Grasshopper Infestation Prediction:
An Application of Data Mining to Ecological Modeling

Waranun Bunjongsat

A Research paper submitted in partial fulfillment of the requirements

for the degree of Master of Science

Major Advisor: Dr. Thomas G. Dietterich

Department of Computer Science

Oregon State University

Corvallis, OR 97331-3202

December 5,1999

Abstract

This thesis presents a case study of applying machine learning tools to build a predictive model of annual infestations of grasshoppers in Eastern Oregon. The purpose of the study was two-fold. First, we wanted to develop a predictive model. Second, we wanted to explore the capabilities of existing machine learning tools and identify areas where further research was needed.

The study succeeded in constructing a model with modest ability to predict future grasshopper infestations and provide advice to farmers about whether to treat their croplands with pesticides. Our analysis of the learned model shows that it should be able to provide useful advice if the ratio of treatment cost to crop damage cost is 1 to 1.67 or more. However, there is some evidence that the model is not able to make good predictions in years with extremely high levels of grasshopper infestation.

To arrive at this successful model, three critical steps had to be taken. First, we had to properly formulate the prediction task both in terms of exactly what we were trying to predict (i.e., the probability of infestation) and the spatial area over which we could make predictions (i.e., areas within 6.77 kms radius of a weather station). Second, we had to define and extract a set of features that incorporated knowledge of the grasshopper life cycle. Third, we had to employ evaluation metrics that were able to measure small improvements in the quality of predictions.

The study identified important directions for future research. In the area of grasshopper ecology, there is a need for improved data gathering tools including a much denser and more widespread network of weather stations. These stations should also measure subsoil temperatures. Recording of the dates of hatching of grasshopper nymphs would also be very valuable. In machine learning, methods are needed for automating the definition and extraction of features guided by qualitative knowledge of the domain, such as our qualitative knowledge of the grasshopper lifecycle.

Table of Contents

Chapter 1 Introduction

Data mining is a new discipline focusing on searching and analyzing data in large data sets to look for unknown patterns and/or new rules from the data. It combines concepts and methodologies from many disciplines including machine learning, databases, data warehousing, statistics, and knowledge-based systems. This paper summarizes a study of the data mining process for a challenging application – the prediction of grasshopper infestations in Eastern Oregon based on the previous year's infestation and the current year's weather. The purpose of this study was to determine whether state-of-the-art data mining tools are capable of making useful predictions on this difficult problem. An additional goal was to understand all of the issues that arise when attempting a data mining task of this kind in order to identify problems that require future research.

This introduction is organized in two parts. First, we describe the grasshopper prediction problem. Then, we give an overview of the data mining process.

1.1 The Grasshopper Infestation Prediction Problem

Grasshoppers are common agricultural pests in Eastern Oregon. If farmers are given sufficient warning, they can spray pesticides which will kill the grasshoppers and prevent major losses to crops and grazing lands. To qualify for government matching funds, which reduce the costs of spraying, farmers need several months' warning in order to have time to prepare environmental impact statements and submit other required paperwork. The goal of the grasshopper prediction task is to provide useful predictions to farmers in November (in time for filing environmental impact statements) and in May (in time to make final spraying decisions).

The adult grasshopper population has been measured every summer in late July since 1953. The results of these measurements have been entered into a geographical information system (GIS), with one GIS "layer" (or map) per year. The map gives a set of regions, and each region is labeled by one of the following labels:

- **None** (infestation of less than 3 grasshoppers per square yard)
- **Non-economic** (infestation of less than 8 grasshoppers per square yard)
- **Economic** (infestation of 8 or more grasshoppers per square yard)
- **Treated** (area sprayed during that year)

An example map is given in Appendix A.

Grasshoppers have an annual life cycle. Females mate and lay eggs in the late summer. These eggs remain in the soil during the winter, where they gradually mature. The rate of maturation is believed to be determined by the soil temperature. The eggs hatch in the late spring, and the immature grasshoppers ("nymphs") emerge from the soil with large appetites. Most agricultural damage is done by the nymphs.

The total grasshopper population has fluctuated tremendously over the period since 1953. Figure A.1 shows a graph of the total number of acres that were classified as non-economic, economic, or treated each year. Pink line of Figure A.1 shows the total number of acres of economic infestation. There are some years when there is virtually no economically-significant infestation, and other years when the infestation is widespread. Fortunately, economic infestation is relatively rare.

Weather data is collected daily at a fairly sparse network of weather stations. Each day, the total precipitation, minimum temperature, and maximum temperature are recorded. Figure A.2shows the distribution of weather stations in Eastern Oregon. Many of these weather stations were not in operation over the entire time period from 1953, and as a result, we have not been able to use their data. As can be seen in the figure, there are relatively few usable weather stations. We obtained the daily data for these weather stations from the office of the State Climatologist (George Taylor).

1.2 The Data Mining Process

The data mining process involves the following steps: 1) selecting and cleaning data, 2) transforming and building features, 3) building a predictive model and/or discovering new patterns, and finally 4) testing and summarizing the discovered knowledge. At each step of the process, different methods from different disciplines are required to accomplish the task. The first two steps involve database and data manipulation methods. The third step involves applying machine learning algorithms to discover patterns and predictive models. The fourth step applies statistical tools to test the knowledge and graphical tools to visualize what has been discovered.

It is estimated that the majority of the effort in the data mining process is in the first two steps. Working on the first step involves importing the data from databases or data warehouses, which contain large amounts of raw historical data. The sources of data can be of many types. For example, they can be formatted text data. They can be tables in a database management system. Or, they can be in the other standard format such as NetCDF (network common data format), which is a common standard for today scientific data. It is impossible for historical data to be without imperfection. An example of imperfection is missing data, which are caused from inability to capture, collect, or record the data at some points of time and space. Dealing with imperfection of data can be incredibly complicated task.

In business applications, new data analysis tools have recently been developed, including online analytical processing (OLAP), multi-dimensional databases (MDD), and star schemata. On-line analytical processing (OLAP) is a set of tools that can be used for simple analyses of data stored in data warehouses. The concept of OLAP is implemented and applied through multi-dimensional representation of data. The multi-dimensional database (MDD) is the major database architecture, designed for supporting OLAP. The star schema is a relational database architecture that implements multi-dimensional views of data using a single fact table and a set of dimension tables.

The concept of OLAP can be used in preparing data for mining. However, the difference between OLAP and data mining is that OLAP only analyzes the data to summarize the information about the data, for example, to see what is the standard deviation of the data, what is the change of selling rate between different years, and so on. Data mining, however, goes beyond that. It covers looking for patterns in the data and unusual events in the data with the goal of finding patterns and events that can help achieve the larger goals of the user (e.g., increase sales, reduce fraud, etc.) To find patterns or rules requires searching in a large space of patterns.

Features, which are also called attributes, are values of a particular type, which represent some qualities of the data objects. They can be attributes of the raw data, or they can be values that have been derived from the raw attributes. In our grasshopper problem, there are 1098 raw attributes. From these, we have defined 52 features to be used for predicting grasshopper infestations. The process of constructing features is called feature extraction or feature construction. Defining features involves searching for a subspace that is useful for the problem. The choice of the features may depend in part on the choice of the learning algorithm, since some learning algorithms require that features be independent, others require that the features be discrete, still others require that the features be real values in the unit interval. Learning algorithms have been extensively studied and developed in the machine learning discipline. The choice of the feature space has a profound effect on the quality of generated hypotheses (Bloedorn and Michalski, 1998).

In addition to defining and extracting good features, it is important to limit the total number of features provided to the learning algorithm. "Although many learning methods attempt to select, extract, or construct features, both theoretical analyses and experimental studies indicate that many algorithms scale poorly in domains with large numbers of irrelevant and/or redundant features (Langley, 1996)" (see also, Liu and Motoda, 1998). Feature subset selection is the process of selecting relevant features and screening out irrelevant ones. A good feature set is one that is sufficient and necessary. A set of features is sufficient if it permits learning algorithms to distinguish between examples belonging to different classes. For example, to detect telephone fraud, features are needed which can distinguish between fraudulent and legitimate phone calls. If the features are insufficient, then there will be cases where two phone calls will have the same values for their features, but one of the calls is fraudulent and the other is legitimate. A set of features is necessary (for a particular learning algorithm) if the algorithm cannot find an accurate hypothesis without those features. This is usually equivalent to saying that the features permit an accurate hypothesis to be represented compactly. It is related to the heuristic of Occam's razor, which states that the simplest among different alternatives is preferred for prediction (Wang et. al., 1998).

Predictive modeling is the task of finding a mathematical formula that can represent the relationship between the features and the target variable. Different modeling methods (e.g., linear regression, neural networks, decision trees, decision rules, etc.) construct different kinds of formulas. The set of relationships that can easily be modeled using a neural network is not the same as the set of relationships that can be easily modeled by decision trees. Hence, it is important to choose the modeling method that is appropriate to the application problem.

Some data mining applications, including our grasshopper infestation study, are problems where it is not known at exactly what level of detail to model the domain, nor is it known exactly what information is relevant for making predictive models at each level of detail. One of the most important purposes of data mining is to answer these questions, to find a good level of detail where a model will have predictive power and to determine what input features will be most useful at that level of detail. As we report below, in our study, we tried many different levels of detail and many different input features before we found a usable model.

Chapter 2 Data

2.1 Sources of Data

Data, which will be used in the data mining process, usually are historical data maintained in data warehouses. The data can be in many forms, for example, tables in a database management system, formatted text, and various binary formats. The raw data is the data that is initially collected and stored in computer-readable form. The derived data is data that has been transformed from raw data. Both raw data and derived data can be either managed data in the database management system or a standard file format, or it can be unmanaged data in a formatted or unformatted text file. No matter whether data is managed or unmanaged, derived or not derived, analyzing these data is a task of the data mining process. We can categorize source of data into three categories as following

2.1.1 DBMS

The source of the data is a DBMS. That is, the data is managed data. We can access the data via transactions to that DBMS and using SQL commands. Usually, one can do it by using SQL transactions or using/writing a program to have transactions via embedded SQL to access the data from the corresponding DBMS.

2.1.2 Text file

The data are in text files, which have not been managed by any DBMS yet. The text data files can be any of the following.

2.1.2.1 Formatted text file

That is, the data are stored in the files in a predictable and periodic format. Such as each line is a record comprising of attributes. Different lines have the same format. The difficulty of acquiring the data depends largely on how complicated the format is. One data importer or loader may be able to import from one format but not from another. The format of each data value in the file can be either delimited format or fixed-width format. Delimited format is a format where different data values can be written in the file with different lengths. However, they are separated from each other by a predefined character, such as a space ‘ ’ or a comma ‘,’. In fixed width format, all data values of the same type are stored in the file with the same length. There can be a delimiter between them or not. If the format does not have a delimiter between data values, it is possible that two data values can connect together into a single string.

2.1.2.2 Unformatted text files

The data are stored in the files without a predictable format. To acquire the data, one needs to do it manually, or else one needs to use some intelligent programs to recognize and import the data.

An Example:

The weather data that we have worked on are data from weather stations in Eastern Oregon. The data have been collected daily for a period of 44 years. Data are in fixed-width format without delimiters. There are two datasets: precipitation and temperature. The temperature dataset has two data variables: maximum temperature and minimum temperature. Data files for both the temperature and precipitation data sets contain data of one weather station for the whole period of time (44 years). One line of text data contains one month of data. Each month has 31 data values (regardless of which month). The dates in that month that do not exist are filled with the missing value symbol ‘M’. Missing value symbols are also used to represent the data whose values were lost or not gathered during the collection process. For the temperature dataset, there are two data variables. Therefore, there are two consecutive lines of data, which are for the same month of data. The first line is for maximum temperature, the second line is for minimum temperature, as shown the example below.

```
YR MO     1  2  3  4  5  6  7  8  9 10 11 12 .. 24 25 26 27 28 29 30 31

====--== -- -- -- -- -- -- -- -- -- -- -- -- .. -- -- -- -- -- -- -- --

1953  1TX  M  M  M  M  M  M  M 56 56 53 38 42 .. 39 51 43 40 35 40 47 43

1953  1TN 19 21 21 21 25 22 23 30 34 21 29 31 .. 27 26 26 24 27 25 27 33

1953  2TX 40 39 46 44 42 40 44 47 48 44 41 46 .. 44 45 50 55 43  M  M  M

1953  2TN 27 28 32 20 28 26 31 31 20 17 21 19 ..  9 11 16 23 29  M  M  M
```

Precipitation data file has only one line of data for each month.

2.1.3 Binary file

The data are stored in a standard binary format, which can be read and written by some utilities or library functions. The advantages of this format are (a) the data can be loaded quickly, and (b) it usually requires less space to store. The disadvantages are (a) accessing the data can be impossible without the utility program or function library, and (b) it may be hard to move the data file to different computer architectures and operating systems.

An example: NetCDF (network Common Data Form) is a binary data format for multi-dimensional datasets. The dataset is stored as a big chunk of data values, which have the same orientation format as a C array. The data format is self-describing. That is, the data contain information about themselves in the file header. The information at the header tells how many dimensions the dataset has and gives information about attributes of variables of the dataset. NetCDF utilities and functions have the ability to read and understand the format of each dataset from reading the header. Another interesting feature of NetCDF is that the NetCDf data format is machine- and platform-independent. That is, the data stored in the NetCDF files can be copied or downloaded to be used on any machine without any modifications or additional work. The NetCDF data format also has its corresponding text format. The text format is described using the Common Data form Language (CDL). Additional information about NetCDF can be found at (Unidata, 1999).

2.2 Imperfections in the data

Data mining requires analyzing historical data. Therefore, it is unavoidable that the data will have various imperfections. Examples of kinds of imperfections are given below:

2.2.1 Missing values

Missing values can arise randomly from time-to-time, or they can exhibit patterns over time. In our grasshopper data, missing values for precipitation and temperature could occur when a large storm overflowed the rain gauge, when it was impossible for a technician to read the instruments, or when a weather station was closed for some period of time. Missing data was recorded in the formatted text file by a special value of "M". In many cases, weather data was missing for longer periods of time: a few weeks, months, or even years. Missing values have to be treated carefully. Occasional, random, missing values can sometimes be "imputed" – that is, their values can be filled in by using an approximated value such as a default value, the most common value, the most probable value, or the average value. In addition, some learning algorithms, such as decision tree algorithms, can handle occasional missing values under the assumption that the values are missing for completely non-systematic reasons. However, if the missing values are too dense, it is typically not appropriate to replace them with imputed values or to treat them as randomly missing. In some conditions (eg., when the results of a medical test are missing because the doctor chose not to perform the test), the absence of data should be handled as if it were a condition in itself, and the missing value can be assigned a neutral value such as "unknown" (Frawley et.al., 1991). However, In the worst case, records containing missing values must be discarded. In our grasshopper problem, we discarded entire weather stations if they had too many missing values for too many years. For the remaining missing values, we applied decision tree learning methods that could ignore them.

2.2.2 Partially Missing Values

There are cases in which the data are "partially missing". In other words, the data have not been gathered completely, but we have partial information about their values.

The example in our weather data case occurs when the precipitation gauge at a weather station could not be read for several days. When this happens, the precipitation gauge just accumulates all of the rain that has fallen. When the gauge is finally read (and emptied), we obtain the total precipitation for the time period since the gauge was last read. The results were recorded in our data files using a special "subsequent value" indicator. For example, if the gauge was not read for three days and then the fourth-day reading showed 150 hundredths of an inch, the precipitation data would be recorded as "5 S S S 150". Here, the "S" means that the recording for that day was skipped. Therefore, 150 was the value of the precipitation, which occurred for the period of these 4 days. We handled this situation by dividing the 150 evenly among the four days.

2.2.3 No Value

A special data value "M", was also used in fixed-format text files to represent dates that do not exist. For example, the 29th of February in non-leap years will not have any data, but it might be necessary to fill in a value in order to maintain symmetry of the arrays of data. Similarly, the 30th and 31st of February never exist, so their precipitation and temperature values were always recorded as "M". Our preprocessing routines detected and ignored these cases.

2.2.4 Noise and Uncertainty

Data values can also be measured inaccurately, which leads to noise and uncertainty in the data. For example, most real-valued attributes are only measured to a fixed degree of precision. All types of attributes may be entered incorrectly into the computer. All learning algorithms for real-valued attributes have built-in mechanisms for handling uncertainty in such attributes, because otherwise there would be great risk of overfitting very small differences in such attributes. The most uncertain values in our grasshopper project were the grasshopper infestation measurements. These are based on a crude field measurement in which a technician walks through a field, visualizes a square foot some distance in front, and counts the number of grasshoppers that hop out of that square foot as the technician walks toward it. This is repeated 18 times, divided by 2, and the result is the estimated number of grasshoppers per square yard. The technician performs several of these measurements over a geographical region, and then draws a map of the infested areas. This map is not the result of performing this measurement on a fixed sized grid, but instead it is based on the technician's measurements combined with his or her view of how the grasshoppers are distributed based on local geography, land use, and so forth.

One way to overcome uncertainty is to combine several uncertain values to obtain a less uncertain aggregate value. We handled the spatial uncertainty of the infestation data by analyzing larger regions so that fine-grained errors would not cause problems.

2.2.5 Missing attributes

If essential features are completely absent from the data base, it may be difficult or impossible to solve the data mining problem. In our grasshopper data, we will see that probably the most important attribute that could be measured would be the date on which the grasshoppers began hatching from their eggs and emerging from the ground. However, this attribute has not been measured on a state-wide scale. Another important attribute that was not measured was the soil temperature. Sometimes new features can be defined that can approximate the missing attributes. For example, in our study, we assumed that air temperature was highly correlated with soil temperature, and we used it instead. We also worked very hard to define an "estimated hatching date" feature, as will be described below.

2.2.6 Dynamic data

A fundamental characteristic of most databases is that their contents are changing over time. This may mean that the data mining analysis can become useless if it is not performed rapidly. For example, in our grasshopper project, an important goal is to provide advice to farmers in time for them to use this advice to decide whether to spray their crops with insecticides. Those predictions should be made in the late spring using the latest weather data. If there are delays in getting this weather data entered into the computer (as there are currently), then the predictions will not be available in time for them to be useful.

Chapter 3 Feature Definition and Extraction

Bishop (1995) has defined the meaning of feature extraction as the following. “One of the most important forms of pre-processing involves a reduction in the dimensionality of input data. At the simplest level this could involve discarding a subset of the original inputs. Other approaches involve forming linear or non-linear combinations of the original variables. Such combinations of inputs are sometimes called features, and the process of generating them is called feature extraction.”

3.1 Issues in Feature Extraction

There are three important issues related to feature extraction: (a) reducing the dimensionality of the data, (b) defining features useful for prediction, and (c) incorporating background knowledge. These issues are actually issues of searching for the most useful feature set from the input feature space of the problem under study.

In this chapter, we will make a distinction between the *input feature space* and the *representation space*. The input feature space is the space of features in which the input data was originally represented. The representation space is the space of features in which the training examples will be represented for input to the learning algorithm. The representation space is the space that results from the feature extraction process. Modifying the representation space changes the set of predictive hypotheses that can be constructed by the learning algorithm.

3.1.1 Reducing Dimensionality

When the input feature space is very large, it is important that the feature space be reduced to obtain a feature set that can be used to build a predictive model. The reduction of the number variables has to be done without losing significant classification information. From another perspective, the search for a small number of derived features is an effort to understand the important aspects of the input data, and the success of the classifier is limited by the success of the feature extraction effort.

The goal of feature extraction is to increase or maximize class separability in a lower dimensional (mapped) space. In most cases, a learning algorithm will perform better with a lower dimensional space. A smaller space usually allows learning algorithms to yield smaller hypotheses. And small hypotheses are more likely to make correct future predictions. A complicated hypothesis can have some parts that fit the noise or irrelevant information in the input data, and this will cause the hypothesis to make prediction errors on new data points (where these particular noisy or irrelevant parts have changed). We call this phenomenon ‘overfitting’. Moreover, it is impossible for learning algorithms to do exhaustive search in a high-dimensional space (i.e., the space of predictive hypotheses). Most algorithms must instead exploit some search scheme or heuristic in building hypothesis (e.g., the greedy scheme is commonly used). Without feature extraction, it is nearly impossible for the learning algorithm to combine the input features to construct an accurate hypothesis.

3.1.2 Defining Useful Features

Defining useful features involves constructing new feature sets and evaluating their usefulness. Constructing new feature sets requires modification of the feature set.

Consider an application that we want to predict the number of grasshoppers from the weather data in the previous year, and we have daily weather data that we summarized into monthly data. Assume that it is true that summer is the reproduction period for grasshoppers. If the summer is dry, it is likely that the grasshoppers will not have enough food to grow enough to be ready for reproduction soon enough. Therefore, the period of time and the number of grasshoppers that contribute to the reproduction of a new generation will be low. Hence, the population of grasshoppers in the following year is more likely to be low. In this case, monthly data will not describe the problem so well. This is because the development period of grasshoppers from nymphs into maturity spans for over three months. Therefore, the amount of rainfall in a single month will not be able to tell very much. However, if we sum the amount of rainfall from July to September, we will have a feature which relates to the problem better than the original features. If we do not provide this feature, the learning algorithm will need to discover it, which is a difficult search problem.

It is often possible to construct a large number of features that each incorporate domain knowledge and that each summarize several raw input features. As a result, there is usually a need to evaluate these features to choose a subset of them to provide to the learning algorithm. This is the problem of "feature selection". Kohavi and John (1998) categorize feature selection into two approaches: the filter approach and the wrapper approach.

Filter Approach – This selects features using a preprocessing step, based on the training data. There are two major classic algorithms in this approach. "FOCUS (Almuallim and Dietterich, 1991; Almuallim and Dietterich, 1994) is a feature selection algorithm on noise-free Boolean domains. It exhaustively examines all subsets of features, selecting the minimal subset of features sufficient to determine the label value for all instances in the training set. The Relief algorithm (Kira and Rendell, 1992; Kononenko, 1994) assigns a relevance weight to each feature, which is meant to denote the relevance of the feature to the target concept." (Kohavi and John, 1998)

Wrapper Approach – Filter approaches to feature subset selection do not take into account the biases of the learning algorithm. In some cases, measures of selection can be devised that are algorithm-specific, but they will not work well with other algorithms. In the wrapper approach, the feature subset selection is done using the learning algorithm as a black box. The feature subset selection algorithm generates subsets of the features, runs the learning algorithm on part of the training data, and evaluates the resulting predictive model on the rest of the training data. Unfortunately, for a set of n features, there are 2^n possible subsets, so it is not feasible to try them all. Instead, heuristic searches (e.g., greedy forward addition, backward-deletion, or best-first addition and deletion) are employed (Kohavi and John, 1998).

3.1.3 Incorporating Background Knowledge

In many applications, the input feature space is very large and there is an unlimited number of feature modification/construction operators. The search for the useful features is therefore intractable. Moreover in many applications, are only a small number of training examples. In such cases, the evaluation of the features cannot be done reliably. It is also possible for the feature selection process to overfit the training data. In this aspect, background knowledge is very important for guiding the construction of useful features. Background knowledge, sometimes called domain knowledge, is the human expert's knowledge about the problem. The knowledge can be direct knowledge or inductive knowledge. Direct knowledge is the knowledge that is proved to have relation with the problem. Inductive knowledge is the knowledge that is induced from other known facts and knowledge. It is simply a hypothesis. If the learning algorithm can successfully use inductive knowledge to make accurate predictions, this can provide additional evidence to support the correctness of that knowledge. Often, the expert has background knowledge that is not strong enough to be used directly in creating features. However, it can be used to narrow the search space for useful features. Donoho and Rendell (1998) call this fragmentary knowledge, and they discuss many different forms of fragmentary knowledge.

3.2 Methods for Feature Extraction

There are three approaches towards making useful features. The first approach executes search algorithms to apply operators to modify the input feature space. The advantage of this approach is that it uses no biases. Therefore, it can help us discover unexpected features. However, this approach has the limitation that the searching process can be intractable, since the input feature space can be very large and there are infinitely many ways of applying the modification operators. Therefore, currently, this approach is limited to minor modifications of the feature space. Background knowledge and fragmentary knowledge can be added to improve the capability of searching. Hu (1998) discusses issues related with this approach.

The next approach is semi-automated feature extraction. This method uses background knowledge to pick a feature modification algorithm, and the algorithm will then construct the features automatically.

The last approach is manual feature extraction based on background knowledge. Despite many years of research on feature extraction, this is probably the most commonly-applied method and the most successful one. In this section we will discuss the last two approaches. We will also discuss the special issues involving feature extraction from time-series and spatial data.

3.2.1 Semi-automated Feature Extraction

In this approach, specific algorithms related to a particular class of problems are designed to extract the features. These algorithms automatically search for useful features relevant to this particular problem class. This approach has been quite successful in certain kinds of aplications. The examples are described below.

LeCun et al. (1989) and Waibel (1989) use time-delay neural networks (TDNNs) and space-displacement neural networks (SDNNs) to learn feature detectors for temporal and spatial data in hand-written character recognition and speech recognition applications. These methods learn feature definitions that are applied uniformly throughout a time series or image by convolving the feature definition with the raw data.

Closely related to time-delay neural networks is the work of Mallet et al. (1998). Their method adapively constructs wavelets (a kind of local kernel feature extractor) to compute important local features for spectral data analysis applications.

Perrin and Petry (1998) use lexical contextual relations to discover features for textual data. Their model exploits the inherent structure of narrative discourse to extract context-dependent lexical relations. It makes use of extra knowledge such as the lexicon, knowledge of linguistic structures, and domain knowledge.

Numerical taxonomy methods rely on a similarity measure over numerical values and the use of a distance measure to cluster data and identify useful features. These methods implicitly assume that objects can be represented naturally in terms of continuously valued variables. When a numerical value is needed, it is necessary to transform nominal features into continuous ones. Mephu, Nguifo and Njiwoua (1998) use the lattice framework to do such feature transformations.

All of these adaptive methods require a large amount of training data, because they are using the training data to make decisions about how to transform the data (into useful features) in addition to using the data to construct the predictive hypothesis. For example, TDNN and SDNN methods require thousands or tens of thousands of training examples of speech and images in order to learn good feature extractors. A small data set does not provide the statistical power to learn such complex structures from th raw data.

3.2.2 Manual Feature Extraction

Manual feature extraction is used when we wish to define features manually. This can occur in two cases. First, if we have substantial background knowledge, manual feature extraction provides a way of incorporating that background knowledge. Second, if there are very few training examples, automated and semi-automated methods will not work, so we must define the best features that we can manually. This approach takes a lot of time, because there are no standard tools for aiding the process. This is an important topic for future research.

3.2.3 Time and Spatial Feature Extraction

3.2.3.1 Time series feature extractions

Time series feature extraction involves scanning over the time dimension(s) to construct features that summarize the time series. For example, the feature might compute the sum, average, or maximum of some parameter in the time series (e.g., total rainfall, average high temperature, maximum temperature). The most complex such feature is the degree day feature which first computes, for each day, the number of degrees by which the high temperature exceeds a given threshhold. It then sums these quantities over a period.

An even more complex kind of feature that was particularly useful in the grasshopper problem is based on finding a pattern and then measuring some property related to the instances of that pattern. For example, we defined a feature called “largest summer rainstorm”, which was defined as the largest amount of rainfall received during 5 consecutive days during the summer. Note that this feature assumes that “summer” is well-defined, but it in turn may be defined in terms of the first and last days that the daily high temperature exceeded a certain threshold. Another complex feature was “intermittent summer rain”, which was the average of the lengths of the periods without rain in the summer. If the rain falls intermittently, then the periods between rainfalls will be brief, and this feature will compute a small value.

In some cases, there is too much noise and random variation in the time series. This can make it difficult to define good feature extractors (e.g., to define the start and end of summer). To reduce this problem, we can first convolve a Gaussian kernel with the time-series, which smooths each data point by taking a weighted sum of the points before and after it in the time series (where the weights are defined by a gaussian probability density function).

3.2.3.2 Spatial feature extraction

Spatial feature extraction is similar to time-series feature extraction, except that time periods are replaced by spatial regions, and temporal patterns are replaced by spatial ones. For example, in the grasshopper problem, we defined regions in two ways: either as a regular grid of rectangular cells or as a circle of fixed radius centered as a given point. We then computed statistics of these regions such as the fraction of the area of the region that is infested. In spatial data, neighborhood relationships are important. For example, in a grid cell, we could define a feature that indicates whether there were any infestations in any of the 8 cells that surround the given grid cell. Another kind of relationship that is important is spatial data is the nearest neighbor relationship. For example, if a grid cell does not contain a weather station, we might define a feature that defines a daily high temperature in terms of the high temperature measured at the nearest weather station. We could also define features such as distance to the nearest economic grasshopper infestation.

In the grasshopper problem, we actually have a time series of spatial maps, defined by the grasshopper infestation maps over the 44 years of the study period. Hence, we can define spatio-temporal features such as the number of years in which a given cell has been infested, or the number of years since the given cell was last infested, and so on.

3.2.4 Tuning the features

Often, a feature definition involves some parameters or thresholds that need to be defined (and that may need to be adjusted). For example, the degree day feature is defined in terms of a temperature threshold, which must be specified. These parameters can be tuned in two ways. One way is to try different values, extract the features, give them to the learning algorithm, and see how accurate the resulting hypothesis is. The other way is to try different values, extract the features, and then examine the extracted features to check them for consistency with background knowledge. For example, if a "summer" feature determines that it was still summer in December, then the temperature threshold is not defined properly (at least in Oregon).

Chapter 4 Learning Algorithms

4.1 The Learning Problem

To define a learning problem, we need to define the task that we would like the program to learn and the performance measure for that task. In this paper, we are concerned with prediction tasks, and the performance measure is related to the accuracy of the predictions made by the learning algorithm.

There are two major kinds of learning problems, supervised learning and unsupervised learning. Supervised learning involves learning an unknown function f that maps f rom an input space X to an output space Y. The teacher provides training examples in the form of (x,y), where $y = f(x)$. The y values can be discrete in which case they are called “classes”, and we think of f as “classifying” the x value into one of the classes. Hence, f is also called a “classifier”. The learning algorithm analyzes the training examples and outputs an hypothesis h. A probabilistic classifier is an hypothesis p that estimates the probability that $f(x) = y$, for each value of y: $p(y|x)$. In addition to classification problems, where the y values are discrete, the y values can also be continuous, in which case the learning process is called “regression”. A probabilistic regression model computes the probability $p(y|x)$.

The second form of learning, unsupervised learning, involves identifying structure within a set of points drawn from some space X. There is no teacher who provides y values. One form of unsupervised learning is clustering, where the learning algorithm attempts to decompose the data into a set of clusters and then (possibly) define each of the clusters. Another form of unsupervised learning is density estimation, where the learning algorithm attempts to find a compact representation of the probability distribution of the data points $P(x)$. In general, unsupervised learning involves trying to find a model that explains how the observed data could have been created. This is called a “generative model” of the data.

The grasshopper infestation problem is the supervised learning problem. The x values describe the previous year’s grasshopper population and the weather throughout the year, and the y values define the severity of the infestation predicted for the coming year. The previous chapter focused on the problem of identifying good features to represent the x values. We should mention briefly that there is a similar problem of defining the classes to describe the y values. This is usually much simpler, because the y values are typically one-dimensional, whereas the x values belong to a high-dimensional feature space. Nonetheless, we will see below that proper definition of both the input features and the output classes was critical to the success that we achieved.

4.2 The Fundamental Tradeoff

As mentioned in the beginning of this chapter, learning is the problem of improving performance on a particular task based on experience. Experience in a learning task is measured by the number of training examples. Algorithmically, learning can be seen as the task of exploring a large hypothesis space to find a hypothesis that performs well on the task that program is trying to learn. Different learning algorithms represent hypotheses differently. For example, decision tree algorithms represent them as decision trees, neural network algorithms represent them using neural networks, and so on. The hypothesis space of a learning algorithm is defined to be the set of all hypotheses that the algorithm could ever output. For example, for decision tree algorithms, the hypothesis space is the space of all decision trees where the number of leaves in the trees does not exceed the number of training examples.

4.2.1 Tradeoff between Number of Examples, Size of Hypothesis Space, and Accuracy of Result

In a learning problem, the learning algorithm must choose one hypothesis out of its hypothesis space. The training data must present enough information for the algorithm to make the right choice – that is, to choose an hypothesis with low error rate. If the hypothesis space is very large, then either a large amount of data is required, or it is very likely that the learning algorithm will output an inaccurate hypothesis, because the learning algorithm will have very little guidance from the training data. This relationship can be summarized by the following result from computational learning theory which relates the error rate on the test data ε, the number of training examples m, and the size of the hypothesis space $|H|$ as follows:

$$m = 1/\varepsilon\,[\log (1/\delta) + \log |H|]$$

This states that if a learning algorithm searches an hypothesis of size $|H|$ and find an hypothesis that makes no errors on the m training examples, then with probability $1-\delta$, the hypothesis will have error rate less than ε when applied to predict the y values of new data points.

The relevance of this for our grasshopper prediction problem is that we have only a small amount of data. Therefore, to obtain reasonable predictive power, our learning algorithms can only explore a small hypothesis space. We can keep the hypothesis space small by keeping the number of features small (because the hypothesis space typically grows superexponentially in the number of features). We can also keep the hypothesis space small by using simple algorithms such as linear regression, or by requiring the hypotheses to be very simple (e.g., by pruning decision trees to keep them very small). Finally, we can keep the hypothesis space small by defining a small number of output classes (e.g., “infested” versus “not infested”) rather than by trying to predict the degree of grasshopper infestation (either area or numbers of grasshoppers) exactly.

4.2.2 Overfitting

In the previous chapter, we discussed the problem of overfitting. This problem arises when a learning algorithm considers hypotheses that are too complex for the amount of data that is available. The fundamental tradeoff discussed above indicates that the result will be (with high probability) an hypothesis with a high error rate.

Here is an example of overfitting. Suppose a cooking equipment company wants to increase its sales. The company decides to advertise on television, so it purchases advertisements during the day-time shows for one month. The result is that the sales increase. We will assume that this increase was a result of the advertising. Now, in the second month, the company decides to buy more advertising time, so it purchases additional advertising during prime time. Suppose that in the second month, sales triple, but that this is just a coincidence related to some other effect (e.g., a price increase by their competitors). The company mistakenly concludes, however, that prime time is a better time for advertising their cooking product. So, the company cancels all the day-time advertisements and invests a lot of money to double its prime time advertisements. The result is that sales shrink.

In this example, the company considered two hypotheses: "advertising improves sales" and "prime-time advertising improves sales a lot". The first hypothesis appears to be less accurate on the training data, but it is actually more accurate in the long run. The second hypothesis was too complex for the amount of data that was available (just two months of data).

If the company conducted a few more months of experiments where it shifted the time in which the advertisements were shown, it would be able to gather enough data to show that prime-time advertising was no more effective than day-time advertising. Nevertheless, there is always a possibility that some external event would coincidentally raise sales whenever prime-time advertisements were shown and decrease sales whenever day-time advertisements were shown. Hence, while more data can reduce the risk of overfitting, it can never eliminate it entirely.

4.3 Decision Tree Algorithms

Mitchell (1997) describes decision tree learning as follows: "Decision tree learning is one of the most widely used and practical methods for inductive inference. It is a method for approximating discrete-valued functions that is robust to noisy data and capable of learning disjunctive expressions." Figure 4.1 shows a decision tree for deciding whether to play tennis based on the weather. A decision tree is evaluated by starting at the root node. The root node of this tree tests the "Outlook" feature, which can take on three values: Sunny, Overcast, or Rain. The tree is evaluated by following the "branches" labeled with these values. For example, if the "Outlook = Sunny", then the next node to be evaluated will be the "Humidity" node. If the "Outlook = Overcast", then evaluation moves to the leaf node, which gives the output of the decision tree, "yes".

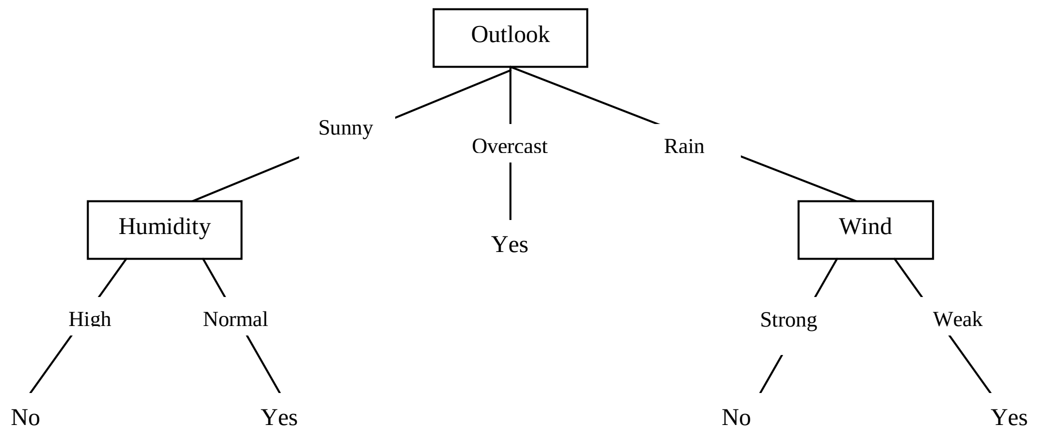


Figure 4.1 A Decision tree for deciding whether to play tennis (Mitchell, 1997)

The decision tree in Figure 4.1 is equivalent to the following disjunctive expression, which tells the conditions under which we should play tennis.

(Outlook == Sunny $\wedge$ Humidity == Normal)

$\vee$ (Outlook == Overcast)

$\vee$ (Outlook == Rain $\wedge$ Wind == Weak)

4.3.1 Growing Decision Trees

Learning algorithms for decision trees usually “grow” them recursively by starting at the root, choosing a feature to test, and then partitioning the training examples based on that feature, and calling the algorithm recursively. The recursion terminates when there are no more features to test, or when the number of available training examples at a node becomes too small.

Although the tree above shows only features that have discrete values, continuous-valued features can also be used. An internal node for a continuous feature tests the value of the feature against a cutoff value. If the feature’s value is less than th cutoff, it branches to the left child. Otherwise, it branches to the right child.

To choose which feature to test in an internal node (and which cutoff value to use, if the feature is continuous), the decision tree algorithms use the entropy reduction (also called the information gain or the mutual information) between the parent node and the child nodes.

The entropy of the examples S at a node is defined as

$$\text{Entropy}(S) = \sum_{i=1}^{k} -p_i \log_2 p_i,$$

where, k is number of classes, and p_i is the fraction of examples in S that belong to a class i. Suppose the node has c children, and that out of the n examples at the node, n_1 of them are sent to child 1, n_2 of them are sent to child 2, and so on. We will call the examples that belong to child j, the set S_j. Then the entropy reduction is defined as

$$\text{Gain} = \text{Entropy}(S) - \sum_j n_j/N \ \text{Entropy}(S_j)$$

The algorithm will grow the tree at each node by first computing the entropy of the current node. Then, for each feature (and each of the possible cutoff values for continuous features), the algorithm will compute the entropy of all child nodes and weight those entropies to compute the entropy gain. The feature (or feature + cutoff) with the largest entropy gain will be chosen to be the test at this node.

4.3.2 Pruning Decision Trees

Typically, the decision tree grown by the above algorithm will be very big, and there is a great danger that it will overfit the data. Most decision tree algorithms therefore prune the tree by choosing an internal node, deleting all of its descendants, and converting it into a leaf node. There are many different pruning algorithms. We employed the decision tree growing and pruning algorithms from the Splus statistical package. These methods allow us to specify the number of leaves in the tree, and the algorithm will prune the tree to achieve that size while still fitting the training data well.

4.4 Regression Tree Algorithms

Decision trees are used when the target y value is discrete or symbolic. If the target value is continuous, a regression tree can be used instead. A regression tree has a split at each internal node just like a decision tree. However, at each leaf of a regression tree, the predicted value is a continuous value. Regression tree learning algorithms choose splits at each node and calculate predicted values at the leaves with the goal of minimizing the squared error of the predicted *y* values on the training data. As with decision trees, the usual approach is to grow a large tree and then prune it to an appropriate size.

Chapter 5 Grasshopper Infestation Prediction

The problem of predicting future infestations of grasshoppers is very difficult. Lockwood (1997) summarizes the various theories and methodologies that have been explored and reports that none of them has yet succeeded in modeling the population dynamics of grasshoppers in a large-scale system. Nonetheless, encouraged by our collaborators, Kathleen Johnson and Dick Jackson of the Oregon Department of Agriculture, we decided to see how well modern machine learning tools would work on this problem.

A major difficulty with any study of this kind, where data are extremely limited, is that there is a danger of "overfitting" the available data – that is, of trying so many different methods and different configurations of those methods that one of them eventually appears to explain the available data. To avoid this, we divided the available data into 3 periods. The data from 1954 to 1978 is called the "Training Data Set". The data from 1979 to 1988 is called the "Validation Data Set". Finally, the data from 1989 to 1997 is called the "Final Test Data Set". During model development, feature construction, feature selection, and so on, we train the models using the training data set and test them with the validation data set. The results of testing with the validation data set are used to decide whether the model has successfully learned something from the training data set. Then out of the models that we have developed, we choose the best configuration and test once again with the final test set.

We have tested a wide range of different models. We describe five of them here. The first four were not able to make any useful predictions. The fifth model, however, provides some useful predictions, so we describe it in more detail. The fifth model appears to be the first model ever discovered with any ability to predict grasshopper population dynamics.

5.1 Linear Regression and State-wide Models

5.1.1 Modeling and Methods

The first model that we studied is called the state-wide model. Our goal was to predict the total number of acres that are infested with grasshoppers. We adopted this task because we thought it might be easier than making predictions within smaller regions, because of the uncertainty involved in the detailed infestation maps. We also wanted to test the hypothesis that the El Nino phenomenon might predict grasshopper outbreaks through its influence on the climate of the Pacific Northwest.

To construct weather features, we decided to choose four representative weather stations in Eastern Oregon. We chose stations that were centrally located and which had very few missing values in their weather attributes. We built two sets of weather features: The November feature set and the June feature set. The November feature set contained the monthly averages of the daily weather data in August, September, October, and November. The June feature set contained the monthly averages of the data in August, September, October, November, March, April, May, and June. We added to both feature sets some features describing the infestation acreage of the previous year, the treatment acreage of the previous year, and several climate indexes such as SHIP1 and NINO that measure the El Nino Southern Oscillation. The list of climate indexes we used appears in Appendix B.1.

We chose to use linear regression as the predictive model, and we measured our predictive accuracy using the squared error between the actual total acreage infested and the predicted total acreage infested. Because we have aggregated all of the data for the entire state into a single number, the total infested acreage, we have only one training example for each year. Hence, we had to carefully control the number of features to avoid overfitting. We employed step-wise regression (with forward selection) to introduce one feature at a time. We also experimented with an exhaustive search for the best four features. To guide feature selection, we used leave-one-out cross-validation within the training set.

We experimented with decomposing the prediction problem into separate prediction problems for non-economic and economic levels of infestation. We call this the "split prediction" approach. Two regression models are trained, one to predict acreage of non-economic infestation, and the other to predict acreage of economic infestation. Their predictions are then summed to obtain a prediction for the entire state.

We also experimented with dividing the state into two regions based on the dominant grasshopper species. There are several counties in the south part of Eastern Oregon where virtually all of the grasshoppers belong to the species *Camnulla Pellucida*. This grasshopper exhibits different behavior, and perhaps different population dynamics, than other species. For example, the females cluster together on the tops of small hills to lay their eggs. So we subdivided Oregon into "Camnulla counties" and "non-Camnulla counties", and tried training two different models to predict total infested acreage within these two regions.

In total, we considered three approaches: (a) direct prediction (single model for total infested acreage), (b) split prediction (separate models for non-economic and economic infestations), and (c) split Camnulla prediction (four different models: non-economic + non-Camnulla, non-economic + Camnulla, economic + non-Camnulla, and economic + Camnulla).

5.1.2 Results

Unfortunately, none of the models that we trained had any predictive power on the validation data set. The problem appears to be that we are overfitting the data extremely easily. We plotted the squared error measured on both the training and validation data sets as the stepwise regression process introduced additional features. There is evidence of overfitting immediately after introducing the second feature, and overfitting increases as more features are added. This suggests that no useful state-wide acreage model can be learned from this data using linear regression.

5.2 Decision Tree and Grid Site Model

5.2.1 Models and Methods

The next approach that we investigated was to divide the state into a grid of 1200 cells and to make separate predictions within each cell. The point at the center of each grid is called a site. We considered two discrete prediction tasks: (a) two classes: no infestation versus any kind of infestation, and (b) three classes: no infestation versus non-economic infestation versus economic infestation. In both tasks, we are trying to predict the infestation at the site point.

To define weather features for each site, we faced a problem. The weather stations are very sparsely distributed, so most of the cells do not contain any weather station. We solved this problem by choosing the weather data for each day of the year based on the nearest weather station that had data for that day (as long as the station was within 6.77 kilometers of the site). Nonetheless, there were many sites for which the nearest weather station was more than 60 km away. Once we had obtained the daily weather data, we aggregated this to obtain the following features: monthly averages of precipitation, minimum temperature, and maximum temperature, average precipitation in spring (March, April, and May), annual average precipitation and temperature, previous-year infestation and previous-year treatment area, elevation of the site, the previous-year infestation of the neighbor cells, and a set of El Nino climate indexes.

We chose decision trees as the predictive model, and we trained them using the data described above. We evaluated the accuracy of the learned models by measuring the percentage of correct predictions that they made on the validation data set.

5.2.2 Results

The decision trees tended to always predict that there would be no infestation at all. This is because infestations are rare events. However, even when the trees made positive predictions, these were often wrong or reflected overfitting of the data. This is a well-known problem that arises in learning problems where one class is much more common than the others. In such cases, a learning algorithm can get a very high percentage of correct predictions by always predicting the most frequent class.

5.3 Regression Trees and the Region Site Model

5.3.1 Models and Methods

In this experiment, we took the same 1200 cells defined in the grid site model but changed the goal of prediction. Instead of trying to make the discrete prediction of the type of infestation at the center of the cell, we chose to predict the area of infestation within the cell. The input features were the same except that we included the area of the previous year's infestation in the 8 cells that surround the current cell.

We chose regression trees as the model, and we evaluated the learned models in terms of the squared error between the predicted area and the actual area of infestation within each cell.

5.3.2 Results

The results of this model did not improve over the previous model.

5.4 Decision Tree and Weather Station Site Model

5.4.1 Models and Methods

The poor results from the grid site model led us to question the wisdom of trying to make predictions at sites that are located far away from any weather station. Hence, instead of defining a set of cells and then trying to find weather stations near those cells, we decided to start with the weather stations and define circular regions around those weather stations. For each of our 75 weather stations, we defined a circular region with a radius of 6.77 kilometers. For each circular region and each year, we computed the area within that region that had no infestation, non-economic infestation, and economic (or treatment) infestation. We used these areas to compute a label for each region as follows. If the area of non-economic or economic infestation was less than 5% of the area of the region, then the region was labeled as having no infestation. Otherwise, the label was the type of infestation with the largest area. We call this the "area infestation label." Hence, under this measure, a region could have only 6% of its area infested and still be treated as having economically important levels of grasshoppers. We adopted this measure to force the learning algorithm to pay more attention to the rare infestation events.

To define the input features, we worked very hard to incorporate knowledge of the grasshopper life cycle. The full list of features is given in Appendix C. One subset of the features was designed to predict the success of egg laying: the longest period without rain in the summer (smld), the largest 5-day rainfall period in the summer (smlr), a measure of the degree to which the rain was intermittent during the summer (smir), and the date of change from summer to fall (ss1, based on temperatures). Another very important feature is the predicted hatching date (htsp), which was based on computing the number of degree days (using a threshold of 55°F) since the beginning of fall. The date on which the total number of degree-days exceeds the hatching threshold, is declared to be the predicted hatching day (see B.4 - 11,12). Finally, we included a set of features designed to measure the effect of weather on the nymphs after they have hatched. This set includes features such as the accumulated cold degree days (using a threshold of 50°F) after hatching date until the start of summer (htcd), the date of occurrence of a 5-day cold spell subsequent to hatching (csp), and the sum of the daily precipitation for the 35 days after hatching date (htpc).

The predictive model in this step was again the decision tree. We considered two different prediction problems: (a) predicting economic versus other (i.e., non-economic + none), and (b) predicting infestation (economic + non-economic) versus no infestation. We measured the predictive power of the trees in terms of the percentage of correct predictions.

5.4.2 Results

Unlike the previous decision trees that we had grown, we found that these trees made sense – the structure of the trees corresponded, as least partially, to our background knowledge. However, the predictive performance was still poor. We tried pruning the trees using both Splus's cost-complexity measure and also manually pruning the trees using our background knowledge. However, this did not produce any improvement in performance.

5.5 Probabilistic Decision Trees and the Weather Station Site Model

5.5.1 Models and Methods

The fact that the decision trees from the last experiment "made sense" led us to consider the possibility that we were not evaluating the predictive power of the trees properly. In other words, perhaps the trees were working fine, but our criteria for success were incorrect. In the machine learning research community, the criterion of percentage of correct predictions has been used almost exclusively. But in other areas, such as computational finance and medical decision making, other measures have been employed. These other measures make sense in cases where even a small improvement in predictive power can lead to large savings in money or human lives.

We therefore explored three other evaluation criteria. The first is a measure of the quality of the probabilistic predictions produced by the decision trees. A decision tree can be used to compute class probabilities. In other words, at each site during the test period, the decision tree can predict the probability of no infestation, of non-economic infestation, and of economic infestation. We can compare this probability to the actual probability of those infestations. To compare probabilities, we use the Kullbeck-Liebler divergence, which is a measure of the distance between two probability distributions. Let p_i be the predicted probability of infestation level i (where i = none, non-economic, or economic), and q_i be the actual probability (which is the percentage of the number of training examples with infestation level i in the particular leaf of the tree). Then the KL divergence is defined as

$$D(p||q) = \sum_i p_i \log (p_i/q_i)$$

The second measure that we considered was the Kappa (κ) statistic. This is the measure of agreement between two "judges". In this case, the two judges are the actual label of the region and the predicted label of the region. Let n_{ij} be the number of sites in which the first judge labels them using label i and the second judge uses label j. The κ statistic is defined in terms of two other quantities, θ_1 and θ_2, as follows:

$$\theta_1 = \sum_i n_{ii} / N$$

$$n_{i+} = \sum_j n_{ij}$$

$$n_{+i} = \sum_j n_{ji}$$

$$\theta_2 = \sum_i \ (n_{i+} \ n_{+i}) / \mathrm{N}^2$$

$$\kappa = (\theta_1 - \theta_2) / (1 - \theta_2)$$

The value of κ ranges from –1 (complete disagreement) to +1 (complete agreement). A κ of zero indicates that there is no relationship between the labels of the two judges. The purpose of the statistic is to correct for the situation where one label is much more common than another label, so that if two judges make random decisions in proportion to the frequency with which the labels occur, their κ score will still be low.

In our experiments, we have modified the κ statistic to handle probabilistic predictions as follows. The quantity n_{ij} becomes the sum over all of the sites of $q_i * p_j$, where these are defined as above for the KL divergence measure.

The third evaluation measure that we considered was the cost savings achieved by making treatment decisions according to statistical decision theory. A farmer must make a decision about whether or not to treat a region (e.g., a field). According to statistical decision theory, the farmer should make this decision based on the probability of infestation, the cost of treatment, and the cost of not treating. Suppose the probability of an economically significant infestation is p. There are four cases to consider, as shown in the following 2-by-2 table:

	Infested (p)	Not Infested ($1 - p$)
Treat	T	T
Don't Treat	D	0

If the farmer decides the treat the field, it costs \$T to treat it, regardless of whether there is an infestation. Assuming the treatment eliminates all possible crop damage, the total cost of this decision is \$T. If the farmer decides not to treat and there is an infestation, he will suffer a cost of \$D. But if there is no infestation, he will suffer a loss of zero dollars. We can compute the expected cost of each action. The expected cost of treating is always \$T, regardless of the probability p of an infestation. The expected cost of not treating is \$$pD$. So the farmer should treat if $T < pD$. To evaluate a probabilistic prediction, we can compute the total loss over the 75 weather stations of making the optimal decision according to the predicted value of p for each station and compare this to the total cost of the optimal decision based on the background probability of infestation, which is to never treat. A probabilistic decision tree is useful if it saves money compared to the policy of never treating the fields.

The exact values of T and D change depending on the market price for crops and the cost of pesticides (and other costs associated with treatment), so we will report the results for various T:D ratios.

We also changed slightly our definition of the target classes for training the decision trees. We call the new target value the max infestation, and it is defined as follows. If the region around a weather station contains both economic infestations and non-economic infestations, and if each of these (taken separately) occupy more than 5% of the area in the region, then the region is labeled as having economic infestation, even if the total area of non-economic infestation is greater. If only one of the two types of infestation occupies more than 5% of the region, then the region is labeled with that infestation type. And if neither type occupies more than 5% of the region, then the infestation type is "none".

We used the same input features for this model as were used for the previous one.

There are two ways that we can train models and make predictions. The first method is the "batch" method, in which we train a decision tree using a fixed period of time (e.g., the training data set), and then use that tree to make predictions for each year of the validation data set or the final data set. This method is computationally simple, but it means that when we are predicting the t^{th} year of the validation data set, we are ignoring data from years 1 through $t - 1$. Hence, the other way to make predictions is to train a model for each year using all of the data available from all previous years. We call this the "year by year" method. We will compare both methods below.

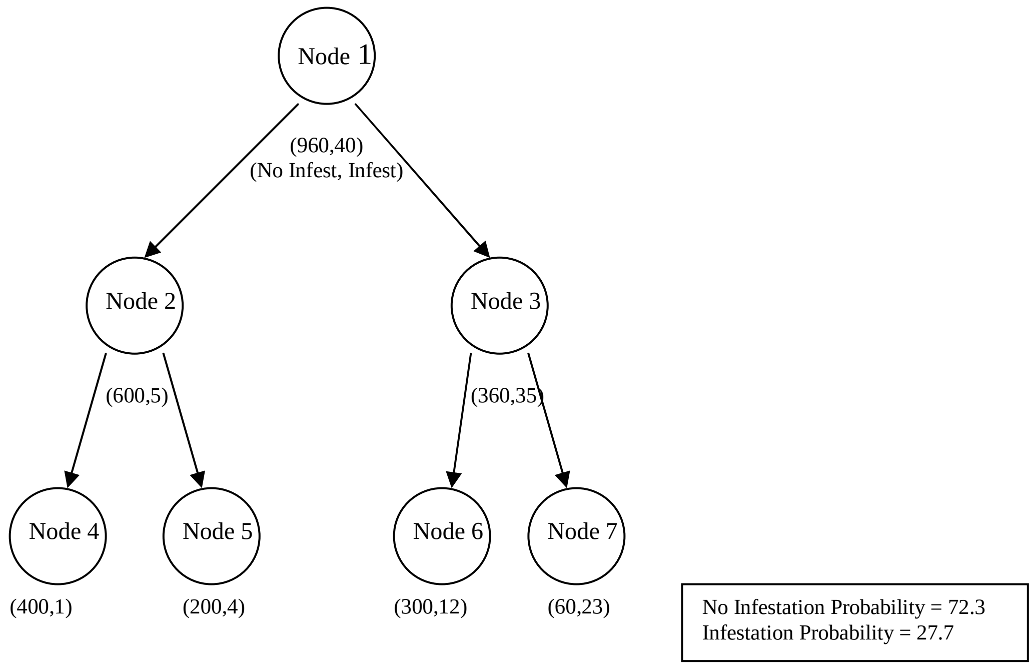


Figure 5.1. A probabilistic decision tree (at node 7)

Figure 5.1 shows an example of a probabilistic decision tree. Consider node 7. If an example is "sent" to this node and we predict that there will be no infestation, we would be correct 72.3 percent of the time, but we would miss out on the fact that the tree has improved the accuracy of its probabilities of infestation. If we predict infestation, we would only be correct 27.7 percent of the time. Since infestations are rare events, if we use a standard method of decision tree construction, the tree will continue to grow and split the training examples until it can predict infestation events. Unfortunately doing so with the small amount of data that we have available (and the complexity of grasshopper population dynamics) will most likely end up building a model that is too complicated. If we then use a standard method of decision tree prediction with this problem, the tree will end up predicting nothing or predicting wrongly. However, instead of predicting yes/no, if we predict probabilities, we will allow the decision tree to take advantage of information gain it has discovered during learning, which is the core concept of decision tree learning algorithms.

It is useful to have a word to describe the set of examples that are "sent" to a given leaf (i.e., that will be classified by that leaf when the tree is used to make predictions). We will refer to such examples as a "zone." All the examples in the same leaf belong to the same zone. To make use of this tree in practice, the farmer will check the decision tree to see in which zone his/her farm belongs. Then s/he will check what s/he should do by following recommendation for each zone.

We introduce the term “zone” because it can make more sense than the term “leaf” for a person who is not familiar with decision trees. Every site within a given zone should be treated the same way, depending on the ratio between T and C. For example, if the cost is 1:10, we will decide to treat zone 7, because we will spend 83, but expect to save around 23*10 = 230.

5.5.2 Results

5.5.2.1 KL Divergence on the Validation Data Set

Figure 5.2 shows the KL divergence of a probabilistic decision tree compared to the KL divergence of a decision tree that predicts no infestation with probability 1.0. A small value for the KL divergence means that the probability of infestation is being predicted accurately. Cost-complexity pruning was applied to prune the tree to have various numbers of leaves. The horizontal axis shows the behavior as the number of leaves varies. We see that the KL divergence is smallest for a decision tree with only five leaves, and that this tree performs much better than the “no infestation” tree. In the graph, the “no infestation” line is not flat, because of missing values. Each path from the root to a leaf of the decision tree tests certain features. If those features are all present at a site, then the tree can make a prediction, and so that site is included in this diagram. The result is that the number of sites included varies depending on the degree to which the tree was pruned.

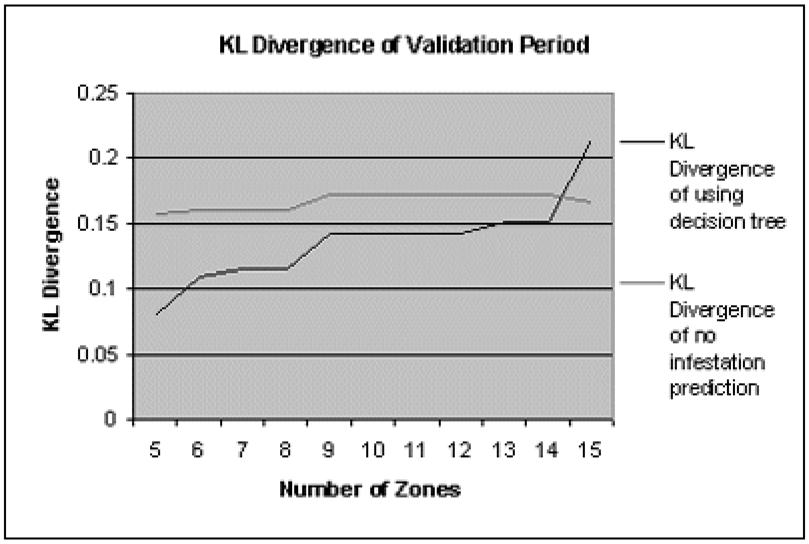


Figure 5.2. Decision tree trained with the training data set improves KL divergence in the validation data set.

5.5.2.2 More Data Yields Improved Models

A question that concerned us was whether it is reasonable to expect a single model to make accurate predictions over the entire period for which we have data. There are many reasons why the factors determining grasshopper population dynamics might have changed during this period. Farming practices might have changed. The mix of crops (and hence, the range of foods for grasshoppers) might have changed. The effects of previous years of spraying pesticides might have permanently reduced the grasshopper population. Changes in land use from farming to housing might lead to changes in grasshopper habitat and behavior. The relative population of different species of grasshoppers might have changed. In short, there is no particular reason to believe that a single model would have good predictive power throughout the period. Indeed, these kinds of changes may explain why it is so hard to find a good model.

If the dynamics are changing over time, then we might obtain more accurate predictions by using only the *k* most recent years of data to train a model for making a prediction in the *k+1*st year. Figure 5.3 shows the results of a study in which we computed the probabilistic distribution distance (KL Divergence) of year-by-year predictions using different values for *k*. The results show that the best approach was to train on all available data. This is weak evidence that over the time period covered by our data, the dynamics are not changing too much. Certainly, it turns out to be better to consider all of the data rather than to ignore data from years in the past.

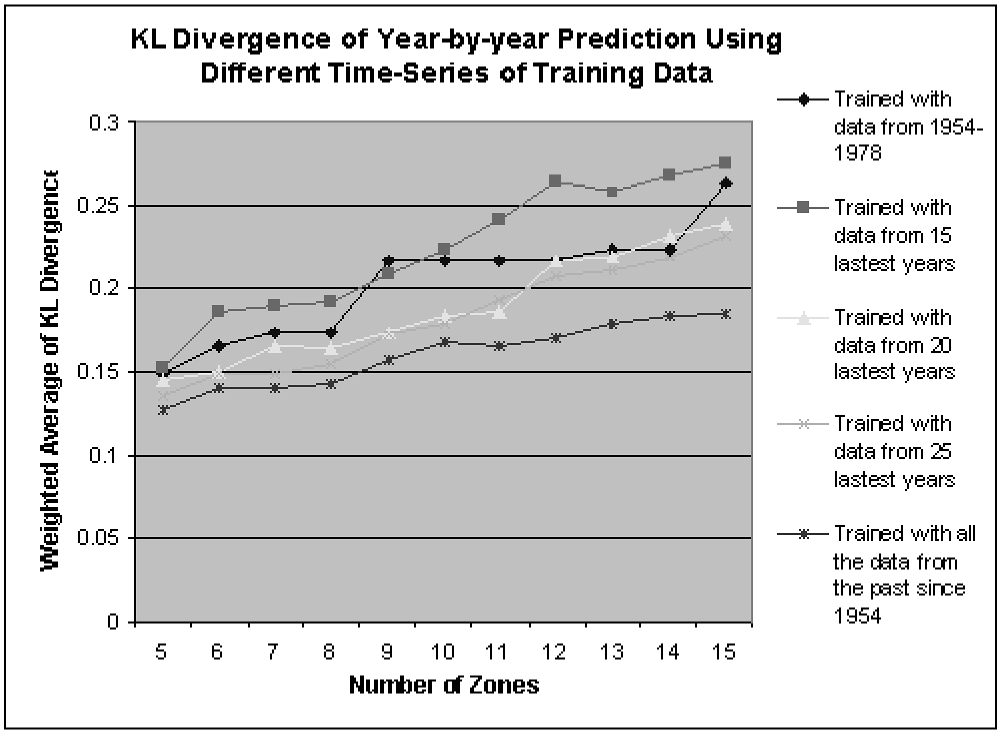


Figure 5.3. Evaluation of the effect of different amounts of training data and different numbers of decision-tree leaves, on the KL divergence between predicted and actual probability of infestation.

However, from figure 5.3, remark that the probabilistic distribution distance of year-by-year prediction using probabilistic decision tree is not better than predicting that there are always no infestation. The no infestation predicting gives KL divergence of year-by-year prediction around the same as or slightly lower than when we predict using decision tree that is trained with all the data from the past. This is because in year-by-year prediction, there are more likely that each leaf of the tree have no infestation. Moreover, the examples that have economic infestations are usually in small leaves. Therefore, when we do weighted summation of KL divergence, the divergences of making wrong prediction of no infestation predicting are weighted down. Therefore, the KL divergence of no infestation predicting becomes lower, despite that the predictions miss out all the infestation events. In this case, it seems not so fair to calculate KL divergence of no infestation prediction on the tree. This is because, in fact, it is performance of the decision tree that it can group the examples with economic infestation together in smaller leaves.

5.5.2.3 KL Divergence over the Final Test Period

Figure 5.4 shows the KL Divergence of a decision tree trained on the entire data set from 1954 through 1988 (i.e., the union of the training and validation data sets) and tested on the final test set. The results are very similar to the validation set results: the decision tree is more accurately predicting the probability of infestation than a tree that simply predicts no infestation with probability 1.0.

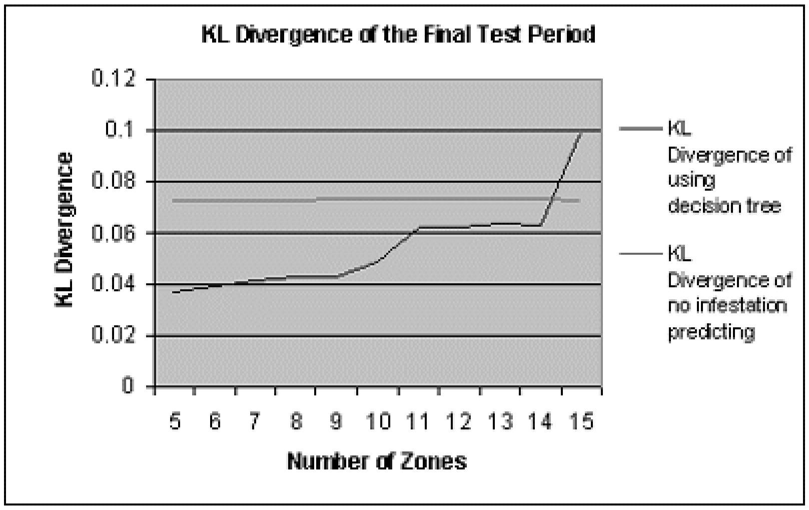


Figure 5.4. KL Divergence in the final test period for a decision tree trained on all the data from 1954 to 1988

5.5.2.4 Evaluation of Probabilistic Decision Trees using Kappa

Our second measure for evaluating the decision trees is the κ statistic. Figure 5.5 shows the κ values of the year-by-year decision trees on the Validation Data Set as a blue line, depending on the number of leaves in the decision tree. The fact that the κ values are greater than zero shows that the decision tree is predicting better than random. However, the values are still quite close to zero. The pink line shows the value of κ that would be attained if $p_i = q_i$ for all classes i. This is the κ score of a probabilistic tree that has a perfect KL divergence score. It is possible to get an even higher κ score if the decision tree sets $p_i = 1$ for the class whose actual probability q_i is largest.

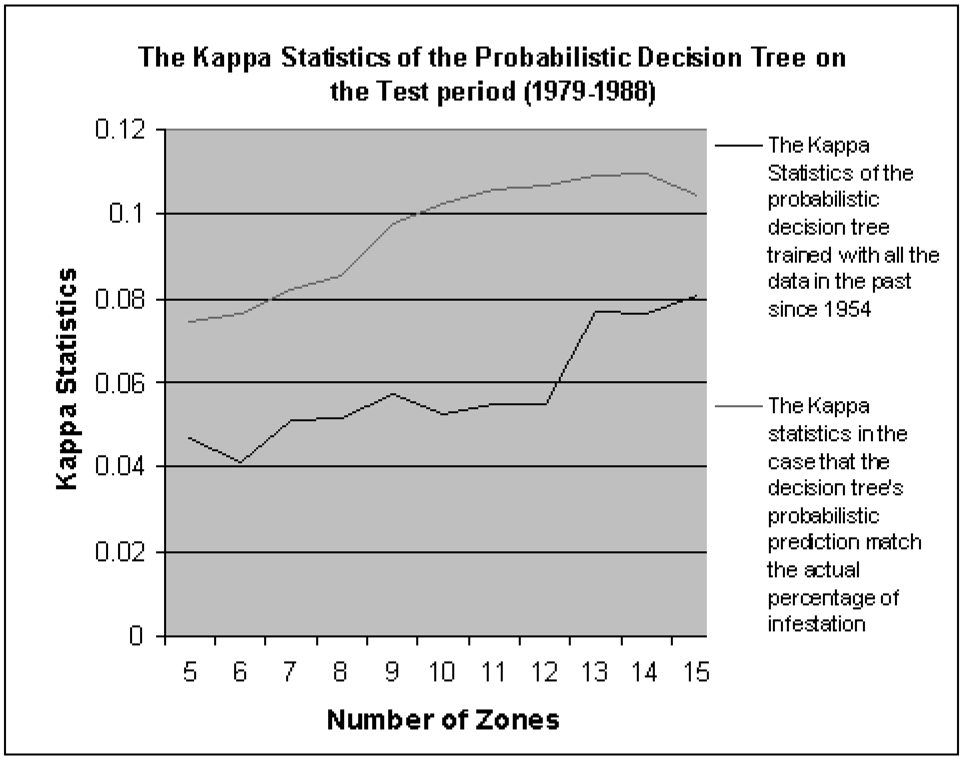


Figure 5.5. Kappa statistics of the decision trees in the test period

Figure 5.6 shows the κ scores for batch predictions over the final test period for a single decision tree trained on the union of the training and validation data sets. Here we see that the κ scores are matching (or in some cases exceeding) the scores of an optimal KL divergence classifier.

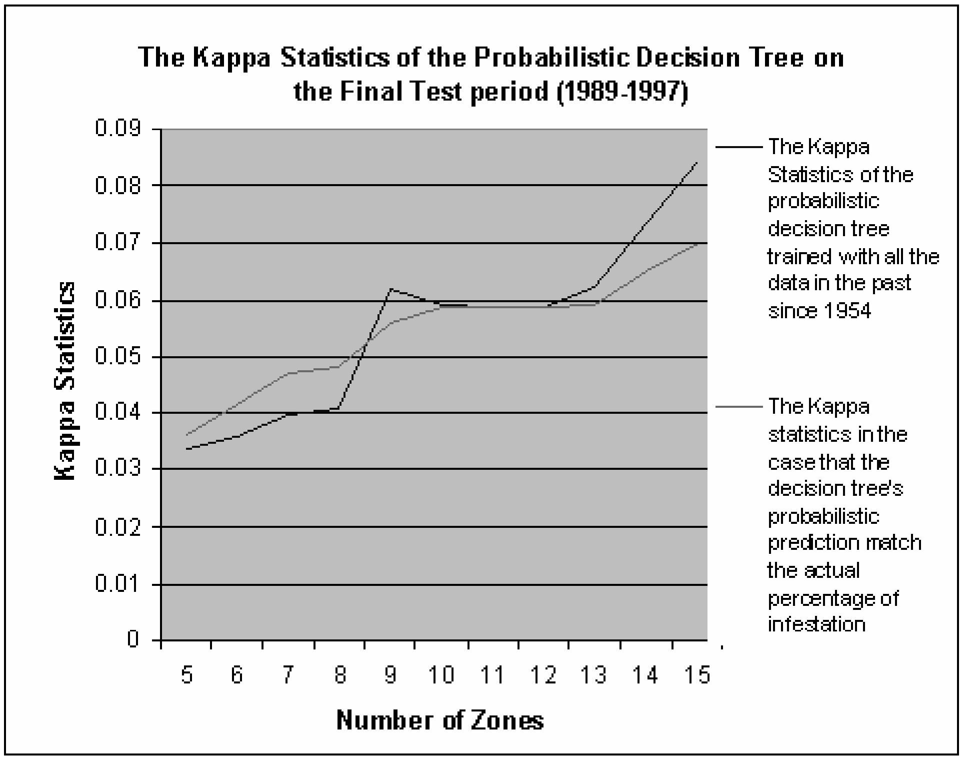


Figure 5.6. Kappa statistics in the final test period of the probabilistic decision tree trained with all the past data collectively

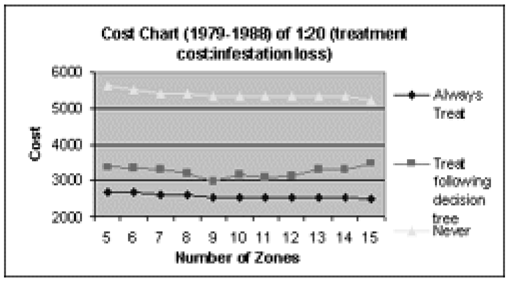
It is interesting that the κ scores improve as the decision trees get larger. This is different from the results for KL divergence, where the smallest trees give the best score. It is also different from the total loss criterion (see below), where again, the smallest trees give the best score. This suggests that the κ score, since it is a modification of the precent correct score, is not the best measure of performance.

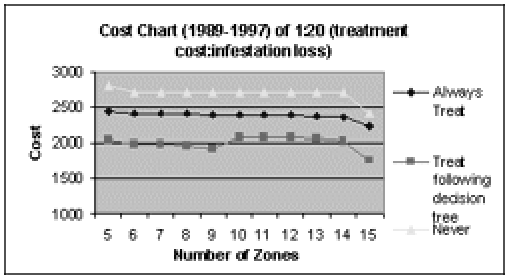
5.5.2.5 Evaluation of Probabilistic Decision Trees Based on Total Loss

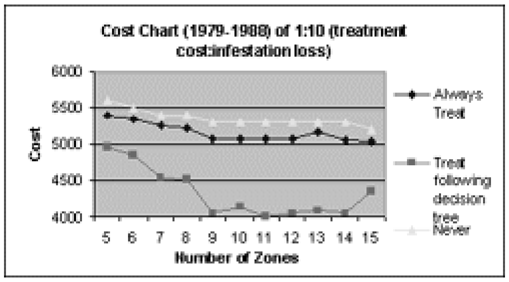
Our final evaluation is based on measuring the financial cost of never treating versus the financial cost of treating according to the recommendations of the probabilistic decision trees. Table 5.1 shows the total cost of year-to-year predictions during the validation period and during the final test period. There are separate graphs for six different ratios of treatment cost T to crop damage cost C. Each graph shows three lines. The yellow line is the cost of never treating; the blue line is the cost of always treating; and the pink line is the cost of treating selectively according to the decision tree predictions (i.e., treating whenever $T < pC$).

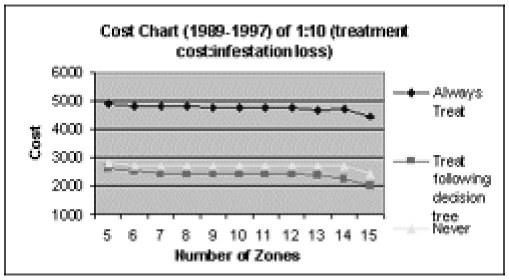
For all three ratios, the policy of treating according to the advice of the decision tree is better than (or indistinguishable from) either of the other two fixed policies. Hence, this suggests that farmers could save some money by following the decision tree recommendations, at least for these ratios of treatment cost to infestation cost. Table 5.2 summarizes this for all three cost ratios using a decision tree with 10 leaves.

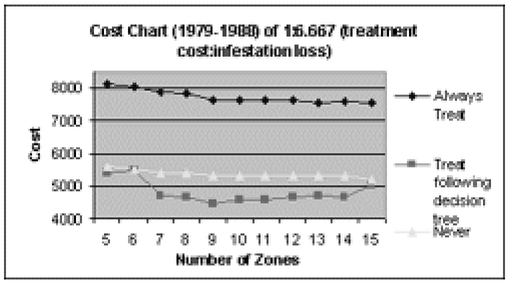
Table 5.1. Plots of the total cost during the validation period and during the final test period.











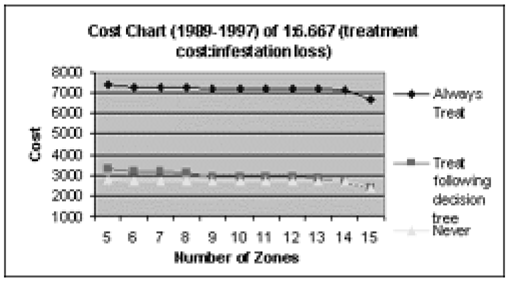
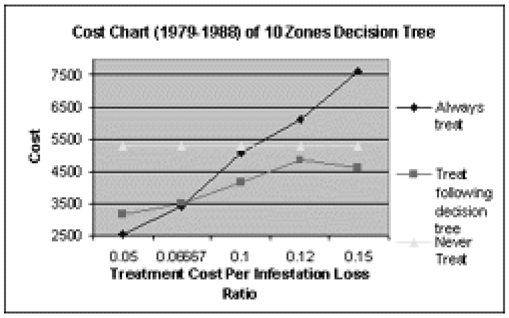
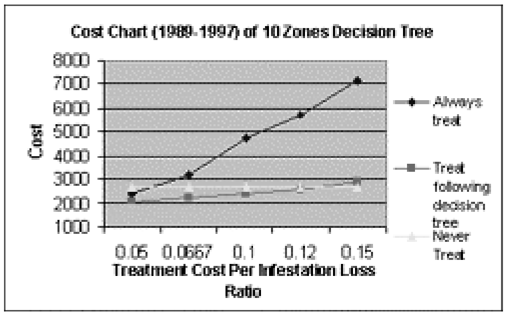


Table 5.2. Plots of Cost and Treatment Cost per Infestation Loss Ratio





5.5.2.6 Visualizations of Predictions

In addition to measuring the performance of the probabilistic predictions on the final test set, it is interesting to plot some of the aggregate predictions made by the trees.   Figure 5.7 plots the actual and predicted total number of sites with some form of infestation over the validation and test period. During the validation period, we see that the predictions have a tendency to lag behind reality. This is typical of first-order time-series models, and it means that the predictions are not very good. In some cases, for example, 1987, we see that the learned model is predicting a large increase in grasshopper infestation when in fact there was a large drop. The learned model behaves much better during the final test period. In many cases, it gets the direction of change correct, even if the magnitude is not correct. Hence, it appears to be capturing the qualitative dynamics of the population even if it is not making precisely the right quantitative predictions. In 1991, it predicts an increase, but not as large an increase as was actually observed; in 1994, it predicts another increase, but this time the increase was less than it predicted.

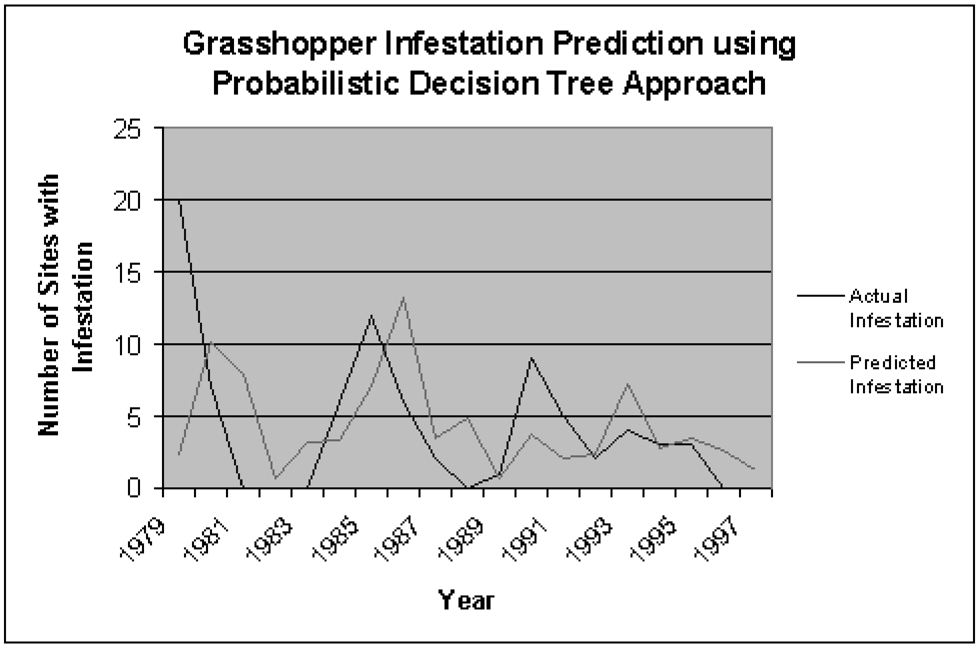
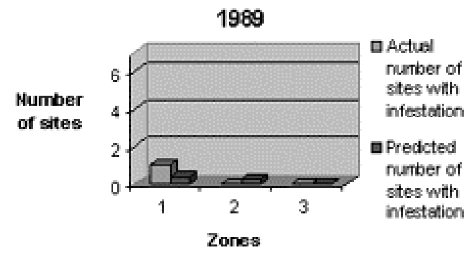


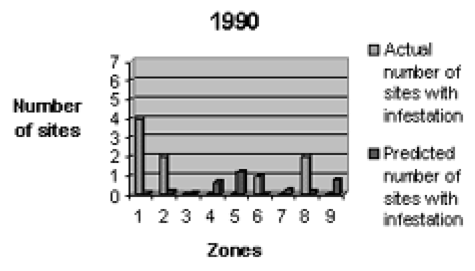
Figure 5.7. Plot of state-wide prediction of the probabilistic decision tree

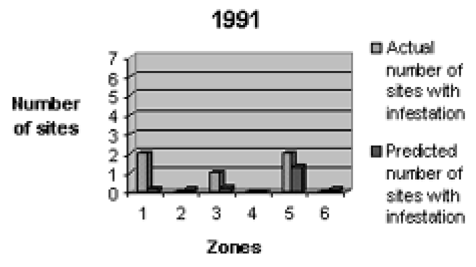
Another way of visualizing the predictions of the model is to consider the individual leaves of the decision tree. We number these leaves starting from 1 (based on the importance assigned to them by the decision tree algorithm) and call them zones, because they correspond to (not necessarily contiguous) regions of the state. Table 5.3 plots a separate bar graph for each year. The number of zones plotted varies from year to year, because in some years (e.g., 1989), none of the higher-numbered zones had any predicted or actual infestations.

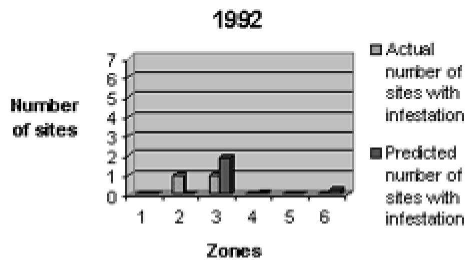
We can see that in some years, the predictions are quite bad. Consider 1990, where zones 1, 2, and 8 have very large numbers of infestations, and the decision tree has underpredicted quite badly. In other years, such as 1994 and 1995, the decision trees made excellent predictions. In most of the years in the final test set, the grasshopper populations were small or medium. There were no population explosions of the size witnessed earlier in the training and validation set periods. It may be that the learned decision tree works better in years with low to moderate numbers of grasshoppers, and that it does not make such good predictions in years with huge outbreaks of grasshoppers.

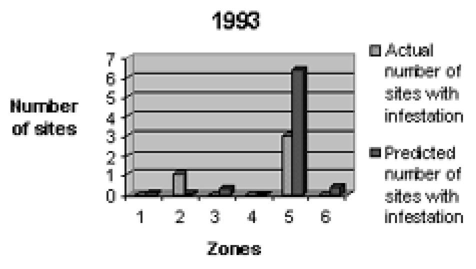
Table 5.3. Bar Graphs Show Prediction of Probabilistic Decision Tree Comparing with Actual Infestation

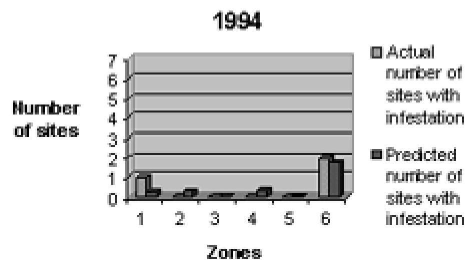


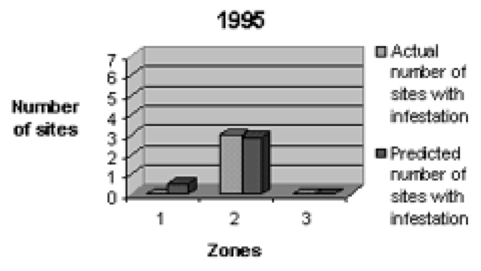


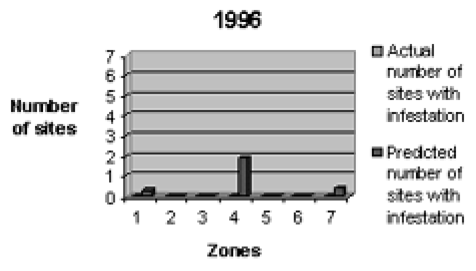


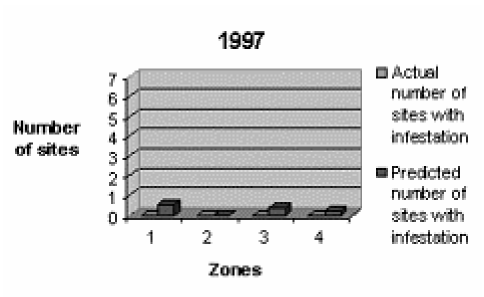











1997
Number of sites
7
6
5
4
3
2
1
0
1
2
3
4
Zones
Actual number of sites with infestation
Predicted number of sites with infestation

Chapter 6 Conclusions and Future Improvements

The studies reported in this paper have shown that it is possible to make useful predictions of grasshopper population dynamics in Eastern Oregon. This was shown using all three of our more sensitive measures of performance: the KL divergence, the κ statistic, and the economic loss.

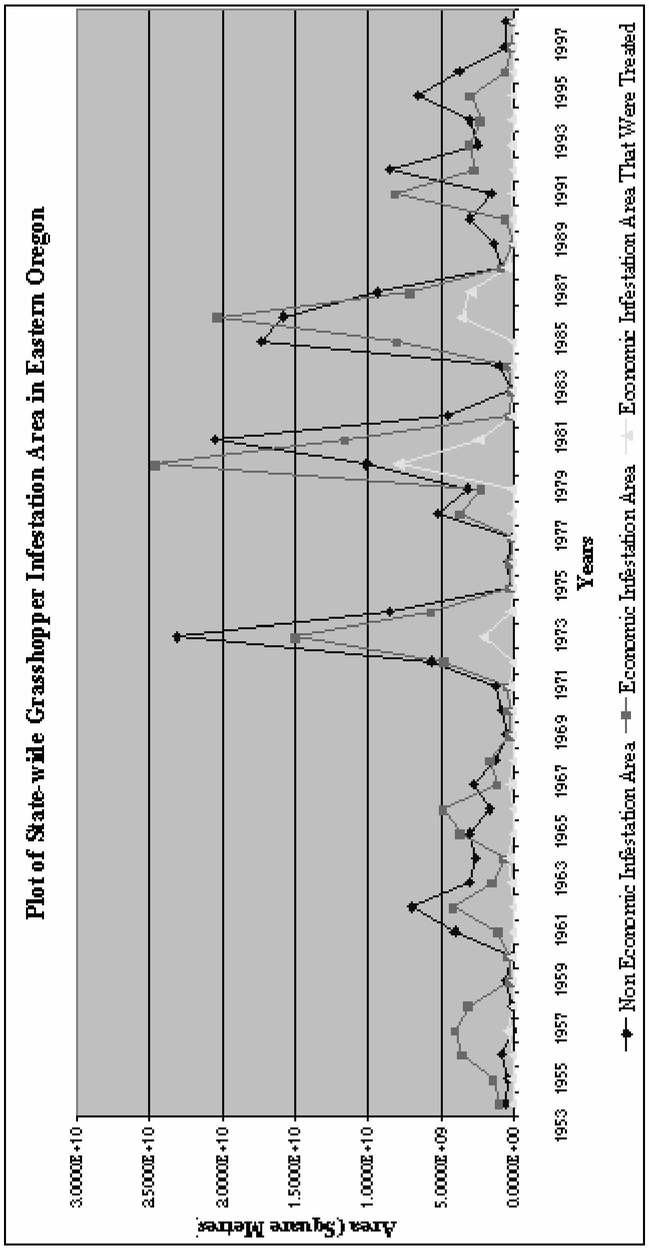
To obtain these successful predictions, three major steps had to be taken. First, we needed to formulate the problem in terms of the weather station site model. This required us to restrict our attention to regions of Eastern Oregon for which good weather data was available. Second, we needed to define and extract features that captured important aspects of the grasshopper life cycle including the egg-laying period, the maturation period, and the weather after hatching. As always, the definition of good features was the most important factor in obtaining successful predictions. The third step was to apply the right criteria for measuring success. The prediction of economic grasshopper infestations resembles other economic decision making problems in which large financial gains can be achieved as long as the model makes predictions that are better than random guessing. In this case, it appears that modest reductions in the costs of grasshopper control can be achieved.

There are many problems remaining for future research. One experiment would be to return to the site grid model and see if the combination of the knowledge-based features and the improved evaluation criteria would allow us to determine whether decision trees are making better predictions there. Another important direction would be to make improved measurements of soil temperature and hatching date to validate and improve the features that were developed for this project. The biggest problem with this project was the absence of good weather and soil temperature data over large regions of the state. Research on grasshopper infestation and many other ecological questions would be substantially advanced if a high-density network of measuring stations could be put into place across the state (and the entire western region of the United States).

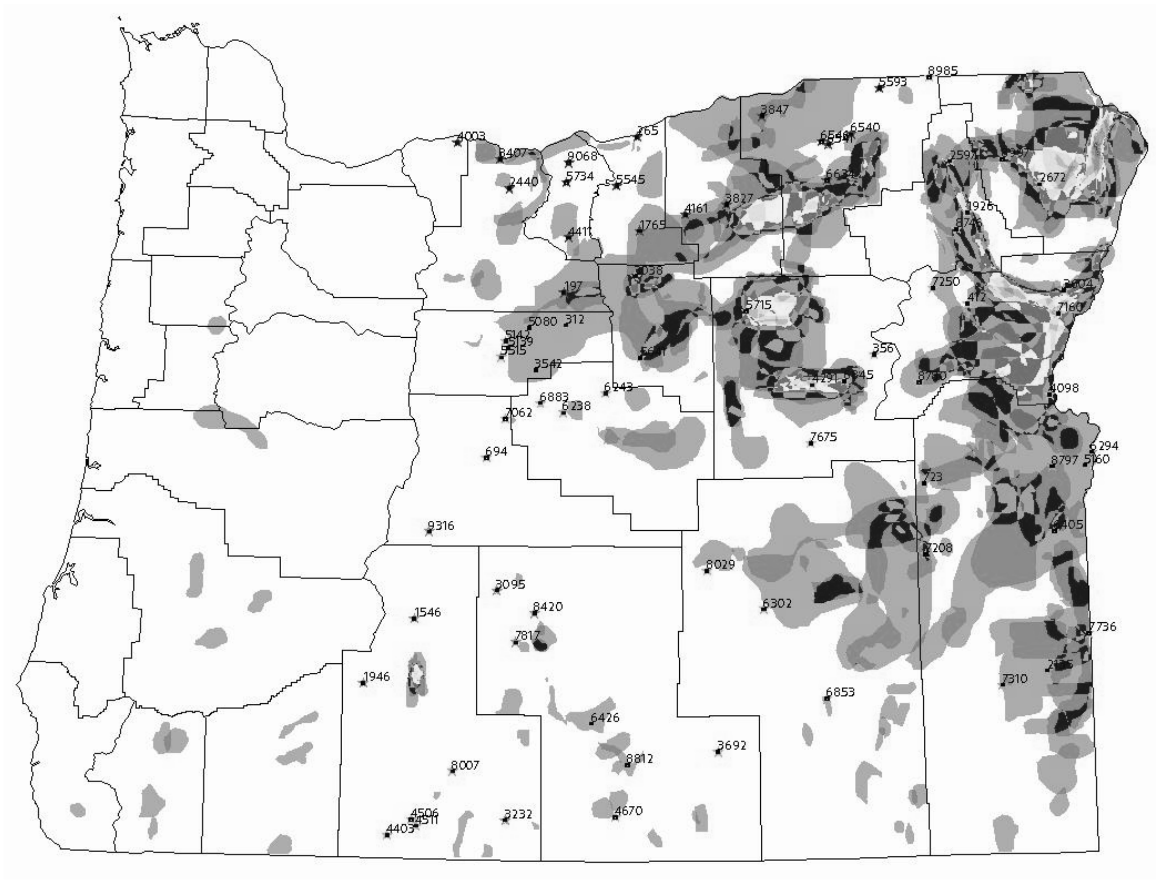
Finally, an important question for future machine learning research is to develop automatic methods for doing many of the steps that we performed manually. It would be wonderful, for example, to have a compact language for describing the data transformations that we have performed. It would also be interesting to try to automatically derive useful features from a qualitative model of the grasshopper life cycle. These are major challenges for future research, but they would allow machine learning methods to be applied to a much wider range of data mining problems than existing tools can currently handle.

Appendix A: Pictures of Grasshopper Infestation in Eastern Oregon Problem

Picture A.1 Plot of Infestations



Picture A.2 Plot of weather stations and the union of infestation coverages in the training and test period. The colors of weather stations indicate different climate zones. The colors of the infestation polygons indicate frequencies of infestation. Red is highest frequency. Grey is the lowest frequency.



Appendix B: Summary of Study, Processing and Modeling in the Grasshopper Infestation Project

B.1 Data Sources

1. Daily maximum and minimum temperature data of the 75 weather stations in eastern Oregon from 1953 to 1997 from the OCS (Oregon Climate Service).

2. Daily precipitation of the 75 weather stations in eastern Oregon from 1953 to 1997 from the OCS.

3. Geographical locations (latitude, longitude, elevation and climate zone) of all the 75 weather stations.

4. ARC/INFO coverage of annual grasshopper infestations surveyed in early July and their treatment areas every year from 1953 to 1997.

5. DEM (digital elevation map) of the USA and Oregon.

6. SHIP-1, SOI, NINO1.2, NINO3, NINO4, NINO3.4, SOI-ANO, SHIP-1-ANO monthly indexes every month from 1953-1997.

7. Oregon counties map

8. Weather data of Oregon in NetCDF files (simulated data, not useful – dropped later)

B.2 Imperfect Data Treatments

1. Fill missing values by getting the data from the nearest weather stations. Done for all the data from all the sites of grid site model, region site model and weather station site model.

2. Fill the subsequent symbols by average out from the subsequent values. Done for the weather station site model. (all other models get the data from this)

3. Prevent the effects from missing values in feature extractions. In weather station site model, we decide that we will not fill he missing values with any interpolations. In feature extraction, we set threshold of the percentage of missing values in extracting features. If the percentage of the missing values is higher than the threshold, the extracted feature will be considered missing value. That is, we preserve the missing value symbols in order to prevent the missing values from contaminating the goodness of data. The percentage threshold we have used is around 5-20% depends on the data. For example, for average temperature and precipitation, we used 5% threshold. For longest dry, largest rain and intermittent rain, we used 20% threshold.

4. Reduce the effects from noise by discarding the infestation in the region or area surrounding the site where the area is less than 5% of the total area belonging to the sites.

B.3 Data Manipulations and Productions

1. Use arc/info to find the annual sum of the infestation, economic infestation, non-economic infestation, and treatment areas in the eastern Oregon, export the result into text files, convert the infestation areas from square meters into acres. (Treatment areas were area with economic infestation and were treated with chemicals in the same year after having found infestation. We always used treatment areas as a factor of study wherever we use infestation in all models).

2. Use arc/info to create a coverage for counties (Crook, Harney, Deschutes, Lake and Klamath) whose majority species of grasshopper is Camnulla (using the Oregon counties map). Use the coverage to create the annual sum of the infestation, economic infestation, and non-economic infestation areas of the Camnulla counties and the non-Camnulla counties.

3. Use arc/info to create an eastern Oregon map from the Oregon counties map.

4. Use Microsoft Excel to manage the infestation data and create last year's infestation data for each year infestation data. (Use row shifting)

5. Use Microsoft Excel and its macros to transform the day of the month and month of the year dimension, of all the maximum and minimum temperature data of the four best weather stations into day of the year dimension.

6. Use Microsoft Excel and its macros to transform the day of the month and month of the year dimension, of all the precipitation data of the four best weather stations into day of the year dimension.

7. Use Microsoft Excel and its formulation to create monthly average weather data from the daily weather data of all year for four selected sites.

8. Use Microsoft Excel to create data files for statewide model using four weather stations, 3827, 3604, 4291 and 8997. The data files contain the annual statewide infestations, monthly average of all the weather data, and SOI, SOI-ANO, SST, and SST-ANO indexes of April, May, June, and November.

9. Import the weather station location into arc/info to create the point coverage for the weather stations. Use arc/info to intersect the weather station point coverage with the infestation coverages. Export the infestation type data at each weather station at each year into text files.

10. Design and implement Site Multidimensional Database classes (SMD), Yearly Multidimensional Database classes (YMD) and Spatial Relation Components using C++ and STL.

11. Use arc/info to create a rectangular grid cover eastern Oregon. There are around 1400 grid points. Each point has distance to its neighbor such that Manhattan distance 1, equals around 12,000 metres. Intersect the grid point coverage with the eastern Oregon map; there are around 1200 sites left. Use these points as a grid site model. Intersect the grid point coverage with the infestation coverages to get the infestation types at each point. (If there is no infestation right at the point, we will consider there is no infestation at the site). Export the grid site geographical data and infestation data into text files. Use SMD and YMD to import the data.

12. Use arc/info to create a net that has the grid points created as in B.3 - 11 centering in each of the cells of the net. Use these cells as a region site model. Intersect the net (or region) coverage with the infestation coverages to get infestation areas inside each region. Export the region site geographical information and infestation data into text files. Use SMD and YMD to import the data.

13. Create a coverage containing circle polygons whose centers are the weather stations. Intersect the circle coverage with the infestation coverages and export the area of infestation polygons inside each circle polygons. Manually combine the areas of different polygons of the same types of infestation inside the same circle polygons to obtain the area of each infestation type surrounding the corresponding weather stations in each year from 1953 to 1997. Use YMD to import the data.

14. Create a function to transform the daily maximum and minimum temperature data files into two sets of data files, one for daily maximum, the other for daily minimum.

15. Use arc/info to compute the distance between weather stations and between weather stations and grid points in the grid site model and region site model.

16. Use arc/info's command "gridtoascii" to create the elevation data file of Oregon from the Oregon's DEM map. Use the geo database to import this information.

17. Create a function to transform the day of the month and month of the year dimension into day of the year dimension. Use this on various occasions.

18. Create a function to transform the daily weather data files into the average monthly data files. Use this on various occasions.

19. Create a function to join two data files so that each record of the joined file has all the attributes of both files. Use this on various occasions.

20. Create functions to get data from netCDF files and project the data into the site model database and save them into the text files in the same format as the weather stations. Compare them with weather station data. Use SMD to convert them into daily files. Found an error on the Oregon NetCDF data files. The unofficial data were produced using the wrong coordinate mapping.

21. Create data in the grid site model and the region site model using SMD and YMD from the NetCDF data. Create all the features have been done on the statewide model including new set of features based on hatching period for grid site model and region site model from the NetCDF data. Check their correctness thoroughly.

22. Create a function to map the daily data from the nearest weather station. If the data at that station is a missing value, find the next nearest weather station, and so on until non-missing data is found. If the nearest weather station whose data is not a missing value is farther than a limit value, assign a missing value into the mapped data.

23. Does the same as 10. But for the annual data instead of for the daily data.

24. Create functions to report infestation type of the yearly infestation features of all the selected sites for all the selected years. This function is to be used with the different infestation data

either infestation at the sites or infestation in the area surrounding the sites. The report function will choose the infestation type using the methods created in 25 and 26.

25. Create functions to determine the infestation type of a specific site from the maximum level of the infestation that occurred in that year at a specific site

26. Create functions to determine the infestation type from the infestation area according to the following definitions. The "Max" infestation for a year is the type of infestation that has the highest infestation level and has an area of infestation larger than a user-defined threshold, that occurred around the site in a given year. The "Area" infestation for a year is the type of infestation that has the largest area in the circle around the site in the circle model for that year and the area is higher than the user-defined threshold.

27. Create the data using 24, 25 and 26. The threshold for the area is 5% of the region belonging to the site.

28. Create an S-plus function to read in the feature values from the feature files and report out the feature values by weather stations, one weather station per one file. The feature values in file's content are ordered chronologically. This is to help analyze the results.

29. Create S-plus functions to do the information gain calculation and information gain performance measurement of the feature in the training data set based on leave-one-out cross-validation. Create S-plus functions to do information gain performance measurement of the features in the training data set based on training data without cross-validation.

30. Use arc/info to create prediction maps using the region site model and grid site model.

31. Use S-plus to create the weather profiles of the top ten best weather stations in the summer and fall period and the winter and spring period. Top ten best weather stations are selected by the percentage of infestation that occurred in the area surrounding the weather stations in the training period. The weather profiles are used to identify the interesting weather features.

32. Write an S-plus function to automatically build the data set from the text files of feature sets produced by the feature extraction program. Write an S-plus function to automatically fill in all the missing values in the data set.

B.4 Feature Extractions

1. Monthly average features of maximum temperature, minimum temperature and precipitation.

2. Elevations of the sites in the site model interpolated from the nearest elevation grid point in the Oregon's DEM map.

3. Mean annual average daily temperature and daily precipitation of each weather station.

4. Frequency of the infestation in a particular (the training) period of each sites of all the models. Use the result to select good sites. Good sites are sites with high infestation frequency. (It has higher number of positive examples in the studies)

5. The percentage of year with infestation in a particular (the training) period of each sites of all the models.

6. In the grid site model, find the number of the neighbor grid points with Manhatton distance 1 which has a particular kind infestation (economic and non-economic) or their combinations. Does the same as above but for the neighbors with Manhatton distance 2.

7. In the region site model, find the sum of the areas of infestations in the neighbor regions with Manhatton distance 1. Does the same as above for the neighbor with Manhatton distance 2.

8. In the grid and region model, find the maximum level of infestations of the neighbor grid points with manhatton distance 1. Does the same for neighrbors with Manhatton distance 2.

9. Design and implement feature extraction classes as mentioned in 5.5.2 Feature Extractors.

10. Create a function to find the largest the changes of temperature in n-latter-days period comparing to n-previous-days period. Create histogram of the sites 355 for the case of non-infestation and infestation years and compare. There were no co-relation was found.

11. Create a function to extract the accumulated hot degree-day in a certain period. Produce the features for summer period, summer-fall period, winter-period, and spring period. The accumulated hot degree-day is calculated using the difference between the average of the daily maximum and minimum temperature and a configurable threshold.

    <u>Algorithm to calculate accumulated hot degree-day</u>

```
Accumulated_hot_degree-day = 0;

From (day_i in day_1 to day_N) {

    Hot_degree-day =  ((day_i maximum temperature + day_i minimum temperature)/2 – threshold).

    If (Hot_degree-day < 0) Hot_degree-day = 0;

    Accumulated_hot_degree-day = Accumulated_hot_degree-day + Hot_degree-day;

}

return Accumulated_hot_degree-day;
```

12. Create a function to predictthe hatching day event by comparing the accumulated hot degree-day at a certain date with the hatching threshold. The accumulated hot degree-days can get the initial value from the other feature. In weather station site model, produce the features for predicting the whether the eggs have enough heat (419.99 F of accumulated hot degree-days) during the fall to get into the diapause period. If not, record how much more heat is needed. In the weather station site model, produce the features for predicting whether and when the eggs hatch using the result of accumulated heat in fall mentioned above and using the temperature in spring. The additional accumulated hot degree-day period required in spring after the egg enters diapause is 106.99 F of accumulated hot degree-days. Experiments in the threshold required in calculating the accumulated degree-day, the threshold of 55 F was chosen over the threshold of 60 F and 50 F, since threshold at 60 F and 50 F gave unrealistic results.

13. Create a function to detect whether there is a cold spell period in some particular period. The length of cold spell is 5 days and the threshold for the temperature is 50 F. In the weather

station site model, the feature was detected for the period after the spring hatching day predicted date until the end of June.

14. Create a function to calculate accumulated cold degree-day. The threshold used in the extraction is 50 F. In the weather station site model, features were calculated for winter period and for the spring period.

15. Create a function to calculate average of the precipitation in a configurable period. In the weather station site model, features were produced for the summer, early fall, late summer and early fall, spring, months of May, June, mid of June to mid of July, early spring (March 22 to April 30) and late spring (May 1 to June 21). Please definition of the late summer and early fall, and early fall in 21.

16. Create a function to calculate average of the daily average temperature (average of daily maximum and daily minimum temperature) in a configurable period. In the weather station site model, features were produced for the summer, early fall, late summer and early fall, spring, early spring, late spring, months of May, June, July, mid of June to mid of July.

17. Create a function to calculate the average of the daily maximum temperature in a configurable period. In the weather station site model, the features were produced for the summer period.

18. Create a function to calculate the average of the daily minimum temperature in a configurable period. In the weather station site model, the features were produced for the summer period.

19. Create a function to calculate the sum of the precipitation in a dynamically configurable period. In the weather station site model, features are calculated for the 35 days period and 50 days period after the hatching day predicted date.

20. Create a function to find the largest rain in a configurable period. In the weather station site model, the features were created to find the largest 5 days rain in the summer period.

21. Create a function to find the longest dry period in a configurable period. In the weather station site model, the longest dry period was searched in the summer period.

22. Create a function to find the intermittent rain. The calculation of this feature is the averaging of the period without rain. If the averaging is high, this means the intermittent-ness of rain is low. If the averaging is low, this means that intermittent-ness of the rain is high. In the weather station site model, features were created for the summer period.

23. Create a function to count the number of days without rain in a configurable period. In the weather station site model, features were created for the late summer and early fall period. Tuning up a good period was done on this period. The periods of Aug 15 to Oct 30, Sep 1 to Oct 30, and Sep 22 to Oct 30, were experimented. The last two periods were chosen. Sep 1 to Oct 30 is called “late summer and early fall”. The period of Sep 22 to Oct 30 is called “early fall”.

24. Write a S-plus function to smooth the curve of daily average temperature (average of maximum and minimum temperature), and find the position where the temperature changed pass the threshold of 55 F, from higher to lower determines the beginning of fall, from lower

to higher determines the beginning of spring. This feature is called “seasons”. Create the feature for weather station site model.

25. Write a S-plus function to produce the smoothen weather profiles and draw the threshold lines to show the season period. Use the function to produce the curve.

B.6 Prediction and Evaluation

1. Create S-plus function to do linear regression as the following. Create S-plus functions to do automatic step-wise linear regression both for direct prediction and split prediction. Create S-plus functions to do automatic exhaustive linear regression.

2. Create S-plus functions to do evaluation of the regression tree and decision tree in the region site model and the grid site model as the following. Create S-plus functions to do regression tree and decision tree test data prediction. Create S-plus functions to do classification evaluation on the regression tree (the evaluation values are the root mean square error of the prediction). Create S-plus functions to do classification evaluation on the decision tree (the evaluation results are the confusation matrices. These are done for both the direct prediction model and economic-non-economic prediction model.

3. Write C++ classes to do the prediction based on the rules manually code in. Write C++ classes to do evaluation of the prediction by calculating confusation matrices over predicted and actual classifications.

4. Write S-plus functions to calculate information gain of the decision tree. Information gain of a decision tree is the difference between sum of entropy at all leaves and entropy at root.

5. Write S-plus functions to do probabilistic difference calculation and to compare between the max infestation and area infestation.

6. Write S-plus functions to evaluate the information gain of all the features in a feature set.

7. Write S-plus functions to do probabilistic distribution distance (KL divergence) calculation.

8. Write S-plus functions to do cost-analysis of using probabilistic decision tree to make treatment decisions.

9. Write S-plus functions to build decision tree using the most-recent training data and calculate KL divergence and weighted sum of KL divergence over the period.

10. Write S-plus functions to build decision tree using all the past data and calculateKL divergence and weighted sum of KL divergence over the period.

11. Write S-plus functions to build decision tree using the most-recent training data and calculate cost analysis over the period.

12. Write S-plus functions to build decision tree using all the past data and calculate cost analysis over the period.

13. Write S-plus functions to build decision tree using all the past data and calculate Kappa statistics over the period.

Appendix C: Feature Set in Weather Station Site Model

C.1 List and Definitions

The following is the list of the features that we used in building the probabilistic decision tree described in section 6.5.

```
 [1] "typeinf.V1"    "typeinf.V2"    "treat"         "flaa"
 [5] "fldd"          "flsp"          "htsp"          "htcd"
 [9] "csp"           "htpc"          "jnaa"          "jnap"
[13] "mayaa"         "mayap"         "spdd"          "sfdd"
[17] "smaa"          "smah"          "smal"          "smap"
[21] "smir"          "smld"          "smlr"          "wtcd"
[25] "wtdd"          "ss1"           "ss2"           "sfnr"
[29] "sfra"          "eflra"         "eflnr"         "sp1ap"
[33] "sp2ap"         "sp1aa"         "sp2aa"         "spaa"
[37] "spap"          "eflaa"         "espaa"         "espap"
[41] "lspaa"         "lspap"         "wtaa"          "wtah"
[45] "wtal"          "wtap"          "espdd"         "lspdd"
[49] "sp1dd"         "sp2dd"         "lasttypeinf.V3" "lastareainf.V3"
[53] "typeinf.V3"    "areainf.V3"
```

General Format of the Naming

[Time Period] + [Extraction Method]

Definition of Time Periods

fl == fall
sm == summer
wt == winter
sp == spring
sf == late summer to early fall (Aug 20 to oct 31)
efl == early fall (Sep 22 to Oct 30)
esp == early spring (March 22 to April 30)
lsp == late spring (May 1 to June 21)
sp1 == late winter early spring (March 12 to April 30)
sp2 == later spring (May 10 to June 21)
jn == june, may == may,

1.1 Definition of Extraction Methods

aa == average of daily average temperature
ap == average of daily precipitation
al == average of daily low temperature
ah == average of daily high temperature
dd == accumulated hot degree day
cd == accumulated cold degree day
sp == sum of the daily precipitation
nr == number of days that it's raining

ra == ap (average of daily precipitation)

1.1 Exceptions

[1] "typeinf.V1" == sitenumber

[2] "typeinf.V2" == year

[3] "treat" == last year treat or not

[7] "htsp" == prediction of hatching date (1 == Jan 1)

[8] "htcd" == accumulated cold degree day after the predicted hatching date

[9] "csp" == date of occurance of coldspell in 5 days period counted from predicted hatching date

[10] "htpc" == sum of the daily precipitation 35 days counted from predicted hatching date

[21] "smir" == average of the intermittent rain in summer (low value means raining pattern are intermittent spreading over the period)

[22] "smld" == longest period without rain in summer

[23] "smlr" == largest rain occurred in 5 days period in summer

[26] "ss1" == the changing date of the season from summer to fall (using Splus's function to smooth the temperature curve and decide the date which the temperature pass from higher to lower than 55F)

[27] "ss2" == the changing date of the season from winter to spring (using Splus's function to smooth the temperature curve and decide the date which the temperature pass from lower to higher than 55F)

[51] "lasttypeinf.V3" == last year's type infestation (highest level of infestation in the area)

[52] "lastareainf.V3" == last year's area infestation (level of infestation that has higher area)

[53] "typeinf.V3" == type infestation (target value)

[54] "areainf.V3" == area infestation (target value)

C.2 Information Gain Performance

Table C.1 Information Gain on Training Set Data

Features	Threshold	PCT-left	PCT-right	Inf. Gain
smap	1.66464	0.956274	0.980477	0.014186
lasttypeinf.V3	0.5	0.9616	0.970554	0.01201

lastareainf.V3	0.5	0.9616	0.970554	0.01201
smlr	13.5	0.957617	0.979167	0.010614
smir	10.5	0.956357	0.922936	0.009535
lspap	1.26206	0.956713	1	0.009034
sp2ap	1.03689	0.956713	1	0.008344
htpc	70.5	0.947674	0.970588	0.007728
mayaa	46.5887	0.952419	1	0.007105
smld	43.5	0.956357	0.945181	0.006544
lspaa	49.8269	0.95297	1	0.006368
smal	51.3232	0.950676	0.940563	0.006261
sfnr	57.5	0.957152	0.926407	0.006223
wtal	31.841	0.952033	0.947695	0.006146
wtap	8.67364	0.958088	0.953857	0.005946
flaa	39.9314	0.945804	0.904762	0.005894
sp1ap	6.81731	0.959236	0.955151	0.00574
mayap	4.59678	0.957036	0.969613	0.005411
smah	77.6342	0.950158	1	0.005077
wtdd	6.25	0.951199	0.954932	0.004779
espaa	41.6562	0.954041	0.97971	0.004763
spap	2.89071	0.957719	0.977778	0.004711
spaa	57.9809	0.955	0.951699	0.004706
sp1aa	41.2019	0.952537	0.976253	0.004556
htsp	192.5	0.948325	0.945178	0.00453
espap	6.7875	0.958809	0.954188	0.004507
jnap	1.38333	0.95674	0.979381	0.004456
smaa	58.0061	0.94996	1	0.004437
csp	511	0.948325	1	0.004219
eflaa	56.5297	0.948675	0.945757	0.004207

spdd	8.75	0.95469	0.988506	0.004128
sfdd	356.75	0.946018	1	0.004072
sp2aa	49.8171	0.954363	1	0.003889
jnaa	53.1334	0.951535	1	0.003868
wtah	48.6988	0.952304	0.942922	0.003823
sp2dd	30.25	0.954363	1	0.003718
flsp	1040	0.956522	0.953952	0.003685
lspdd	33.25	0.95297	1	0.003591
wtaa	42.0294	0.951787	0.949389	0.003463
ss2	135.5	0.953684	0.949287	0.003299
eflnr	28.5	0.957071	0.92	0.00291
sfra	2.89583	0.956577	0.966169	0.00275
ss1	60.5	0.955071	1	0.002673
eflra	6.2738	0.956825	0.952933	0.002399
htcd	-50	0.948223	1	0.002373
fldd	25.25	0.945804	0.925926	0.002336
wtcd	969.5	0.951787	1	0.002289
treat	NA	0.9616	0.962547	0.002216
espdd	1.25	0.954041	0.962433	0.000936
sp1dd	61.75	0.952537	0.951073	0.000902

Note:

1. PCT-left is the percentage of non-infestation examples that have feature value less than threshold.

2. PCT-right is the percentage of non-infestation examples that have feature value greater than or equal to theshold.

Table C.1 Information Gain on the Combination of Training Set and Test Set Data

Features	Threshold	PCT-left	PCT-right	Inf.Gain
smap	1.68902	0.938673	0.97774	0.023105
lasttypeinf.V3	0.5	0.944762	0.959044	0.016525
lastareainf.V3	0.5	0.944762	0.959044	0.016525
smlr	13.55	0.93882	0.971867	0.015258
sp1aa	41.7548	0.933908	0.976364	0.011541
espaa	41.5688	0.935087	0.983645	0.011392
smld	43.5	0.936933	0.921053	0.011139
smir	8.9375	0.936933	0.885077	0.010548
wtah	48.6988	0.934256	0.917542	0.009754
spaa	48.6929	0.935845	0.978166	0.00967
htsp	162.5	0.928809	0.907963	0.009564
lspaa	53.9279	0.933564	0.971681	0.00941
spdd	16.75	0.935702	0.977728	0.008773
mayaa	49.1371	0.932958	0.984076	0.008647
smal	41.4695	0.933824	0.983819	0.008221
wtal	32.0669	0.934142	0.928661	0.007869
ss2	123.5	0.934144	0.917541	0.007619
flaa	38.8791	0.93076	0.870748	0.007584
wtcd	1912.5	0.93349	0.950462	0.007268
sp2aa	55.25	0.934611	0.966184	0.00701
wtaa	33.9286	0.93349	0.892405	0.00696
espdd	1.25	0.935087	0.964444	0.006756
smaa	60.1555	0.93299	0.986486	0.006711
eflra	0.488095	0.938243	0.865854	0.006585
sfdd	631	0.931034	0.974277	0.006255
sp1dd	1.25	0.933908	0.963516	0.005956
jnaa	56.2916	0.932806	0.976974	0.005853

lspdd	138.25	0.933564	0.964286	0.005819
espap	8.85	0.939199	0.935167	0.005816
lspap	1.24038	0.939269	0.987903	0.005378
sp2dd	30.25	0.934611	1	0.005375
wtap	10.8577	0.939626	0.936138	0.005051
flsp	1040	0.940179	0.936753	0.004961
eflnr	39.5	0.938037	0.945713	0.004884
sp1ap	9.04808	0.939619	0.936306	0.004804
mayap	1.5	0.939284	0.972574	0.00453
smah	85.25	0.933559	0.946546	0.004032
htpc	7.5	0.928958	0.971292	0.003967
spap	8.22826	0.939578	0.937073	0.003655
ss1	60.5	0.936098	1	0.003505
csp	-49	0.928809	1	0.003387
htcd	-50	0.929023	1	0.003184
eflaa	42.3274	0.932754	1	0.003076
sfnr	69.5	0.938205	0.936309	0.002783
sfra	1.03472	0.938192	0.971061	0.00269
wtdd	3.25	0.932551	0.936528	0.002401
sp2ap	7.2561	0.938924	0.943561	0.001878
jnap	12.6667	0.939463	0.938511	0.001405
fldd	0.25	0.93076	0.964912	0.000567
treat	NA	0.944762	0.945021	8.79E-05

Note:

1. PCT-left is the percentage of non-infestation examples that have feature value less than threshold.

2. PCT-right is the percentage of non-infestation examples that have feature value greater than or equal to theshold.

References

Bloedorn, E., Michalski, R. S. (1998). *Data-driven Constructive Induction: Methodology and Applications.*In Feature Extraction Construction and Selection A Data Mining Perspective, 51-68, Kluwer Academic Publishers.

Donoho, S., Rendell, L. (1998), *Feature Construction Using Fragmentary knowledge*. In Feature Extraction Construction and Selection A Data Mining Perspective, 273-288, Kluwer Academic Publishers.

Frawley, W. J., Piatetsky-Shapiro, G., Matheus C. J. (1991), *Knowledge Discovery in Databases: An Overview*, Knowledge Discovery in Databases, eds Shaprio and Frawley 1-27, AAAI Press / The MIT Press

Hu, Y. (1998), *Constructive Induction: Covering Attribute spectrum*. In Feature Extraction Construction and Selection A Data Mining Perspective, 257-272, Kluwer Academic Publishers.

Kohavi, R., John, G. H. (1998), *The Wrapper Approach*. In Feature Extraction Construction and Selection A Data Mining Perspective, 33-50, Kluwer Academic Publishers.

LeCun, Y., Boser, B., Denker, J.S., Henderson, B., Howard, R.E., Hubbard, W., Jackel, L.D. (1998), *Backpropagation Applied to Handwritten Zip Code Recognition* in Neural Compuatation Volume 1. Number 4., 541-551.

Liu, H., Motoda, H. (1998), Less is more, In Feature Extraction Construction and Selection A Data Mining Perspective, 3-12, Kluwer Academic Publishers.

Lockwood, J. A. (1997), *Rangelang Grasshopper Ecology*. In The bionomics of grasshoppers, katydids, and their kin, ed. Gangwere, S. K., Muralirangan, M. C., Muralirangan, M. Oxon, OX, UK ; New York : CAB International.

Mallet, Y., de Vel, O., Coomans, D. (1998*), Integrated Feature Extraction Using Adaptive Wavelets*. In Feature Extraction Construction and Selection A Data Mining Perspective, 175-190, Kluwer Academic Publishers.

Mitchell, T. M., (1997), Machine Learning, McGraw-Hill.

Nguifo, E. M., Njiwoua P. (1998), *Using Lattice-based Framework as a Tool for Feature Extraction.* In Feature Extraction Construction and Selection A Data Mining Perspective, 205-218, Kluwer Academic Publishers.

Perrin, P., Petry, F. (1998*), Lexical Contextual Relations for the Unsupervised Discovery of Texts Features.* In Feature Extraction Construction and Selection A Data Mining Perspective, 157-173, Kluwer Academic Publishers.

Unidata (1999), *UCAR Unidata Program Center*, http://www.unidata.ucar.edu.

Waibel, A., 1998, Modular Construction of Time-Delay Neural Networks for Speech Recognition in Neural Computation, Volume 1., Number 1., 39-46.

Wang, H., Bell, D., Murtagh, F. (1998). *Relevance Approach to Feature Subset Selection*. In Feature Extraction Construction and Selection A Data Mining Perspective, 85-99, Kluwer Academic Publishers.