String s="";
        for(int i=0;i<m.length;i++){
                for(int j=0; j<m[0].length;j++){
                        s+=m[i][j] + " ";
                }
                s+="\n";
        }

      return s;
}

public String toString(int[] v){
        //print vector
        if(v==null)
                return "null";

        String s="";
        for(int i=0; i<v.length;i++){
         s+=v[i] +" ";
        }
        s+="\n";
        return s;
}
```

```
        public void copy(int[][] in,int[][] out){
                for(int i=0; i<in.length;i++){
                        out[i]=Arrays.copyOf(in[i], in[i].length);
                }
        }

        public void copy(int[] in,int[] out){
                out=Arrays.copyOf(in, in.length);
        }

}


public class Zeta1TopNode {

public SupervisedMapper sm;
public SpatialPooler sp;
public double[] output;

        public Zeta1TopNode(int SpatialPoolerAlgorithm,
                double MaxDistance,double sigma,int mapperAlgorithm,int numCategories){

                this.sp=new SpatialPooler(SpatialPoolerAlgorithm,MaxDistance,sigma);
                this.sm=new SupervisedMapper(mapperAlgorithm,numCategories);
        }

        public void trainSpatialPooler(double[] in,int numChildren){
                sp.processInputVector(in,numChildren);
        }

        public void setSupervisedMapperNumCoincidences(){
                sm.setNumCoincidences(sp.counts.length);
        }

        public void trainMapper(double[] in,int numChildren,int category){
                int c;
                c=sp.getCoincidenceIndex(in, numChildren);
                sm.learn(c, category);
        }

        public void setInferenceMode(){
                sm.setInferenceMode();
        }

        public void inference(double[] in){
                double[] sp_output=new double[sp.counts.length];
                sp.computeOutputDistribution(in,sp_output);
                sm.inference(sp_output);
                this.output=sm.output;
        }
}


public class Zeta1TopNode {

public SupervisedMapper sm;
public SpatialPooler sp;
public double[] output;

        public Zeta1TopNode(int SpatialPoolerAlgorithm,
```

```
                double MaxDistance,double sigma,int mapperAlgorithm,int numCategories){

                this.sp=new SpatialPooler(SpatialPoolerAlgorithm,MaxDistance,sigma);
                this.sm=new SupervisedMapper(mapperAlgorithm,numCategories);
        }

        public void trainSpatialPooler(double[] in,int numChildren){
                sp.processInputVector(in,numChildren);
        }

        public void setSupervisedMapperNumCoincidences(){
                sm.setNumCoincidences(sp.counts.length);
        }

        public void trainMapper(double[] in,int numChildren,int category){
                int c;
                c=sp.getCoincidenceIndex(in, numChildren);
                sm.learn(c, category);
        }

        public void setInferenceMode(){
                sm.setInferenceMode();
        }

        public void inference(double[] in){
                double[] sp_output=new double[sp.counts.length];
                sp.computeOutputDistribution(in,sp_output);
                sm.inference(sp_output);
                this.output=sm.output;
        }
} import java.awt.image.BufferedImage;
import java.awt.image.Raster;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;


public class HTMnetwork {
        public Zeta1TopNode level3;
        public Zeta1Node[] level2;
        public Zeta1Node[] level1;
        public int numcats=49;
        HTMnetwork(){

        }

        public void createLevel3(){
                int SpatialPoolerAlgorithm=0;
                double MaxDistance=0.0;
                double sigma=5.0;
                int mapperAlgorithm=0;
                int numCategories=numcats-1;
                level3=new Zeta1TopNode(SpatialPoolerAlgorithm,
                                MaxDistance,sigma,mapperAlgorithm,numCategories);
        }


        public void createLevel2(){
                level2=new Zeta1Node[16];
```

```
                int SpatialPoolerAlgorithm=0;
                double MaxDistance=0.0;
                double sigma=5;
                int transitionMemory=5;
                boolean symetricTime=false;
                int maxGroupSize=Integer.MAX_VALUE;
                boolean detectBlanks=false;
                int topNeighbors=3;
                int temporalPoolerAlgorithm=0;
                boolean overLappingGroups=false;
                for(int i=0;i<16;i++){
                        level2[i]=new Zeta1Node(SpatialPoolerAlgorithm,
                                        MaxDistance, sigma, transitionMemory,
                                         symetricTime, maxGroupSize, detectBlanks,
                                         topNeighbors, temporalPoolerAlgorithm,
overLappingGroups);
                }
        }


        public void createLevel1(){
                level1=new Zeta1Node[64];
                int SpatialPoolerAlgorithm=0;
                double MaxDistance=0.0;
                double sigma=5;
                int transitionMemory=5;
                boolean symetricTime=false;
                int maxGroupSize=Integer.MAX_VALUE;
                boolean detectBlanks=false;
                int topNeighbors=3;
                int temporalPoolerAlgorithm=0;
                boolean overLappingGroups=false;
                for(int i=0;i<64;i++){
                        level1[i]=new Zeta1Node(SpatialPoolerAlgorithm,
                                        MaxDistance, sigma, transitionMemory,
                                         symetricTime, maxGroupSize, detectBlanks,
                                         topNeighbors, temporalPoolerAlgorithm,
overLappingGroups);
                }
        }

        public void trainLevel1(){
                double[][] data=new double[64][16];

                String cleandir="/Users/me/nta/share/projects/pictures/data.d/clean/";
                File dir1 = new File(cleandir);
                String[] categories= dir1.list();


                /*************************************************************/
                for(int cat=1;cat<numcats;cat++){
                        File dir2= new File(cleandir+categories[cat]);
                        String[] pics = dir2.list();
                        for (int j=0; j<pics.length; j++) { //for each image in the
directory
                                int
imgOK=getImageVectors(cleandir+categories[cat]+"/"+pics[j],data); //get image vectors
                                //System.out.println(pics[j]);
                                if(imgOK>0){
                                        for(int i=0;i<64;i++){ //for each node
```

```
                                        level1[i].trainSpatialPooler(data[i], 1);
//train the spatial pooler of each node
                                        //System.out.println(children[j]+"
"+toString(data[i]));
                                }

                        }
                        else{
                                System.out.println("File not an image,skipping.");
                        }
                }
        }

        /*************************************************************/
        for(int i=0;i<64;i++){
                level1[i].setTemporalPoolerCounts(); //pass the counts vector to
the temporal pooler
        }
        /*************************************************************/
        for(int cat=1;cat<numcats;cat++){
                File dir2= new File(cleandir+categories[cat]);
                String[] pics = dir2.list();
                for (int j=0; j<pics.length; j++) { //for each image in the
directory
                        int
imgOK=getImageVectors(cleandir+categories[cat]+"/"+pics[j],data); //get image vectors
                        //System.out.println(children[j]);
                        if(imgOK>0){
                                for(int i=0;i<64;i++){
                                        level1[i].trainTemporalPooler(data[i], 1);
                                }

                        }
                        else{
                                System.out.println("File not an image,skipping.");
                        }
                }
        }
        /*************************************************************/
        for(int i=0;i<64;i++){
                level1[i].setInferenceMode();
        }

}



public int getImageVectors(String imagePath,double[][] out){

        BufferedImage img = null;

        try {
            img = ImageIO.read(new File(imagePath));
        } catch (IOException e) {
        }

        if(img==null){
                return -1;
        }
```

```
                Raster r=img.getData();
                int height=r.getHeight();
                int width=r.getWidth();

                int windowh=4;
                int windoww=4;

                int i=0;
                for(int y=0;y<height; y=y+4){
                        for(int x=0; x<width; x=x+4){
                                r.getPixels(x,y,windoww,windowh, out[i]);
                                //System.out.println(toString(out[i]));
                                i++;
                        }
                }

                return 1;

        }

        public String toString(double[] v){
                //print vector
                if(v==null)
                        return "null";

                String s="";
                for(int i=0; i<v.length;i++){
                 s+=(int)v[i] +" ";
                }
                s+="\n";
                return s;
        }

        public String toString(int[] v){
                //print vector
                if(v==null)
                        return "null";

                String s="";
                for(int i=0; i<v.length;i++){
                 s+=v[i] +" ";
                }
                s+="\n";
                return s;
        }

        public void trainLevel2(){
                double[][] data=new double[64][16];

                String cleandir="/Users/me/nta/share/projects/pictures/data.d/clean/";
                File dir1 = new File(cleandir);
                String[] categories= dir1.list();

                double[] d1,d2,d3,d4,d_concat;
                /*************************************************************/
                for(int cat=1;cat<numcats;cat++){
                        File dir2= new File(cleandir+categories[cat]);
                        String[] pics = dir2.list();
                        for (int j=0; j<pics.length; j++) { //for each image in the
directory
```

```
                                int
imgOK=getImageVectors(cleandir+categories[cat]+"/"+pics[j],data); //get image vectors
                                //System.out.println(pics[j]);
                                if(imgOK>0){
                                        for(int i=0;i<16;i++){ //for each node
                                                d1=level1[i*4].inference(data[i*4]);
                                                d2=level1[1+i*4].inference(data[1+i*4]);
                                                d3=level1[2+i*4].inference(data[2+i*4]);
                                                d4=level1[3+i*4].inference(data[3+i*4]);
                                                d_concat=concatenateVectors(d1,d2,d3,d4);
                                                level2[i].trainSpatialPooler(d_concat, 4);
//train the spatial pooler of each node
                                                //System.out.println(children[j]+"
"+toString(data[i]));
                                        }

                                }
                                else{
                                        System.out.println("File not an image,skipping.");
                                }
                        }
                }

                /*************************************************************/
                for(int i=0;i<16;i++){
                        level2[i].setTemporalPoolerCounts(); //pass the counts vector to
the temporal pooler
                }
                /*************************************************************/
                for(int cat=1;cat<numcats;cat++){
                        File dir2= new File(cleandir+categories[cat]);
                        String[] pics = dir2.list();
                        for (int j=0; j<pics.length; j++) { //for each image in the
directory
                                int
imgOK=getImageVectors(cleandir+categories[cat]+"/"+pics[j],data); //get image vectors
                                //System.out.println(children[j]);
                                if(imgOK>0){
                                        for(int i=0;i<16;i++){
                                                d1=level1[i*4].inference(data[i*4]);
                                                d2=level1[1+i*4].inference(data[1+i*4]);
                                                d3=level1[2+i*4].inference(data[2+i*4]);
                                                d4=level1[3+i*4].inference(data[3+i*4]);
                                                d_concat=concatenateVectors(d1,d2,d3,d4);
                                                level2[i].trainTemporalPooler(d_concat, 4);
                                        }

                                }
                                else{
                                        System.out.println("File not an image,skipping.");
                                }
                        }
                }
                /*************************************************************/
                for(int i=0;i<16;i++){
                        level2[i].setInferenceMode();
                }

        }
```

```
	public double[] concatenateVectors(double[] d1,double[] d2,double[] d3,double[]
d4){
		double [] out=new double[d1.length+d2.length+d3.length+d4.length];

		for(int i=0; i<d1.length;i++){
			out[i]=d1[i];
		}
		for(int i=0; i<d2.length;i++){
			out[d1.length+i]=d2[i];
		}
		for(int i=0; i<d3.length;i++){
			out[d1.length+d2.length+i]=d3[i];
		}
		for(int i=0; i<d4.length;i++){
			out[d1.length+d2.length+d3.length+i]=d4[i];
		}
		return out;
	}

	public void trainLevel3(){
		double[][] data=new double[64][16];

		String cleandir="/Users/me/nta/share/projects/pictures/data.d/clean/";
		File dir1 = new File(cleandir);
		String[] categories= dir1.list();

		double[] d1,d2,d3,d4,d_concat;
		double[] d_concat2=null;
		/*************************************************************/
		for(int cat=1;cat<numcats;cat++){
			File dir2= new File(cleandir+categories[cat]);
			String[] pics = dir2.list();
			for (int j=0; j<pics.length; j++) { //for each image in the
directory
				int
imgOK=getImageVectors(cleandir+categories[cat]+"/"+pics[j],data); //get image vectors
				//System.out.println(pics[j]);
				if(imgOK>0){
					for(int i=0;i<16;i++){ //for each node
						d1=level1[i*4].inference(data[i*4]);
						d2=level1[1+i*4].inference(data[1+i*4]);
						d3=level1[2+i*4].inference(data[2+i*4]);
						d4=level1[3+i*4].inference(data[3+i*4]);
						d_concat=concatenateVectors(d1,d2,d3,d4);

	d_concat2=concat(d_concat2,level2[i].inference(d_concat)); //train the spatial
pooler of each node
						//System.out.println(children[j]+"
"+toString(data[i]));
					}
					level3.trainSpatialPooler(d_concat2, 16);
					d_concat2=null;

				}
				else{
					System.out.println("File not an image,skipping.");
				}
			}
		}
```

```
		/*************************************************************/

		level3.setSupervisedMapperNumCoincidences();

		/*************************************************************/
		for(int cat=1;cat<numcats;cat++){
			File dir2= new File(cleandir+categories[cat]);
			String[] pics = dir2.list();
			for (int j=0; j<pics.length; j++) { //for each image in the
directory
				int
imgOK=getImageVectors(cleandir+categories[cat]+"/"+pics[j],data); //get image vectors
				//System.out.println(children[j]);
				if(imgOK>0){
					for(int i=0;i<16;i++){
						d1=level1[i*4].inference(data[i*4]);
						d2=level1[1+i*4].inference(data[1+i*4]);
						d3=level1[2+i*4].inference(data[2+i*4]);
						d4=level1[3+i*4].inference(data[3+i*4]);
						d_concat=concatenateVectors(d1,d2,d3,d4);

	d_concat2=concat(d_concat2,level2[i].inference(d_concat)); //train the spatial
pooler of each node
					}

					level3.trainMapper(d_concat2, 16, cat-1);
					d_concat2=null;

				}
				else{
					System.out.println("File not an image,skipping.");
				}
			}
		}
		/*************************************************************/

		level3.setInferenceMode();
	}

	public double[] concat(double[] A, double[] B) {
		if(A==null)
			return B;
		else if(B==null)
			return A;
		else{
			double[] C= new double[A.length+B.length];
			System.arraycopy(A, 0, C, 0, A.length);
			System.arraycopy(B, 0, C, A.length, B.length);

			return C;
		}
	}

	public void classify(){
		double[][] data=new double[64][16];
		int correct=0;
		int numimages=0;
		String cleandir="/Users/me/nta/share/projects/pictures/data.d/clean/";
		File dir1 = new File(cleandir);
		String[] categories= dir1.list();
```

```
                double[] d1,d2,d3,d4,d_concat;
                double[] d_concat2=null;
                /*************************************************************/
                for(int cat=1;cat<numcats;cat++){
                        File dir2= new File(cleandir+categories[cat]);
                        String[] pics = dir2.list();
                        for (int j=0; j<pics.length; j++) { //for each image in the
directory
                                int
imgOK=getImageVectors(cleandir+categories[cat]+"/"+pics[j],data); //get image vectors
                                //System.out.println(pics[j]);
                                if(imgOK>0){
                                        for(int i=0;i<16;i++){ //for each node
                                                d1=level1[i*4].inference(data[i*4]);
                                                d2=level1[1+i*4].inference(data[1+i*4]);
                                                d3=level1[2+i*4].inference(data[2+i*4]);
                                                d4=level1[3+i*4].inference(data[3+i*4]);
                                                d_concat=concatenateVectors(d1,d2,d3,d4);

        d_concat2=concat(d_concat2,level2[i].inference(d_concat)); //train the spatial
pooler of each node
                                                //System.out.println(children[j]+"
"+toString(data[i]));
                                        }
                                        level3.inference(d_concat2);

                                        if(level3.output[cat-1]==1.0){
                                                correct++;
                                                //System.out.println(correct+" out of " +
numimages +"images");
                                        }
                                        //System.out.println(pics[j]+"
"+toString(level3.output));

                                        numimages++;
                                        System.out.println(numimages);
                                        d_concat2=null;
                                }
                                else{
                                        System.out.println("File not an image,skipping.");
                                }
                        }
                }

                System.out.println(correct+" out of " + numimages +"images");
        }

}
```

Matlab code

```
clear all
clc
load('cleandata.mat')
load('distorteddata.mat')
numtrain=size(picVectors,1); %number fo training examples
numtest=size(picVectors_test,1); %number of test examples.
results=zeros(15,1);
c=-1;
for k=1:15
dist=zeros(numtrain,1);
```

```
correct=0;

for p=1:numtest
    for i=1:numtrain

dist(i,1)=sum(xor(picVectors(i,:),picVectors_test(p,:)));%(sum(xor(picVectors(i,:),picVec
tors_test(p,:)))+c)^10;
    end

    [dist_sorted,index]=sort(dist); %sort distances from least to greatest
    %the image in picVectors(i,:) is of category picCategory(i);

    prediction=mode(picCategory(index(1:k))); % take k nearest neighbors, find mode

    if(prediction==picCategory_test(p))
        correct=correct+1;
    end

end

results(k,1)=(correct/numtest)*100;
fprintf('k= %i ,Percent correct out of %i test images is %f \n',k, numtest,
results(k,1));


end

% k= 1 ,Percent correct out of 8941 test images is 5.211945
% k= 2 ,Percent correct out of 8941 test images is 4.641539
% k= 3 ,Percent correct out of 8941 test images is 3.813891
% k= 4 ,Percent correct out of 8941 test images is 3.813891
% k= 5 ,Percent correct out of 8941 test images is 3.735600
% k= 6 ,Percent correct out of 8941 test images is 3.579018
% k= 7 ,Percent correct out of 8941 test images is 3.634940
% k= 8 ,Percent correct out of 8941 test images is 3.679678
% k= 9 ,Percent correct out of 8941 test images is 3.646125
% k= 10 ,Percent correct out of 8941 test images is 3.455989
% k= 11 ,Percent correct out of 8941 test images is 3.154010
% k= 12 ,Percent correct out of 8941 test images is 3.008612
% k= 13 ,Percent correct out of 8941 test images is 2.863214
% k= 14 ,Percent correct out of 8941 test images is 2.986243
% k= 15 ,Percent correct out of 8941 test images is 2.885583

%polynomial kernel-same result



clear all
clc
load('cleandata.mat') %load undistorted training images
load('distorteddata.mat') %load distorted testing images
yhat=zeros(48,8941); %stores the prodictions
picVectors=double(picVectors); %convert the binary images to doubles
picVectors_test=double(picVectors_test);

%perform 48 logistic regressions, one for each category
for i=1:48
i %progress

binary_catergory_train=(picCategory==i); %call the current category being train, 1, all
other 0;

b=glmfit(picVectors,double(binary_catergory_train),'binomial','link','probit'); %binary
logistic regression
yhat(i,:)=glmval(b,picVectors_test,'probit'); %get prediction for each test image.

end

for j=1:8941 %for each test, find the category that recieved the largest yhat and take
that to be the prediction.
%this assumes 0<yhat<1 and yhat>.5 means class 1, else class 0

    [yhat_max(j),ind_max(j)]=max(yhat(:,j));
```

```
end

result=sum(ind_max==picCategory_test')/8941;

%result=.0211 , distr='binomial', link='logit'


 %result=0.0275 , distr='normal', link='identity'


 %result=0.0217 , distr='binomial', link='probit'
clear all
clc
load('cleandata.mat')
load('distorteddata.mat')
numtrain=size(picVectors,1); %number fo training examples
numtest=size(picVectors_test,1); %number of test examples.


% for m=1:numtrain
%     if(picCategory(m)==1)
%         fprintf('+1 ')
%     else
%         fprintf('-1 ')
%     end
%
%     for i=1:1024
%         if(picVectors(m,i)==0) %assume inverted image to save space in output file
%            fprintf('%i:1.0 ',i)
%         end
%     end
%     fprintf('\n')
% end


% for m=1:300
%     if(picCategory_test(m)==1)
%         fprintf('+1 ')
%     else
%         fprintf('-1 ')
%     end
%
%     for i=1:1024
%         if(picVectors_test(m,i)==0) %assume inverted image to save space in output file
%            fprintf('%i:1.0 ',i)
%         end
%     end
%     fprintf('\n')
% end
```

References

- Numenta.com-All references found here.
- Talk, HTM: Biological Mapping to Neocortex and Thalamus
- Zeta1 Algorithms Reference Version 1.5
- Advanced NuPIC Programming, Document Version 1.8.1 September 2008
- How the brain might work: a hierarchical and temporal model for learning and recognition, Dileep George June 2008
- I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun,
- Large Margin Methods for Structured and Interdependent Output Variables,

Journal of Machine Learning Research (JMLR), 6(Sep):1453-1484, 2005.
http://jmlr.csail.mit.edu/papers/volume6/tsochantaridis05a/tsochantaridis05a.pdf