Spatial Supervised Learning
Using Recurrent Sliding Window Classifiers

Dan P. Vega
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis OR 97331

June 3, 2005

Abstract

Spatial Supervised Learning seeks to learn how to assign a label to each pixel in a spatial grid such as the pixels of remote-sensed images. The standard approach is to treat each grid cell separately and to use only the measured features of the grid cell to determine the assigned label. However, spatial data usually exhibits spatial patterns so that labels of nearby grid cells are correlated. It should be possible to learn this correlation and exploit it to improve the accuracy of the predicted labels. This project studies simple recurrent sliding window classifiers for this task. Recurrent sliding window classifiers feed their predicted output labels back as inputs when making predictions for adjacent grid cells. The project shows how to generalize sequential sliding window classifiers to spatial data and also studies the effectiveness of three ensemble methods: (a) voting among sliding windows that approach a given grid cell from multiple directions, (b) bagging, and (c) boosting. These methods are applied to the C4.5 decision tree algorithm and to the naive Bayes algorithm. The results show that for naive Bayes, an increase in the input and output context results in a significant improvement over the no-context approach, and this result is improved with boosting, while bagging has little effect. For C4.5, using context delivers a 36.5% reduction in error, and when combined with boosting the reduction in error is 56.1%. Bagging also resulted in improvement for C4.5 making the greatest reduction in error, 46.3%, when only output context was included as additional features.

1 Introduction

Traditionally machine learning learned functions $y = f(x)$ from training examples of the form (x_i, y_i), where x_i is a fixed-length vector of feature values and y_i is a corresponding class label.

Over the past few years, people have begun to study structural supervised learning problems where each x_i is a graph consisting of nodes and arcs. Each node s, denoted $x_{i,s}$ is described by a vector of features, and each arc defines a relationship between two nodes. Each node s also has a corresponding class label $y_{i,s}$. Let y_i denote the labels assigned to the nodes of graph x_i. The learning problem is to learn a function F that takes a new input graph x_i and outputs a new label assignment $y_i = F(x_i)$. The function is learned by analyzing N training examples $(x_i, y_i)_{i=1}^{N}$.

The simplest example of this is sequential supervised learning, where each x_i is a simple sequence of objects (e.g., the words in a sentence) and each y_i is a sequence of class labels (e.g., the parts of speech). In sequential supervised learning, many complex methods have been studied. However, one of the simplest, recurrent sliding windows, is very fast to train and gives very good performance.

1.1 Simple and Recurrent Sliding Windows

The technique employed by simple and recurrent sliding windows is to convert the input and output sequences into a set of windows [12]. This is shown in Table 1. The window is centered on x_i, and the amount of context around x_i is defined by the left input context (LIC) and right input context (RIC). In Table 1, LIC=RIC=3. The beginning and ending are padded with a null character, in this case the '$-$' character.

Simple Sliding Windows

(X,Y)	$x_1\ x_2\ x_3\ x_4\ x_5\ x_6$	$y_1\ y_2\ y_3\ y_4\ y_5\ y_6$
w_1	$-\ -\ -\ x_1\ x_2\ x_3\ x_4$	y_1
w_2	$-\ -\ x_1\ x_2\ x_3\ x_4\ x_5$	y_2
w_3	$-\ x_1\ x_2\ x_3\ x_4\ x_5\ x_6$	y_3
w_4	$x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ -$	y_4
w_5	$x_2\ x_3\ x_4\ x_5\ x_6\ -\ -$	y_5
w_6	$x_3\ x_4\ x_5\ x_6\ -\ -\ -$	y_6

Recurrent Sliding Windows

(X,Y)	$x_1\ x_2\ x_3\ x_4\ x_5\ x_6$	$y_1\ y_2\ y_3\ y_4\ y_5\ y_6$
w_1	$-\ -\ -\ x_1\ x_2\ x_3\ x_4\ -\ -$	y_1
w_2	$-\ -\ x_1\ x_2\ x_3\ x_4\ x_5\ -\ y_1$	y_2
w_3	$-\ x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ y_1\ y_2$	y_3
w_4	$x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ -\ y_2\ y_3$	y_4
w_5	$x_2\ x_3\ x_4\ x_5\ x_6\ -\ -\ y_3\ y_4$	y_5
w_6	$x_3\ x_4\ x_5\ x_6\ -\ -\ -\ y_4\ y_5$	y_6

Table 1: Simple and Recurrent Sliding Windows

If the sequential relationship were completely contained in the x_i's, then simple sliding windows would be sufficient in capturing the sequential nature contained in the data. But, often there is sequential information contained in the y_i's, that is not present in the x_i's or, possibly, it is easier to capture the relationship between the y_i's. An illustration showing a relationship between y_i's can be found in part-of-speech tagging. In part-of-speech tagging, the sequence of y_i's is constrained by the rules of grammar. Therefore, y_{i-1} constrains the possible labels for y_i, and adding more context may help in capturing farther reaching relationships. In recurrent sliding windows, some y_i's are included in X, and this can be seen in Table 1, under "Recurrent Sliding Windows". The number of y_i's to include in X is defined by the left output context (LOC) or the right output context (ROC). ROC or LOC, but not both, determine the number of y_i's to include in each X. Which one, (LOC or ROC) depends on the direction the sequence is traversed during classification time. In Table 1, LOC=2, and ROC=0, that is, each X_i includes $\hat{y}_{i-2}$ and $\hat{y}_{i-1}$, which are to the left of y_i.

Another instance of structural supervised learning arises when each x_i is a 2-dimensional rectangular mesh. In particular, suppose that x_i is an image. Each node in x_i is a pixel with links to its north, south, east, and west neighbors. y_i is an assignment of a class label to each pixel in the image. This problem arises in remote sensing in ecology and agriculture. In ecology, scientists often wish to assign a land cover class (e.g., forest, agriculture, urban, grassland) to each pixel in a satellite image. In agriculture, people often wish to predict what crop is growing at each pixel location. The features describing each pixel typically include passively-collected information (e.g., reflected sunlight in various frequency bands) and actively-collected information (e.g., synthetic aperture radar readings).

A particular example of this is the Feltwell data set [9]. This data set consists of 15 images (each on a different band), and 1 ground truth image. Thus, we have a dataset where each node $x_{i,s}$ is a vector with 15 features, and each $y_{i,s}$ is either labeled or unknown. Nine of the bands of this dataset were gathered using Synthetic Aperture Radar (SAR), while the remaining six bands were gathered using an Airborne Thematic Mapper (ATM). SAR data can show the coarseness of the terrain, while ATM data represents the reflectance of the ground cover in different frequency bands of the electromagnetic spectrum. These agricultural fields are located in the Feltwell (UK) region. The fields are made up of the following: sugar beets, stubble, bare soil, potatoes, carrots, and unknown. As you can imagine, bare soil and stubble may be hard to differentiate. More about the dataset will be discussed in section 3.1.

The goal of this project is to extend the recurrent-sliding window methodology to 2-D mesh graphs and evaluate its effectiveness.

2 Extending the RSW Methodology

There are three key concepts in the RSW methodology:

1. Input context
2. Output context
3. Scan order

In the sequential case, the input context, as shown in Table 1, is determined by LIC and RIC. These determine how many $x_{i,s}$'s from the left and right, respectively, to include in each X_i. Similarly, the output context, also shown in Table 1, is determined by LOC or ROC. Which one is used depends on the scan order which can be thought of as left to right (use LOC), or right to left (use ROC).

How can we generalize these concepts to the 2-D case?

The input context generalizes directly. However, it doesn't quite make sense to have a different amount of context for every direction in a 2-D mesh, for instance, we could have a north, south, east, and west input context, to name just a few. So, instead, we use a single value for determining the input context window size, and we refer to this value as IC. If, for instance, IC=3, then we have a 3 pixel by 3 pixel IC window size centered on x_i. Furthermore, a mask is applied to this window, to choose a "circular" subset of this square. By applying a mask, the context sizes increase in smaller steps. For example, using our mask, four pixels are added when IC=3, instead of eight pixels, and similarly, the output context at OC=3 is halved down from four to two. The various IC and OC values used and their corresponding masks are shown in Table 3.

The output context and the scan order are harder to generalize. Suppose we were just scanning the image from left-to-right and top-to-bottom. Then we could define the output context as some subset of the set of pixels s' that will already have $\hat{y}_{i,s'}$ values assigned to them at the time pixel s is to be classified. While using these facts, we define the output context in nearly the same manner as we did the input context. Just as we did with the input context, the output context is controlled by one variable, OC, instead of LOC and ROC. This variable, OC, determines the output context window size, just as the value of IC determines the input context window size. A mask is applied to this OC window, and those pixels in s' for which we already have $\hat{y}_{i,s'}$ values are let through the mask while the pixels yet to be classified are held back. During training time, even though all of the pixels have labels, we use the same mask as we will be forced to use during classification time. The masks used can be seen in Table 3. This makes for a tricky time bootstrapping the first classifications, and this is discussed later in the section 3.3 on handling edge pixels.

2.1 Spatial Ensembles

Previous work has shown that ensembles can radically improve the performance of sequential supervised learning methods [12] [1]. Ensembles are typically constructed by perturbing the initial learning problem, for example by drawing different subsamples of the training data or the features or by introducing output codes [4]. We have developed a new way to define ensembles: We can vary the scan order of the problem. Specifically, we achieve this scan order variation by rotating and reflecting the initial dataset.

Since our dataset is an aerial image, and since the path the plane took across these fields is arbitrary, a classifier should be able to classify pixels correctly no matter which direction the plane flies while collecting data. In order to create a classifier that is independent of orientation, data from many directions is needed to remove the directional bias in the dataset. These many directions were artificially generated by taking the original dataset and rotating and reflecting it. Thus, the original training set, see Figure 1, was used to generate an eight-fold training set by adding reflections and rotations of the original data, see Figure 2. These add to our training set in a significant way only because of the orientation of the context pixels changes with the orientation of the image. One could imagine rotating these images any arbitrary amount, and thereby generating more examples. As long as some output context is used, this is a viable technique

to generate more training examples.

This ensembling of multiple scan orders also has beneficial effects in testing our classifiers by allowing us to compute the majority vote of the predictions from each scan order. In section 4.7 on *Ensemble Effects*, this is discussed in detail.

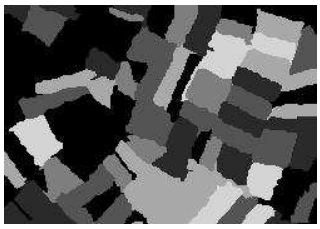


Figure 1: "Original" training set

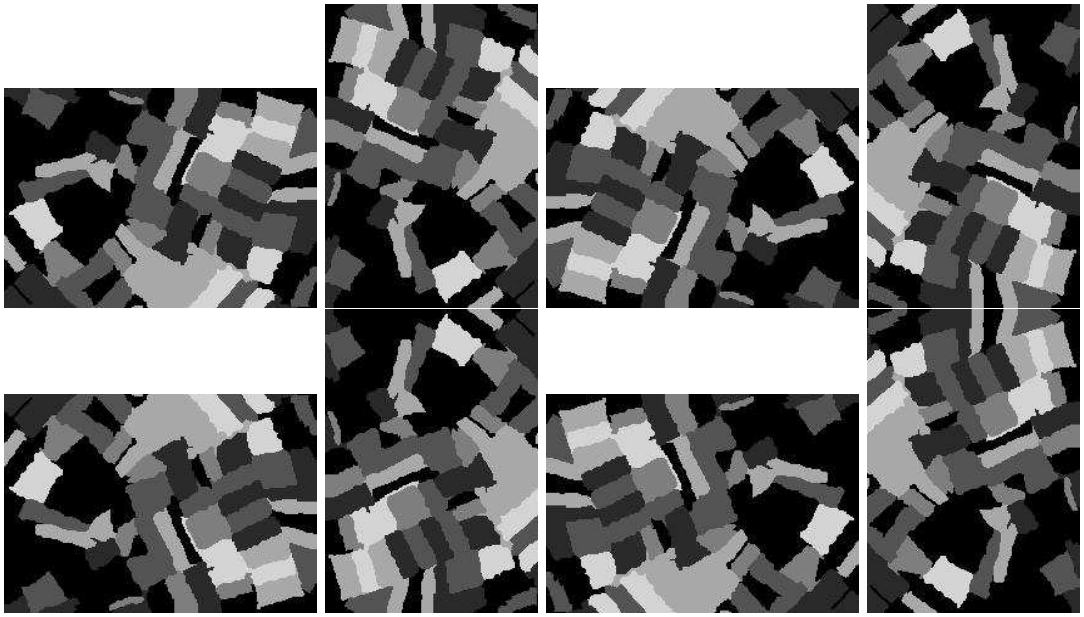


Figure 2: Eight images, the original plus seven from rotations and reflections of the original training set

2.2 Definition of Input Context and Output Context

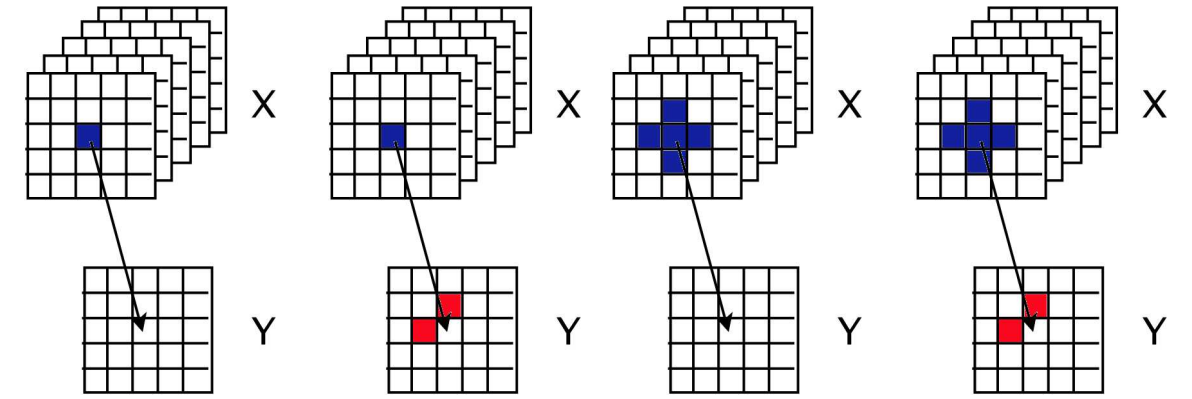
The Feltwell dataset consists of one large image, with 15 sensor readings per pixel. It is denoted as $\mathbf{X}$ in Table 2, along with a class label denoted $\mathbf{Y}$. Each image is w pixels wide. The pixels are numbered from 1 in the upper left corner to N on the lower right corner in raster order. Let us choose a pixel $\mathbf{m}$ sufficiently away from the image boundaries as to need no special handling. Thus the location of x_{m-w} is the pixel directly above x_m. Likewise x_{m+w} is directly below x_m.

There are four cases to explore. The first is when both the input context and the output context are both set to one. This is traditional supervised learning with no sliding windows. This is shown in equation (1), and the corresponding entry in Table 2. The second case is that of setting the output context window size to greater than one, while setting the input context window size to one. This is done by including the $\hat{y}_{i,s'}$ values of surrounding pixels, see equation (2). The third case is that of input context greater than one, while

output context is set at one. This is achieved by including the x data from the surrounding pixels. See equation (3) and the corresponding diagram in Table 2 which shows an example setting the input context to three. The fourth and final interesting case is when both input context and output context are greater than one. See equation (4) for an illustration of setting both the IC and OC to three.

Less of the output context is available because we must traverse the image in raster order. In this way, only pixels previously visited (during classification time) will have values available for inserting into the current pixel's feature vector. Because of this restriction, during training, we must also only include the same pixel locations. Other context sizes can be found in Table 3.

$$P(\hat{y}_m|x_m) \tag{1}$$
$$P(\hat{y}_m|x_m, \hat{y}_{m-1}, \hat{y}_{m-w}) \tag{2}$$
$$P(\hat{y}_m|x_m, x_{m-1}, x_{m+1}, x_{m+w}, x_{m-w}) \tag{3}$$
$$P(\hat{y}_m|x_m, x_{m-1}, x_{m+1}, x_{m+w}, x_{m-w}, \hat{y}_{m-1}, \hat{y}_{m-w}) \tag{4}$$



Traditional (1,1): eqn (1) OC (1,3): eqn (2) IC (3,1): eqn (3) IC and OC (3,3): eqn (4)

Table 2: Illustration of (input context, output context) equations

2.3 Input and Output Context Masks

In spatial supervised learning, the number of features grows in proportion to the square of the context size. In this project, not only are we interested in the behavior of the classifiers utilizing large contexts, but also very small contexts. If we relied solely on the size of the sliding window to determine how many pixels to include, the number of features would take very large steps. So, to take a closer look at the behavior in smaller steps, it was necessary to develop a mask within the context square to determine whether or not a pixel is included in the context. For this mask, although we considered plus shapes, squares, and vertical or horizontal swaths, in the end we chose to use a circular shaped mask. In Table 3 the mask shape and sizes can be found. Each element in a mask refers to either an $x_{i,s}$ or a $y_{i,s}$ in the case of input and output context respectively during training time. During classification time, though, this represents $x_{i,s}$ and $\hat{y}_{i,s'}$. In Table 3, there are five columns, the first being the **IC** or **OC value**. This is the height and width of the context square size, and along that row, size and shape for the IC and OC mask are shown. Also, shown is the number of features that this represents for our spatial classifier when used on the Feltwell dataset. Each mask is represented by a grid of 1's and 0's, where each element of the grid corresponds to a single pixel s. The pixels labeled '1' are included in the contexts, while the pixels labeled '0' are excluded. Each element, '1' or '0', in the **IC mask** corresponds to the features, $x_{i,s}$, of a particular node, s, which in this case is a pixel. In the Feltwell dataset, each $x_{i,s}$ is a feature vector containing 15 values. The column headed **OC mask**

IC or OC value	IC mask	OC mask	#IC features	#OC features
1	1	0	15	0
3	010 111 010	010 100 000	15 X 5	2
5	0 0 1 0 0 0 1 1 1 0 1 1 1 1 1 0 1 1 1 0 0 0 1 0 0	0 0 1 0 0 0 1 1 1 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0	15 X 13	6
7	0 0 0 1 0 0 0 0 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0 1 1 1 1 1 0 0 0 0 1 0 0 0	0 0 0 1 0 0 0 0 1 1 1 1 1 0 0 1 1 1 1 1 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	15 X 29	14
9	N/A	0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	N/A	22
11	N/A	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	N/A	30

Table 3: Input and Output Context mask sizes

is similar to the **IC mask** column. However, only those pixels which are in s', for which we already have $\hat{y}_{i,s'}$ values, can be included in the current feature vector. Although we rotate and reflect the training and testing image, we always traverse each image in raster order, that is, from left-to-right and top-to-bottom. Because of this traversal order, we can say that pixels either above or directly left of the pixel to be classified are in the set s'. These are the pixels whose $\hat{y}_{i,s'}$ values we can include in the current feature vector. During training time, these are the correct class labels, $y_{i,s}$, but during testing time, these are only predictions. Two things to note are that there are a few entries marked "N/A", and that all context sizes are odd numbers. The former is due to the fact that these context sizes are not used in this project (due to the size of the feature vector that they produce), while the latter is because we want the focus pixel to be in the center of the context window.

Joshi [12] found that the choice of the amount of input and output context had a huge impact on the quality of the results. He explained the results in terms of bias-variance theory. If the input and output contexts are too small, then the learning algorithm cannot capture the full structure of the data. However, if the contexts are too large, then the learning algorithm can easily overfit the training data. Furthermore, increasing input context can reduce the need to use output context and vice versa. He concluded that it was important to evaluate, in each application, various different combinations of input and output context sizes to determine which gives the best results.

3 A Study of RSW and Spatial Ensembles on the Feltwell Dataset

3.1 Feltwell Dataset

As mentioned earlier, the Feltwell data set [9] [10] consists of 15 images (each on a different band), and 1 ground truth image. Thus, we have a dataset where each node $x_{i,s}$ is a vector with 15 features, and each $y_{i,s}$ is either labeled or unknown. Nine of the bands of this dataset were gathered using Synthetic Aperture Radar (SAR), while the remaining six bands were gathered using an Airborne Thematic Mapper (ATM). SAR data can show the coarseness of the terrain, while ATM data represents the reflectance of the ground cover in different frequency bands of the electromagnetic spectrum. These agricultural fields are located in the Feltwell (UK) region.

The reflectance readings from each pixel for a given frequency band are collected to construct an image. Each of these images is normalized between 0 and 255. This is the number of grayscale values representable by 8 bits. Each of these images is a 250 pixel by 350 pixel grayscale image. The dataset was divided into two contiguous regions, with each being a 250 pixel by 175 pixel block. More discussion about dividing the training set can be found in section 3.1.2. This makes both the training and the test set consist of 43750 instances. Of these instances, the training set contains 13249 unknowns, while the test set has of 18594 pixels of unknown ground truth.

Class Label	Ground Cover Type	Training Set Instances	Testing Set Instances
?	Unknown	13249	18594
1	sugar beets	6959	8082
2	stubble	8702	4626
3	bare soil	3986	1266
4	potatoes	6842	5339
5	carrots	4012	5843

Table 4: Ground Truth labels for the Feltwell dataset

Each plant and soil type have a so called *spectral signature*, or spectral response curve. This signature can be used to discriminate between the different classes [15]. In this project we are training classifiers to differentiate between the five different types of plants, see Table 4, with a large number of unlabeled pixels.

3.1.1 Handling Pixels Without Labels

In this project, many pixels are of an unknown class, while, on the other hand, the features of the pixels are known for every pixel. In fact, over the entire dataset, 36% of the pixels have unknown class. In the training and test sets, 30% and 43% respectively are of an unknown class.

Although, in simple sliding windows, unknown labels are not a problem, they do become an issue in recurrent sliding windows. When we use recurrent sliding windows, the unknowns are inserted into the $x_{i,s}$ feature vectors of each node s that has a neighbor of unknown class in its output context. So, in each node, s, we have less context to learn from than we would have with a fully labeled dataset because it is sprinkled with unknowns. But, unknowns in ground truth is a common problem as the cost of obtaining ground truth for a dataset is expensive.

If we were to try to exclude all pixels of unknown ground truth from the creation of the training set, this would create a more complicated construction process. For instance, we would need to deal with questions such as, "How should a single strip of unknown pixel ground truth be handled?". Would we just collapse the data, skipping over the missing data, or something else? As it is, we still had to decide what to do with edge pixels for whom, initially "off image" context values of $x_{i,x}$ and $y_{i,s}$ are undefined. The handling of edge pixels is further discussed in section 3.3.

With all of these issues one might be led to believe that output contexts would need to be large in order to overcome the lack of information in small contexts due to unknowns, but this is simply not true. Increasing the output context from size one to size three, which only includes two more $\hat{y}_{i,s'}$ values, never lowers and more often significantly improves the accuracy of the classifier at every input context setting for all of the algorithms used in the project.

None of the algorithms that we used in this project have any problems dealing with instances with unknown labels. Although naive Bayes finds training examples with missing values useful, those examples must have a known class value. If that class value is missing, naive Bayes simply ignores those instances in its counts. Likewise, C4.5 employs the method of proportional distribution to handle training examples with missing values as long as they have a known class label [13]. So, no special handling was necessary in training the classifiers. On the other hand, there were some special considerations made in the calculation of the accuracies. Unknown pixels were not used in the calculation of the accuracy. That is, the accuracy was calculated by dividing the number of correctly labeled pixels by the total number of labeled pixels.

3.1.2 Spatially Correct Division of the Dataset

Spatial supervised learning with recurrent sliding windows adds some complication to the division of the dataset into training and testing sets. Ideally, the test set and training sets are non-overlapping and are drawn randomly without replacement from the dataset [14]. But the inclusion of context throws a bit of a wrench into this idea. For instance, should instances in the training and test sets overlap? No. But what about the contexts of those instances? Well, ideally, they shouldn't overlap either. Unfortunately, randomly selecting pixels to include in each set would severely limit the number of instances available for training and testing.

Let us illustrate this problem of random selection with the following. Imagine a map where the selection of a pixel creates a small red or blue "crater" on the map, where the blue and red represent pixels and their contexts which have been selected for the training and test set instances respectively. If different colors were not allowed to overlap at all, then instance pixels would need to be about twice the context size apart in order to be independent of each other. On the other hand, this would be less restrictive toward two pixels overlapping within the same set.

In order to deal neatly with this problem, the data set was divided in half. That is, the 250 pixel $\times$ 350 pixel dataset was divided halfway down the long end into sets, similar to how a paper would fold in half.

One half is used as the training set while the other is used for the test set and each is a 250 pixel × 175 pixel section. The distribution of the classes is fairly even, as can be seen in Table 4, while spatially the sets remained contiguous. This makes the algorithm for recurrence much more simple, as each pixel has a number and that number conveniently represents the pixel's location physically in the image.

3.2 Algorithms

Most of the algorithms used in this project, including our new spatial supervised learning meta-classifier, are implemented in Weka [17]. Weka is an open source machine learning classification tool which comes out of The University of Waikato. The Weka based algorithms include (with Weka names in parenthesis): naive Bayes (NaiveBayes), C4.5 decision trees (J48), bagging (Bagging), and boosting (Adaboost.M1) algorithms. For each of these, experiments were done while varying both the input and output context sizes. We also used bagging and boosting and measured their effect on the results. Finally, our spatial ensembling algorithm, written in Perl, is used as a post processing step to produce a final classification map.

3.2.1 Naive Bayes

Naive Bayes makes a strong assumption called *class conditional independence*. This is the assumption that feature values are conditionally independent given the class label. This, in the context of our problem suggests that the values of the neighboring pixels are independent of each other, and only depend on the class of the pixel in focus. But, we know that in an agricultural field, there is a strong likelihood that neighboring pixels will be the same plant type. As naive Bayes constructs distributions over features, then it seems that it would be constructing distributions over the *spectral signatures* of the different classes. Surprisingly, naive Bayes performs as well as C4.5 on this problem. More about the naive Bayes classifier can be found in John and Langley's paper [11].

3.2.2 Decision Trees (C4.5)

In our experiments, we use a pruned C4.5 decision tree algorithm, based on Quinlan [13]. We set the algorithm to use a confidence threshold of .25, while the minimum number of instances per leaf is set to two. This algorithm builds a classification model based on the (attribute, value) pairs of the examples in the training set. Each node in the tree splits the examples on the attribute that gives the most information on the data. The pruning helps to avoid overfitting the training data, and the confidence threshold tells the classifier which leaves to prune. The minimum number of instances per leaf keeps the tree from growing down to one instance per leaf which would essentially allow the model to memorize the training data. More information on decision trees can be found in Quinlan [13].

3.2.3 Bagging

Bagging, or Bootstrap Aggregating, is a voting or ensemble method introduced by Breiman [3]. Bagging utilizes *bootstrap samples* which are created by taking **m** instances from the training set with replacement, and in out project we use bags which are the same size as the training set. The number of these bags that we use for our experiments is 10 and 20. We stopped at 20 because there is little change between the results of 10 and 20 bags. For each bag, a classifier is constructed with the final classifier being the combined votes of individual classifiers with arbitrary tie breaking.

Breiman states that the one vital ingredient needed for bagging to help is for the classifier to be bagged to be unstable. So, for our experiments, we should expect more of an improvement from C4.5 than from naive Bayes, because C4.5 is less stable than naive Bayes. More about bagging can be found in a paper by Leo Breiman [3] and also Bauer and Kohavi's paper [2].

3.2.4 Adaboost

Adaboost or Adaptive Boosting was introduced by Freund and Schapire [7]. Adaboost generates a set of classifiers sequentially, and votes them. Weights of training instances are adaptively changed depending on the classifiers which were previously built. When, in this paper we say Adaboost (N), this refers to the number of classifiers built, N, which are then voted within the Adaboost algorithm. In this project we ran experiments with 10, 20, and 30 as the setting for the number of classifiers in Adaboost. We stopped at 30 because of the little change in the results from Adaboost (20) to Adaboost (30).

Boosting can be implemented using reweighting or resampling. In our experiments with Weka, the defaults were used for everything except the number of boosting iterations. More about Adaboost can be found in Freund and Schapire's paper [7], as well as in Bauer and Kohavi's [2] and an overview of ensembles by Dietterich [5] [4].

3.3 Handling Edge Pixels

The power of spatial sliding windows is in its ability to include the contexts of pixels. These contexts are easily constructed when the context does not extend beyond the boundary of the dataset. The idea of a boundary might not readily jump to mind, when speaking of machine learning datasets, so let us take a step back and view the dataset as a whole.

Our approach relies on the structure of the data and, in particular, on the spatial layout in the images which compose the dataset. So, when we are sliding a window across the pixels in the dataset, our window will, in the proximity of the edge pixels, extend beyond the boundaries of the image.

When this happens what should we do?

We considered four different approaches to this problem: (a) treating feature values as unknown, (b) repeating the boundary value for unknowns beyond it, (c) using a gibbs sampling technique to generate more pixels synthetically, or (d) using a reflection of the data across the boundary.

The first case, using unknowns, doesn't seem very inspired, and makes no effort to use the spatial information that we do have. The classifiers may even throw out instances with unknowns in the features. So, in the interest of having as many training examples as possible, using something other than unknowns seems desirable. The second case, where the boundary values are repeated sounds pretty good, but feels a bit too rigid. The third method sounds wonderful, but requires more coding effort and creates another variable in the experiments, while the final option seems to make the best tradeoff between making the most of the spatial data available and minimizing the amount of added complexity. Needless to say, we chose the reflection option.

Now let us take a moment to clarify how this boundary reflection works. Since the dataset needed to be traversed in some order, raster order was decided upon. Raster order, then, would be as is shown in Table 5. Without a sliding window, R_1C_1 is first, R_1C_2 second, and R_MC_N traversed last, as shown in Table 5. On the other hand, with a sliding window, R_1C_1 is still first, but it is the focus pixel in the center of the sliding window. The context values to the south, southeast, and east are easily identified. They are defined by the image. But the values to the northeast, north, northwest, west, and southwest are undefined. What values should be included in this context? To determine this, we make the edge pixels the axis of reflection. An example of this is shown in Table 6. There, the axis of reflection is shown in bold in a 5 pixel $\times$ 5 pixel context window centered on $\mathbf{R_1C_1}$ before any mask is applied.

During training time, this reflection of the edges is done as a preprocessing step. Later, during classification time, the labels output by the classifier must be placed in real time, in their correct reflection positions.

R_1C_1	$\rightarrow$	R_1C_2	$\rightarrow$	...	$\rightarrow$	R_1C_N
						$\downarrow$
$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$
$\downarrow$						
R_2C_1	$\rightarrow$	R_2C_2	$\rightarrow$	...	$\rightarrow$	R_2C_N
						$\downarrow$
$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$
$\downarrow$						
.	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
						$\downarrow$
$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$
$\downarrow$						
R_MC_1	$\rightarrow$	R_MC_2	$\rightarrow$	...	$\rightarrow$	R_MC_N

Table 5: Pixels are traversed in raster order

		North			
	R_3C_3 R_3C_2	$\mathbf{R_3C_1}$	R_3C_2	R_3C_3	
	R_2C_3 R_2C_2	$\mathbf{R_2C_1}$	R_2C_2	R_2C_3	
West	$\mathbf{R_1C_3}$ $\mathbf{R_1C_2}$	$\mathbf{R_1C_1}$	$\mathbf{R_1C_2}$	$\mathbf{R_1C_3}$	East
	R_2C_3 R_2C_2	$\mathbf{R_2C_1}$	R_2C_2	R_2C_3	
	R_3C_3 R_3C_2	$\mathbf{R_3C_1}$	R_3C_2	R_3C_3	
		South			

Table 6: Reflection of Pixel Values in a 5 × 5 context window

This insertion of labels as we go can be both beneficial and detrimental to the accuracy of the classifier. This depends on several factors including the strength of the relationship between the $y_{i,s}$ labels, the classifier's ability to learn that relationship, and the quality of $\hat{y}_{i,s'}$, which are the labels output by the classifier. The resulting classification images that we see are a result of a combination of these factors, and the propagation of error is clearly visible in Figures 9 and 11. This clear propagation of error is a direct confirmation that we have achieved a propagation of belief in our experiments. There is more discussion on this topic in section 4.7.

4 Results and Discussion

The results of this project are surprising. Although we expected that at the higher input or output context sizes that the accuracy would rapidly drop off, and along increasing both IC and OC values there would be a peak, this is not what our results show. For example, our results show that in the case of C4.5, there is a peak at (7,3), see Figure 7, and a sudden drop-off in accuracy at (7,5).

Both classifiers in this study performed similarly. Although naive Bayes outperforms C4.5 with no bagging or boosting, they perform equally well after boosting 20 or 30 iterations. The window settings which give the best accuracy for each type of experiment run is shown in Table 7.

Experiment	Best Accuracy	{(IC size,OC size)}
NB	77	{(1,3)}
Adaboost NB (10)	77	{(1,3), (3,5)}
Adaboost NB (20)	81	{(3,5)}
Adaboost NB (30)	81	{(3,5)}
Bagging NB (10)	77	{(1,3)}
Bagging NB (20)	77	{(1,3)}
C4.5	74	{(7,3)}
Adaboost C4.5 (10)	78	{(5,3)}
Adaboost C4.5 (20)	82	{(3,3)}
Adaboost C4.5 (30)	81	{(3,3), (5,3)}
Bagging C4.5 (10)	78	{(1,3)}
Bagging C4.5 (20)	78	{(1,3)}

Table 7: Best context sizes for each set of experiments

Although other researchers have used this dataset, since we include spatial information in our study, there are no direct comparisons which can be made. Given that, our best results have nearly the same accuracy as the those Solberg et. al. [16] who used Maximum likelihood on the Feltwell dataset. Giacinto et. al. [8] experimented with dynamic classifier selection which outperforms our methods by about 10 percentage points.

In each of the following subsections, the non-ensemble results are shown side by side with the ensemble results for comparison, and in each table the accuracy of the classifier is shown for various input and output context sizes. The best result is shown in bold in each table. In order to aid visualization of the results, contour plots are also shown of the results.

4.1 C4.5

C4.5 with the window setting (1,1), which is equivalent to using no sliding windows, achieves an accuracy of 59%. If one could avoid the terrible settings of (5,5) and (7,5), all of the context window settings are better than (1,1). This can be easily seen in the contour plot in Figure 3. The best accuracy of 74% is achieved at

the setting (7,3). This best accuracy setting might be hard to find without trying many different settings. Changing from (7,3) to (7,5) causes a drop in accuracy of 50 percentage points, see Table 8 .

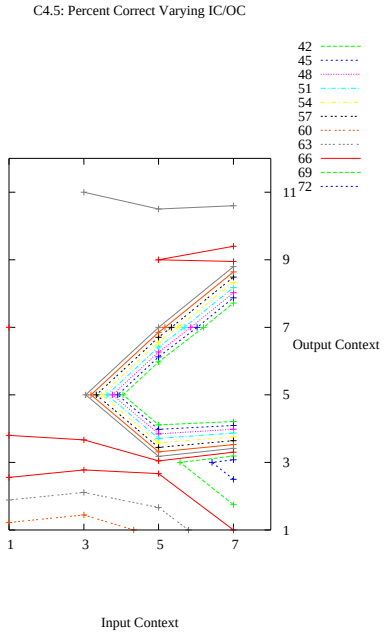


Figure 3: Contour plot of the Accuracy of C4.5 at Various IC and OC Sizes

	IC	1	3	5	7
OC	1	59	58	61	66
	3	68	67	67	**74**
	5	63	64	22	21
	7	66	63	63	28
	9	64	64	66	67
	11	65	63	62	62

Table 8: Accuracy of C4.5 at Various IC and OC sizes

4.2 Naive Bayes

Naive Bayes with the window size (1,1), which is equivalent to using no sliding windows, achieves an accuracy of 71%. Although adding input context shows some improvement, adding output context gives us our greatest gains at (1,3). This is surprising in that previous work by Joshi shows that naive Bayes doesn't handle output context very well [12]. This effect may be seen in output contexts greater than three. It may be that an output context of three is just small enough to help without overwhelming the naive Bayes classifier.

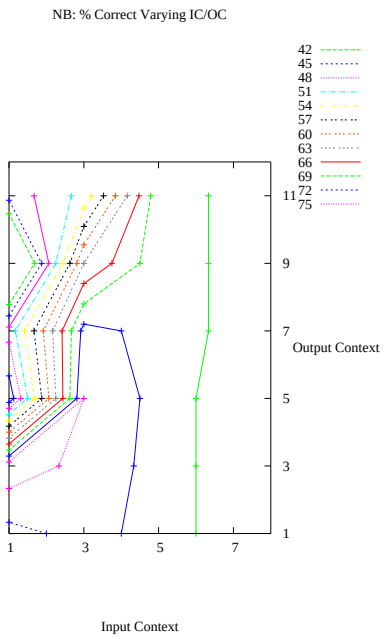


Figure 4: Accuracy of naive Bayes at Various IC and OC sizes

	IC	1	3	5	7
OC	1	71	73	71	67
	3	**77**	74	71	67
	5	43	75	71	67
	7	49	73	71	68
	9	31	63	71	68
	11	46	52	71	68

Table 9: Accuracy of naive Bayes at Various IC and OC sizes

4.3 Bagging C4.5

Bagging C4.5 improves the accuracy of the classifier and significantly changes the settings of the input context size that performs the best. Without bagging, the best settings are (7,3), and with bagging the best is (1,3), see Table 10. Although bagging is far from free, the reduction of vector length from IC=7 to IC=1 is nearly a factor of 29! This, in turn, significantly reduces the memory requirement during training and testing, while delivering an improvement in accuracy.

In the contour plots, see Figure 5, an area of very poor performance is clearly seen in the area near (7,5), and the area of best performance can be seen growing near (1,3) and (3,3) when transitioning from ten to twenty bags.

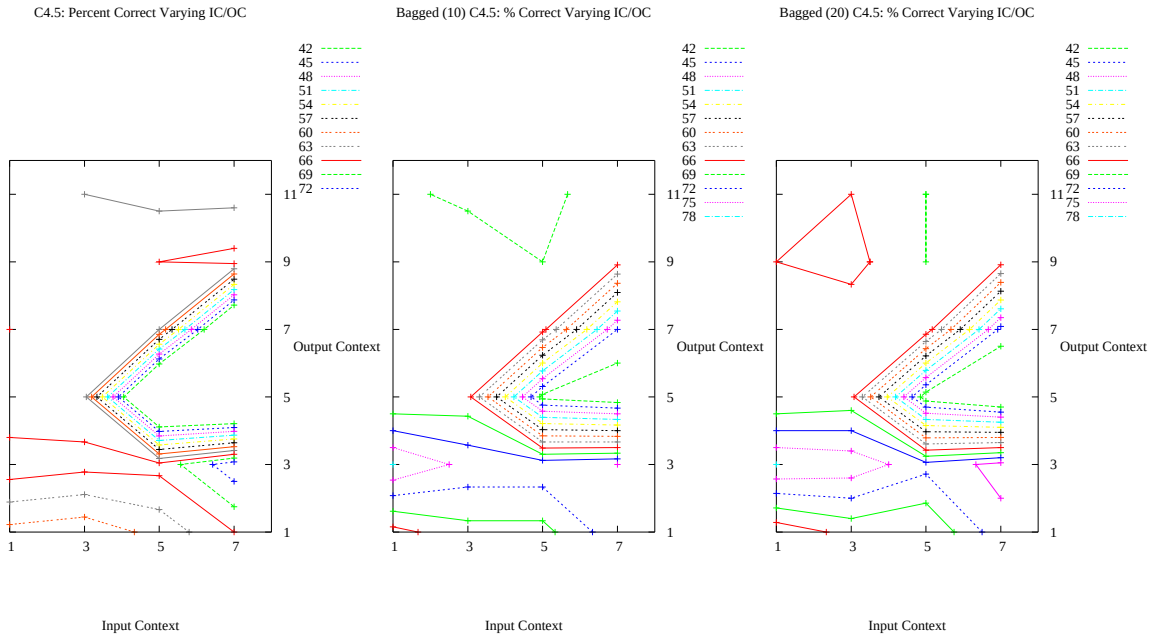


Figure 5: Effect of bagging using C4.5

C4.5

	IC	1	3	5	7
OC	1	59	58	61	66
	3	68	67	67	**74**
	5	63	64	22	21
	7	66	63	63	28
	9	64	64	66	67
	11	65	63	62	62

Bagging (10) C4.5

	IC	1	3	5	7
OC	1	65	68	68	74
	3	**78**	74	74	75
	5	66	67	41	39
	7	68	68	67	45
	9	67	66	69	67
	11	68	67	70	67

Bagging (20) C4.5

	IC	1	3	5	7
OC	1	64	67	66	74
	3	**78**	77	73	76
	5	66	67	40	36
	7	68	68	68	44
	9	66	65	69	67
	11	68	66	69	67

Table 10: Effect of bagging using C4.5

4.4 Bagging Naive Bayes

Bagging naive Bayes has very little effect on the accuracy. Both with and without bagging, the best settings are (1,3), see Table 11. Bagging even hurts the classifier at (1,7) and (1,11), which Breiman suggests is possible for stable classifiers [3], such as naive Bayes. In the contour plots, see Figure 6, the lack of change is clearly visible, with no sweet spots of high accuracy emerging or growing. A small change in the output

context size, from three to five, causes the accuracy to drop nearly in half.

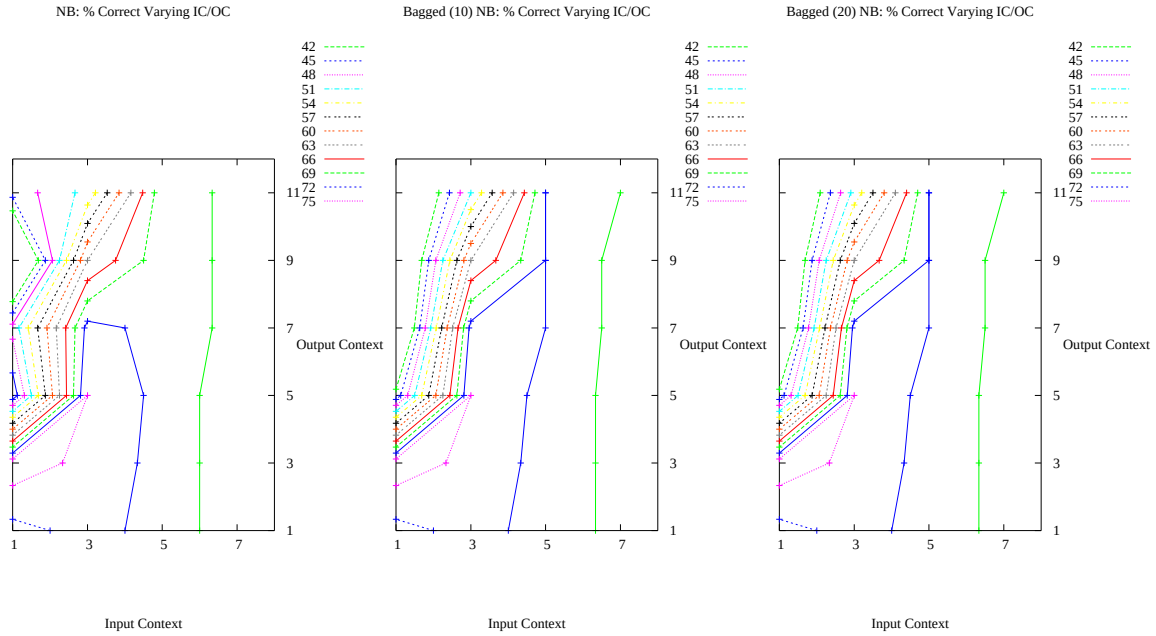


Figure 6: Effect of bagging using naive Bayes

(a) naive Bayes

	IC	1	3	5	7
OC	1	71	73	71	67
	3	**77**	74	71	67
	5	43	75	71	67
	7	49	73	71	68
	9	31	63	71	68
	11	46	52	71	68

(b) Bagging (10) naive Bayes

	IC	1	3	5	7
OC	1	71	73	71	68
	3	**77**	74	71	68
	5	43	75	71	68
	7	32	73	72	68
	9	31	63	72	68
	11	30	51	72	69

(c) Bagging (20) naive Bayes

	IC	1	3	5	7
OC	1	71	73	71	68
	3	**77**	74	71	68
	5	43	75	71	68
	7	32	73	72	68
	9	31	63	72	68
	11	30	52	72	69

Table 11: Effect of bagging using naive Bayes

4.5 Boosting C4.5

In the results of boosting C4.5, see Figure 7 and Table 12, we see that as we increase the number of iterations of boosting with the C4.5 classifier, the area of best performance seems to grow from larger input context window sizes towards smaller input context window sizes. This change is evident in the contour plots in Figure 7. The benefit of better accuracy at smaller context sizes is two-fold. Firstly, the accuracy is improved, and secondly, there is a reduction in computational resource requirements. This second point is important because it shows that using boosting allows us to use significantly fewer features. A change from (7,3) to (3,3) is a decrease in the number of features of over 50%, which, in this case, is 240 fewer features.

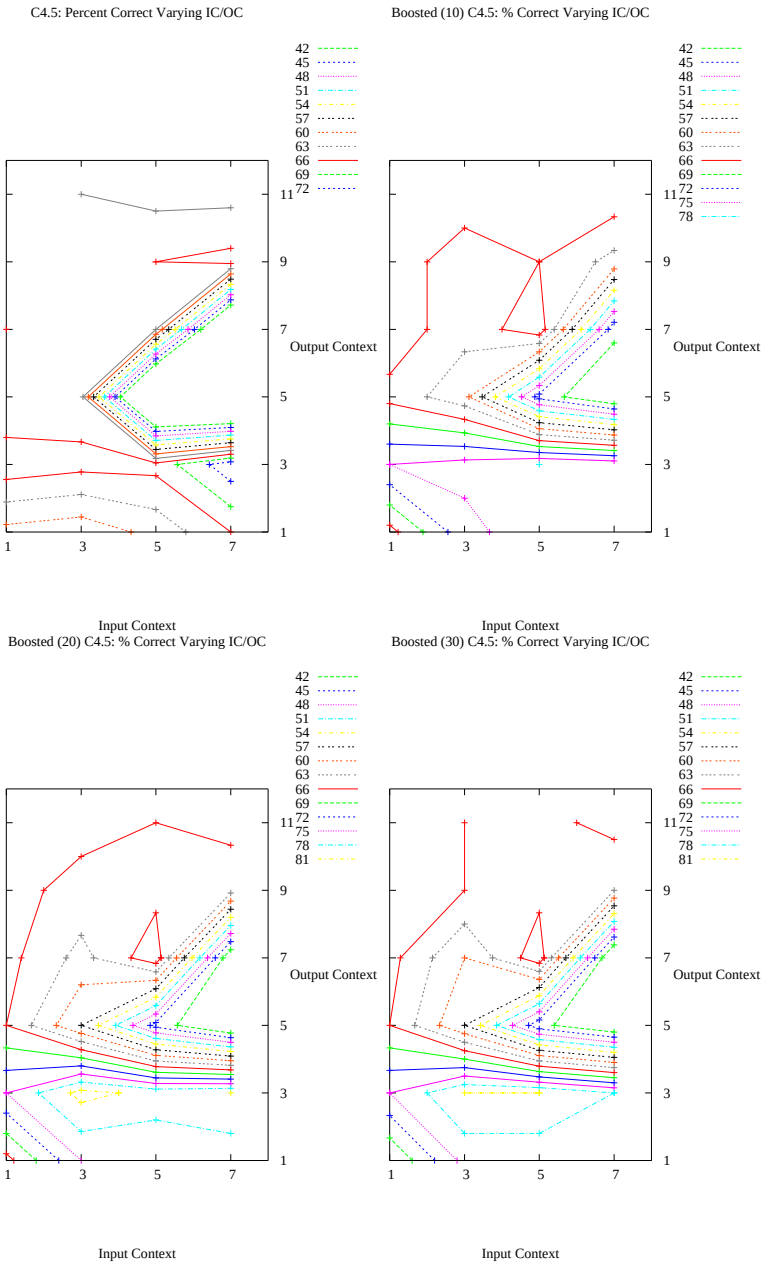


Figure 7: Effect of Adaboost using C4.5

C4.5

	IC	1	3	5	7
OC	1	59	58	61	66
	3	68	67	67	**74**
	5	63	64	22	21
	7	66	63	63	28
	9	64	64	66	67
	11	65	63	62	62

Adaboost (10) C4.5

	IC	1	3	5	7
OC	1	65	74	77	77
	3	75	76	**78**	77
	5	65	61	44	38
	7	68	64	68	43
	9	67	65	66	62
	11	66	67	67	68

Adaboost (20) C4.5

	IC	1	3	5	7
OC	1	65	75	75	76
	3	75	**82**	80	81
	5	66	57	44	37
	7	67	62	68	39
	9	67	65	65	64
	11	66	67	66	67

Adaboost (30) C4.5

	IC	1	3	5	7
OC	1	66	76	76	76
	3	75	**81**	**81**	78
	5	66	57	43	38
	7	67	60	68	37
	9	67	66	65	63
	11	66	66	65	67

Table 12: Effect of boosting using C4.5

4.6 Boosting Naive Bayes

In Figure 8 we see that as we increase the number of iterations of boosting with the naive Bayes classifier we expand the number of good combinations of input and output context. With the setting (3,5), naive Bayes shows over 80% accuracy at 30 iterations. Boosting has mixed results with naive Bayes, with little change between boosting 20 and 30 iterations. It results in a 4 percentage point gain in accuracy from 77% at (1,3) to 81% at (3,5).

(a) naive Bayes

	IC	1	3	5	7
OC	1	71	73	71	67
	3	**77**	74	71	67
	5	43	75	71	67
	7	49	73	71	68
	9	31	63	71	68
	11	46	52	71	68

(b) Adaboost (10) naive Bayes

	IC	1	3	5	7
OC	1	71	74	71	68
	3	**77**	74	72	68
	5	69	**77**	74	70
	7	32	75	75	72
	9	32	65	75	73
	11	31	59	72	72

(c) Adaboost (20) naive Bayes

	IC	1	3	5	7
OC	1	71	74	71	68
	3	77	74	71	68
	5	75	**81**	74	70
	7	32	75	76	72
	9	32	65	75	73
	11	31	59	72	72

(d) Adaboost (30) naive Bayes

	IC	1	3	5	7
OC	1	72	74	72	68
	3	78	75	72	68
	5	76	**81**	74	70
	7	32	75	76	72
	9	32	65	75	73
	11	31	59	72	72

Table 13: Effect of boosting using naive Bayes

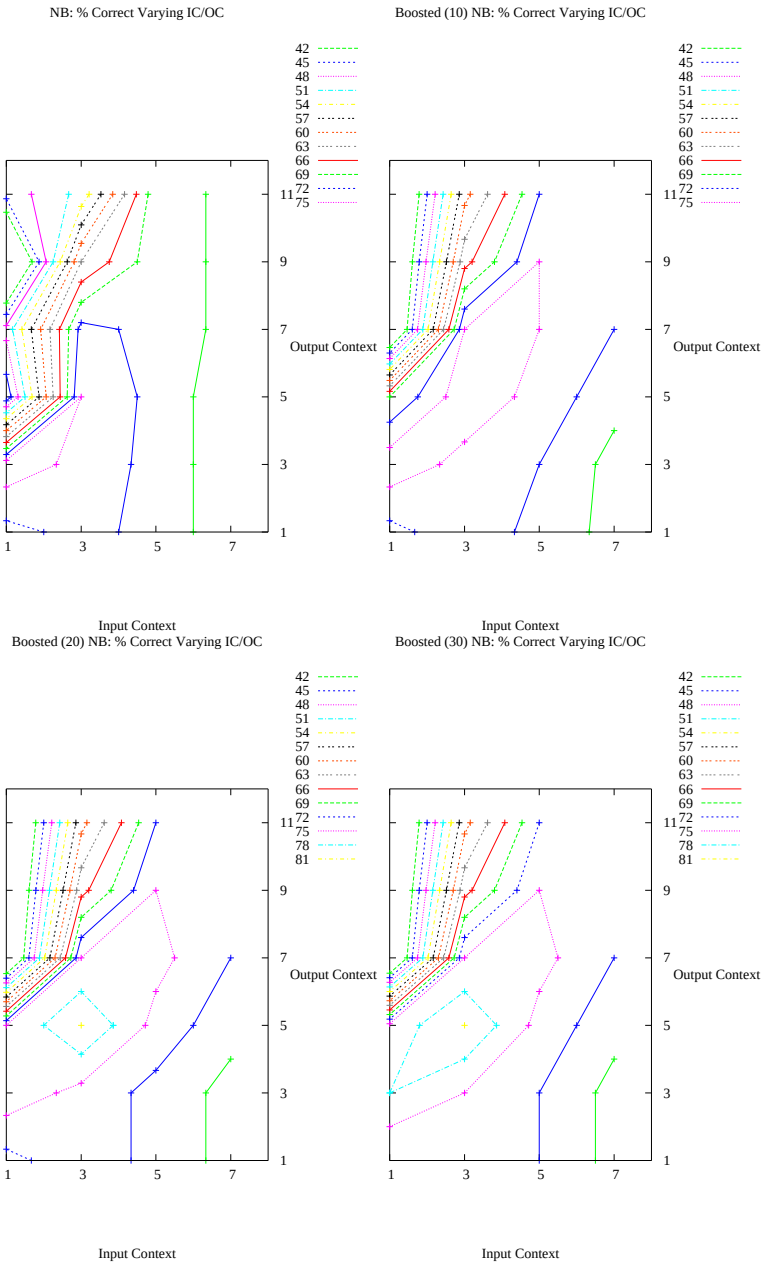


Figure 8: Effect of adaboost using naive Bayes

4.7 Ensemble Effects

In this project, we have ensemble effects not only in the classification stage with the voting of individual classifiers created by bagging and boosting, but also in the spatial ensemble where we use voting of multiple image classification maps.

While bagging and boosting are well known, we have yet to discuss how exactly we go about performing this spatial ensemble. The spatial ensemble is a straightforward majority voting of the resulting classification maps from the eight test images, with an arbitrary tie breaker.

How can we evaluate the individual contribution made by each ensemble?

One way to see the benefit from each ensemble alone is to compare the accuracy of the classifier with and without the ensemble in action. In Tables 14 and 15, we show the contribution that each ensemble type has made. Each section in the table takes a selected (IC,OC) setting, and shows the accuracy at that setting with a "bare" classifier, and the accuracy of that classifier with an ensemble classifier (e.g. bagging or boosting). This shows the benefit due to the ensemble classifier.

Then, to see the benefit of the spatial ensemble we compare the accuracy of both the ensemble classifier and "bare" classifier with and without the spatial ensemble. For the case with the spatial ensemble, we vote the eight different classification maps to generate a final map, and for the case without the spatial ensemble, we average the 8 accuracies to get the final accuracy. In this way, we can investigate the contribution of the spatial ensemble by comparing these two accuracies.

In Table 14, the context sizes were selected so that at least one of the accuracies was the "best" accuracy for that set of experiments. You can refer back to Table 7 to see other best accuracies. Also shown are those same results with and without the spatial ensemble. Please note that only the accuracies with the spatial ensemble are known to be the best or worst.

When comparing the values in a row from left to right, we see the effect of spatial ensembling. We can see that C4.5 benefits the most from this ensemble in Table 14. Naive Bayes is a more stable classifier, and as such benefits less from spatial ensembling than C4.5. When comparing values in a column, we see the benefit due to bagging or boosting.

Ensemble Algorithm	**Spatial Ensemble**	
	No	**Yes**
NB (3,5)	74	75
Boosted (20) NB (3,5)	77	**81**

Ensemble Algorithm	**Spatial Ensemble**	
	No	**Yes**
NB (1,3)	75	77
Bagged (10) NB (1,3)	74	**77**

Ensemble Algorithm	**Spatial Ensemble**	
	No	**Yes**
C4.5 (3,3)	63	67
Boosted (20) C4.5 (3,3)	76	**82**

Ensemble Algorithm	**Spatial Ensemble**	
	No	**Yes**
C4.5 (1,3)	62	62
Bagged (10) C4.5 (1,3)	75	**78**

Table 14: Accuracy of naive Bayes and C4.5 with and without ensemble effects. Values in **bold** are the **best** results for all IC/OC settings

Conversely, the worst performance settings and accuracies are shown in Table 15. The accuracy is shown for four different cases. These are for the "bare" classifier and the ensembled classifier each with their results using spatial ensembling and also when the resulting accuracy on each of the eight test sets are averaged.

As expected, when the performance is at its best as we are showing in Table 14, the spatial ensembling has a greatly beneficial effect. On the other hand, when the classifier's performance is poor, spatial ensembling

Ensemble Algorithm	Spatial Ensemble	
	No	**Yes**
NB (1,11)	46	44
Boosted (10) NB (1,11)	30	**31**

Ensemble Algorithm	Spatial Ensemble	
	No	**Yes**
NB (1,11)	46	44
Bagged (10) NB (1,11)	29	**30**

Ensemble Algorithm	Spatial Ensemble	
	No	**Yes**
C4.5 (7,5)	23	**21**
Boosted (20) C4.5 (7,5)	39	**37**

Ensemble Algorithm	Spatial Ensemble	
	No	**Yes**
C4.5 (7,5)	23	**21**
Bagged (20) C4.5 (7,5)	36	**36**

Table 15: Accuracy of naive Bayes and C4.5 with and without ensemble effects. Values in **bold** are the **worst** results for all IC/OC settings

either helps or hurts performance, but neither by very much. In these poor accuracy cases, the ensemble classifier has a much greater impact than the spatial ensemble, but nevertheless, the accuracy remains poor. This can be seen in Table 15.

Another way to visualize the effect of the spatial ensemble is to view the eight classification maps and the final classification map side-by-side. These eight classification maps are ensembled together to create our final classification map for each experiment. C4.5 and naive Bayes behave somewhat differently, and so, an example of each is shown in Figures 9 and 11 respectively.

As mentioned earlier, the inclusion of output context leads to a propagation of belief in the classification of the pixels in the images. This is due to the classification of the current pixel being dependent on the labels of previously classified pixels. This is exactly what we are after in the case of agricultural fields. In agricultural fields, the y_i's would be large contiguous regions of a single plant type or monoculture. And so, if we believe that a pixel is bordered by corn to the north and west, we should be more likely to predict the focus pixel to be corn also and less likely to predict that the focus pixel is soybeans. This propagation of belief can turn into propagation of error when the classifier makes errors. While this is not good for the accuracy of each classification map, when the spatial ensemble is applied, much of the error is eliminated, while much of the benefit is preserved. Two beautiful examples of this propagation of error and then elimination of error can be seen in Figures 9 and 11. In each, the individual classification maps for each of the eight testing sets are shown side-by-side with result of the spatial ensembling at the bottom.

In Figure 9 the results for the best C4.5 setting at (7,3) are shown. These eight images are the classification maps for the eight-fold test set, while the image at the bottom is the result of the spatial ensemble.
The average accuracy of these eight maps is 64% while the accuracy with the spatial ensemble is 74%! Figure 11 shows the results for naive Bayes with the context setting of (3,7). Without the spatial ensemble, the average accuracy is 70%, and individually as low as 67%, while the accuracy after the spatial ensemble is 73%.

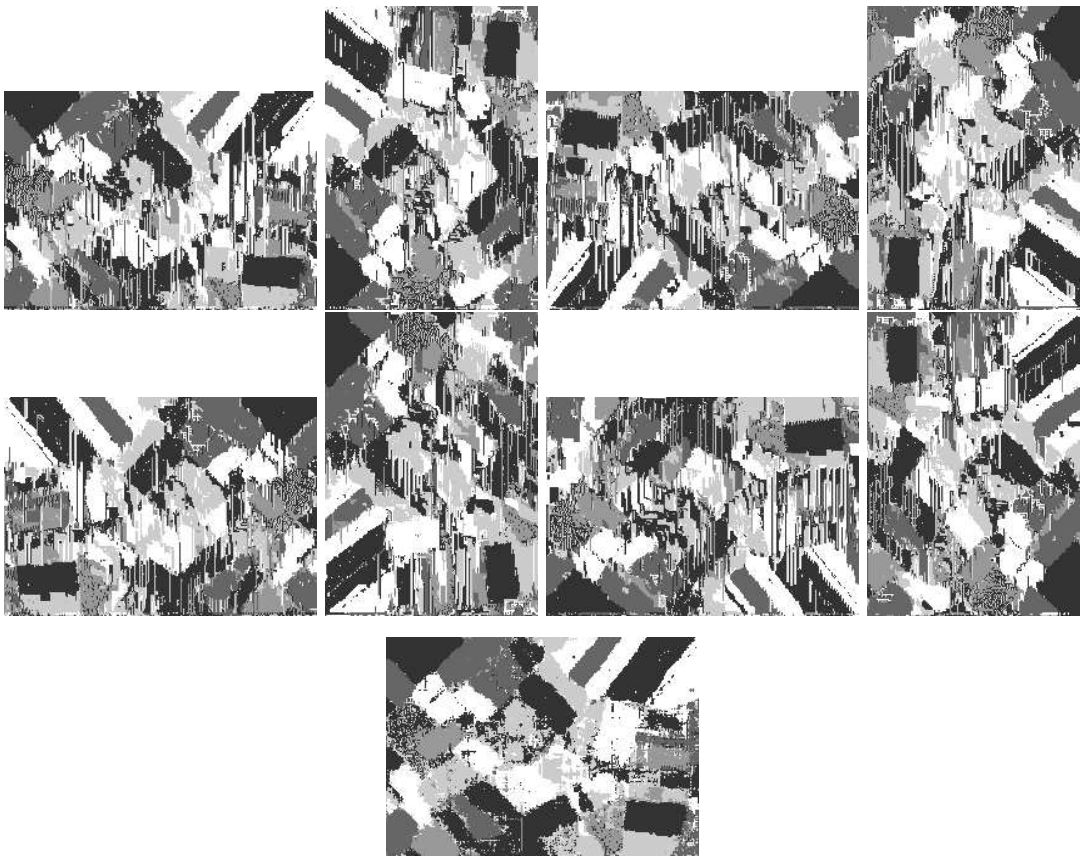


Figure 9: Best C4.5 (7,3) results. Each of the eight images at the top corresponds to a recurrent sliding window classifier in a single scan order (the pictures have been rotated so that the scan order is left-to-right and top-to-bottom). We can see the "runny paint" effect whereby an error in one pixel is propagated for many pixels in the direction of the scan order. The bottom image is the result of the spatial ensemble voting. The voting has eliminated most of the runny paint to produce a much more accurate predicted image.

Figure 10: Test set ground-truth image for comparison with the spatial ensemble results in Figures 9 and 11. The large black area corresponds to the pixels with unknown ground-truth labels.

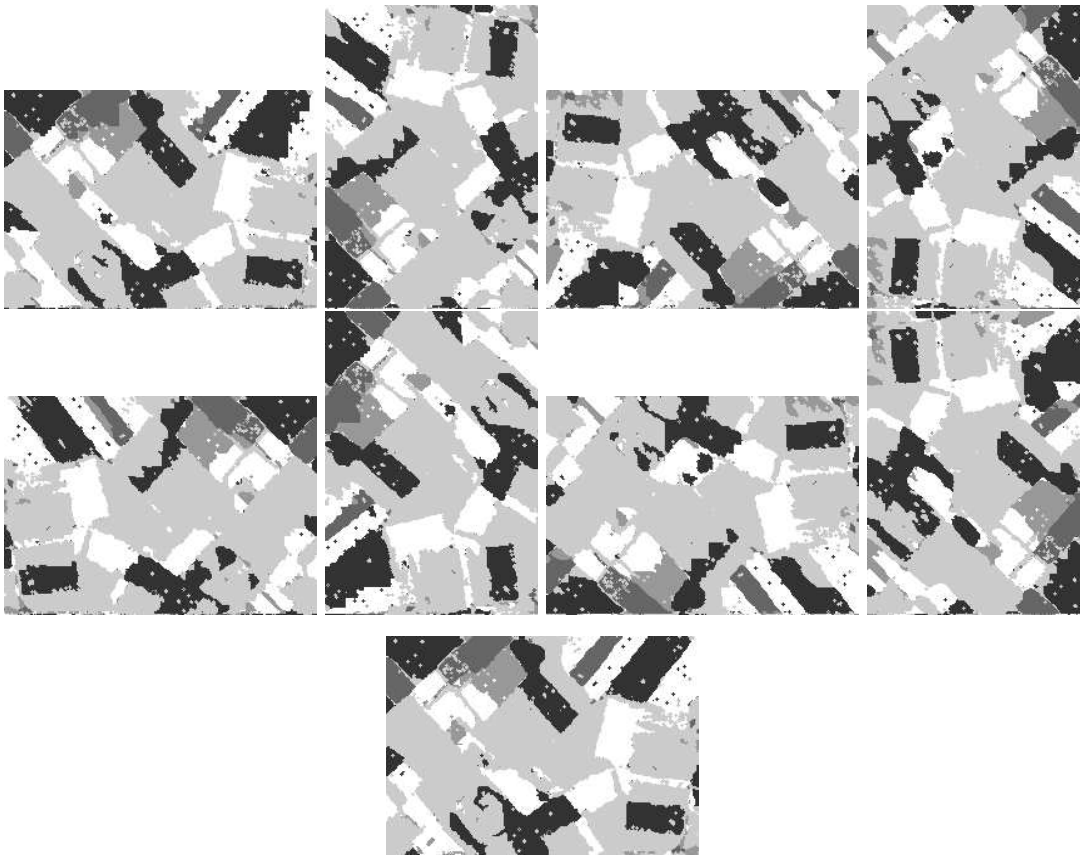


Figure 11: Naive Bayes (3,7). Each of the eight images at the top corresponds to a recurrent sliding window classifier in a single scan order (the pictures have been rotated so that the scan order is left-to-right and top-to-bottom). We can see the "stair-stepping" effect whereby an error in one pixel is propagated for many pixels and "stretches" the agricultural fields in the direction of the scan order. The bottom image is the result of the spatial ensemble voting. The voting has eliminated most of the stretched areas to produce a much more accurate predicted image.

5 Conclusions

5.1 Recommendations

We considered three criteria in deciding which method to recommend for general use: (a) accuracy, (b) robustness, and (c) training time. Using those features as criteria, here are our recommendations. Use boosting as it improves performance on both classifiers studied, while bagging seems to have had little effect. A similar out-performance of bagging by boosting was also seen by Freund [7]. While both classifiers take about the same amount of time during classification, C4.5 takes considerably longer to train than naive Bayes, so try naive Bayes first, if you are short on time. Finally, when boosted, both algorithms are nearly equally as robust as each other, so you are nearly equally likely to hit an area of really poor performance while exploring the various (IC,OC) settings with either algorithm.

Our results, like Joshi's [12], show that there is no substitute for trying all the (IC,OC) combinations in order to find the best settings and results for any of the classification algorithms. There is one method, though, which may lead you to a quick global maximum. This method is called hill-climbing.

In hill-climbing, the course of experiments is guided by following the best accuracy achieved and performing all of the experiments for neighboring (IC,OC) settings. That is, say we start with the settings (IC,OC). Then, to perform hill-climbing, we must run the experiment on the neighbors of (IC,OC). Then, repeat this last step with the new focus set to the (IC,OC) settings which produced the best results. We would continue repeating until we have already tested the neighbors of the best settings. Once this point is reached, the accuracy is a local maximum. In the algorithm below, a neighbor is eligible if it is an (IC,OC) combination on which you are willing to, but have not yet, run an experiment.

1. Initially run experiments on setting of (IC',OC') at (1,1)
2. repeat until (IC',OC') == (IC,OC):
   (a) Set: IC=IC', OC=OC'
   (b) Run experiment on all eligible neighbors of (IC,OC)
   (c) Set (IC',OC') to settings with the maximum accuracy, taking new (IC,OC) settings in case of a tie

In our experiments, hill-climbing would discover the best results for all but one of the experiments starting from (1,1). See Table 16 for the number of (IC,OC) settings necessary to complete the hill climbing algorithm. The total number of combinations experimented at in this project per algorithm is 24, whereas with hill climbing, all those that made the global maximum did so in an average of 9 (IC,OC) settings.

Algorithm	Number of experiments to maximum
C4.5	Failed to reach global maximum
AdaBoosted (10) C4.5	11
AdaBoosted (20) C4.5	9
AdaBoosted (30) C4.5	12
Bagging (10) C4.5	6
Bagging (20) C4.5	6
naive Bayes	6
Adaboost (10) naive Bayes	11
Adaboost (20) naive Bayes	11
Adaboost (30) naive Bayes	11
Bagging (10) naive Bayes	6
Bagging (20) naive Bayes	6

Table 16: Number of different (IC,OC) settings to Maximum

5.2 Further Research

Our results suggest the algorithms studied perform better when given large input and output contexts. One explanation of this behavior is that the pixels are small relative to the size of the features of agricultural fields. A classifier may need a large window on the scene to "see" the edge of the field or detect a boundary between one field and another. In such cases, very nearby pixels may not contain enough information. This problem could be addressed in two ways: (a) by defining a "donut context" in which a disk of nearby pixels is ignored and only a ring of more distant pixels is used for input and output context and (b) by increasing the coarseness (reducing the resolution) of the image using area subsampling.

In the "donut context" technique, one could imagine a mask in the shape of a disk which covers local pixel data. Then, the context sizes are set to exceed the size of this mask. In this way, the classifier is given only "distant" pixel information, and no local pixel information. This allows us to answer the question of whether or not it is just the distant pixel information that gives us the gains that we're looking for, or if it is some combination of the distant and near pixels.

The subsampling technique would utilize a regularly spaced grid overlaying the image. Within each grid square, there is a set of pixels. From these pixels, using some function, a single pixel is selected to represent the group. This is referred to as area subsampling by Dodgson [6].

Through subsampling, the resulting image would have a more condensed structure. One might imagine a smaller looking yet similar image to the original. This condensing would, presumably, move more distant information in the original image closer together in the resulting image. This may result in smaller context window sizes performing in a fashion similar to larger context sizes in the original. This would help to answer the question of whether or not the classifiers would perform the same or better with smaller context sizes as it did with larger context sizes with the original training set. This may also lead to training and testing with a more sparsely populated context. Something a checker-board type mask might provide.

5.3 An additional control in this experiment

In all cases reported in this paper, the algorithm was trained on all 8 rotations and reflections of the original data. We should also have run the algorithm on just the original image to see how much benefit was obtained by training on the reflections and rotations.

6 Bibliography

References

[1] G. Bakiri. Converting English text to speech: A machine learning approach. Technical Report 91-30-2, Department of Computer Science, Oregon State University, Corvallis, OR, 1991.

[2] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1-2):105–139, 1999.

[3] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[4] T. G. Dietterich. Machine learning research: Four current directions. *AI Magazine*, 18(4):97–136, 1997.

[5] T. G. Dietterich. Ensemble methods in machine learning. *Lecture Notes in Computer Science*, 1857:1–15, 2000.

[6] N. A. Dodgson. Image resampling. Technical Report UCAM-CL-TR-261, University of Cambridge, Computer Laboratory, August 1992.

[7] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.

[8] G. Giacinto, Roli F., and Fumera G. Selection of classifiers based on multiple classifier behaviour. In *SSPR/SPR*, pages 87–93, 2000.

[9] G. Giacinto, F. Roli, and L. Bruzzone. Combination of neural and statistical algorithms for supervised classification of remote-sensing images. *Pattern Recognition Letters*, 21(5):385–397, 2000.

[10] IEEE GRSS. Data fusion reference database data set grss_dfc_0006, 2001.

[11] G. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the 11th Annual Conference on Uncertainty in Artificial Intelligence (UAI-95)*, pages 338–345, San Francisco, CA, 1995. Morgan Kaufmann Publishers.

[12] S. Joshi and T. G. Dietterich. Calibrating recurrent sliding window classifiers for sequential supervised learning (revised), 2003.

[13] J. R. Quinlan. *C4.5: Programs for Empirical Learning.* Morgan Kaufmann, San Francisco, CA, 1993.

[14] S. Serpico, L. Bruzzone, and F. Roli. An experimental comparison of neural and statistical non-parametric algorithms for supervised classification of remote-sensing images. *Pattern Recognition Letters*, 17(13):1331–1341, 1996.

[15] N. M. Short Sr. The remote sensing tutorial (http://rst.gsfc.nasa.gov/), 2005.

[16] A. Solberg, G. Storvik, and R. Fjrtoft. A new approach to statistical multisensor image classification. In *Proc. International Geoscience and Remote Sensing Symposium (IGARSS'02)*, Toronto, Canada, 24–28 June 2002.

[17] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools with Java implementations.* Morgan Kaufmann, San Francisco, CA, 2000.

APPENDIX

A Terminology

Here is a list of some of the terminology included in this paper.

Input Context (IC) - This is the square size that bounds the number of input pixels included in a feature vector. If IC=1, only the features of the target pixel are used for classification. If IC=3, then a 3-pixel by 3-pixel square is used, and the upper, lower, left and right pixels are used as input context. See Table 3 for sample contexts.
Output Context (OC) - Inclusion of previously predicted class labels into features used in recurrent sliding windows. Similar to IC (above) the setting of the OC is an upper bound on the number of labels included in a feature vector. When OC=1, there is no "output context", and when IC=3, the pixels located above and to the left are included. See Table 3 for sample contexts.
(IC, OC) - This ordered pair denotes an input context setting and output context setting pair. For example, (3,3) means IC=3, and OC=3.
Pixel - This is the smallest element of an image.
Subsampling - This is a technique to reduce the size of an image by selecting one pixel from the original image to represent a group of pixels from the original image.
Class Label - This is the label which corresponds to a feature vector.
Ground Truth - This is the correct class label for a feature vector. For instance, corn, may be a pixel's ground truth label.
Bagging - An ensemble method of creating many classifiers using variations of the training set to create varying classifiers. These classifiers are voted to create a final ensemble classifier.
Boosting - An ensemble method of placing a greater emphasis on training examples which are incorrectly labeled during training time of previously constructed classifiers. Like bagging, classifiers are voted to create a final ensemble classifier.
C4.5 - Decision Tree Classifier. Creates a model of the data which is then used to classify new instances.
Naive Bayes - naive Bayes Classifier. Simple classifier which is very stable. It generates probability distributions over the features for each possible class.
Supervised Learning - The type of machine learning which deals with training from a dataset of labeled examples.
Supervised Sequential Learning - Sequential learning deals with learning sequences, in this project, this is implemented using a sliding window.
(2-D) Sliding Window - A technique used to allow the context of a pixel to be included in it's feature vector.
(2-D) Recurrent Sliding Window - A sliding window for input and output context. In addition to the input context, a recurrent sliding window allows for the inclusion of previously generated predictions in the current feature vector to be classified.
Ensembles - There are three types of ensembles in this project. The first is due to the voting of individual classifiers created by bagging or boosting. The second is due to the voting over multiple scan orders. Both ensemble effects are important.
Spatial Ensemble - In this project, the results of eight classification maps are voted to produce a final mapping with arbitrary tie breaking.