Investigating Multi Instance Classifiers for improved virus classification in TEM images

Sujan Kishor Nath
Abstract

**Investigating Multi Instance Classifiers for improved virus classification in TEM images**

*Sujan Kishor Nath*

CBA together with the industrial partners Vironova AB (Stockholm) and Delong Instruments (Czech Republic) have a joint research project with the goal of developing a table-top TEM with incorporated software for automatic detection and identification of viruses. A method for segmenting potential virus particles in the images has been developed as has various measures of characteristic features, mainly based on texture, for distinguishing between different virus types. Different virus species generally have different sizes and shapes but their width (diameter if approximately spherical) is a rather conserved feature as is the protein structure on their surface (seen as texture patterns in the images).

In the project they currently focus on using different texture measures calculated on a disk centered within an object for classifying the virus species. Extracted feature measures calculated for one position for (at least) 100 objects of 15 different classes of viruses exist for use in this project. The aim of this thesis is to investigate if/how feature vectors calculated in multiple positions can be used to improve the classification. Since the viruses have very different shapes, from approximately spherical to highly pleomorphic (like boiled spaghetti), the number of possible positions for extracting feature vector will be different for different virus objects.

Another goal is to investigate how the distribution of measures calculated on small patches within the disk shaped feature area can be used in the classification, rather than combining them into one measure as is currently done.
Acknowledgements

First of all, I would like to express my gratitude to Ida-Maria Sintorn for providing me the opportunity to do this project as my thesis work. I would like to thank my supervisor Gustaf Kylberg for his invaluable suggestion and support during my thesis work.

I am thankful to the Center for Image Analysis for accepting my thesis proposal and my examiner Ida-Maria Sintorn for her invaluable suggestions and providing her invaluable time to read my report. Many many thanks to my friends who helped me directly or indirectly to successfully complete this project. Special thanks to Mesfin Mamuye, Krishna Paudel, Lu Ping and many more.
Contents

1 Introduction 9

2 Supervised Learning 11
   2.1 Classification ................................. 11
   2.2 Test and Training ............................. 12
   2.3 Cross Validation .............................. 13
   2.4 Evaluation .................................. 14

3 Multi-Instance Learning 15
   3.1 Introduction .................................. 15
   3.2 Multi-Instance Learning and Classifiers .......... 16
      3.2.1 Simple MI ............................... 16
      3.2.2 Axis-Parallel Rectangle .................... 16
      3.2.3 Diverse Density ......................... 17
      3.2.4 Citation k-NN ............................ 17
      3.2.5 Support Vector Machine(SVM) for MI ........ 18

4 Weka Multi-Instance Learning Tool 20
   4.1 Introduction .................................. 20
   4.2 Data Format .................................. 21
      4.2.1 Propositional Format: ..................... 21
      4.2.2 Multi-instance Format: .................... 21
   4.3 Filter: Preprocessing the Data .................. 22
   4.4 Classifier .................................... 23
      4.4.1 Building a Classifier .................... 23
   4.5 Evaluation ................................... 23

5 Texture Features 25
   5.1 Gabor Filter ................................ 25
   5.2 Zernike Moments ............................. 26

6 Experiment and Result 27
   6.1 Data set .................................... 27
      6.1.1 Virus Data Set ........................... 27
      6.1.2 Kyilberg Texture Dataset v. 1.0 .............. 28
   6.2 Experiments ................................ 29
      6.2.1 Evaluation on Virus Data (Gabor Filter Response) 29
      6.2.2 Evaluation on Virus Data (Zernike Moment) .... 30
      6.2.3 Evaluation on Texture Dataset ............... 31

7 Summary 32

Bibliography 33
# List of Figures

1.1 Image Data Sets ................................................. 10
2.1 Classifier Model *Overfitting* Problem. .......................... 13
2.2 Confusion Matrix .................................................. 14
3.1 Multiple Representative Patches .................................. 15
3.2 Polynomial Kernel .................................................. 19
5.1 Gabor Filters ....................................................... 25
5.2 Zernike Moments ..................................................... 26
6.1 Method 1: Generate MI bag ......................................... 27
6.2 Method 2: Generate MI bag ......................................... 28
Chapter 1

Introduction

Multi Instance (MI) classification is a variation of supervised classification for problems where training examples are ambiguous: each example object may have multiple candidate feature vectors (instances) that represent it. The study on MI learning is inspired by the drug activity prediction problem. Multiple-instance learning was first introduced in [DL97] and it proposed learning axis-parallel concepts to solve the MI learning problems. In supervised classification, every training example is assigned to a real-valued or discrete class label and represented by a single feature vector or instance. However, in MI classification an example object is represented by a collection of feature vectors (called a bag) that describes it. Also class labels are only assigned to the bags of instances instead of each instance. In the binary case a bag is assigned a positive label if at least one instance in that bag is positive, if all the instances of a bag are negative then the bag is assigned a negative label (or given a negative label). The objective of MI classification is to classify unknown bags based on training example bags.

Two datasets are used in supervised learning called the training and test dataset respectively. In MI learning the training set is a collection of labeled bags each composed of many instances without individual labels.

The main aim of this thesis is to evaluate the performance of different MI classifiers on two datasets shown in Figure 1.1 (Virus Texture Dataset 1.0 and Kylberg Texture Dataset 1.0 [Kyl13]) and compare the classification result with classic classifier result. For the Virus texture dataset, features to be used in the classification were generated using Regional Zernike Moments ([JFMM09]) and Gabor filter banks ([Kyl13]). The multiple filter kernels are used to generate different responses from each image. Each filter response contains a lot of information. Commonly the mean, standard deviation and properties like entropy are computed and used as feature values. However, doing so important information may be lost. In my thesis I evaluate two approaches to cope with this problem: In the first method each virus patch is considered a bag and each pixel of the filter response is an instance of that bag. In the second method multiple patches are extracted from each virus image. Here, every extracted patch is an instance of that particular virus image. Both methods are described in chapter 5.
Figure 1.1: Image Data Sets. (a): Texture Data Set (b): Virus Data Set. Source: http://www.cb.uu.se/gustaf/
Chapter 2

Supervised Learning

In machine learning, supervised learning is the problem of determining the category of an observation based on the training data set whose category memberships are known. Each observation will be described with a set of quantitative or qualitative properties, known as the features vector. These feature vectors may be categorical (e.g. A+, B-, AB for blood group), ordinal (e.g. High, Low, Medium for height), real-valued (e.g. stock price). Some Algorithms e.g. k-nearest neighbor (KNN) [Wik12], support vector machines (SVM)[ATH03] support only discrete data or real valued data. An example of supervised learning would be to identify the type of an unknown virus based on texture and shape measures or assigning health status (e.g. healthy, ill) of a patient based on observed characteristics of the patient (blood pressure, body temperature, presence or absence of certain symptoms etc.)

2.1 Classification

Classification is the machine learning technique of inferring a function from known training data. The training dataset is composed of a set of training observations. In supervised learning, each observation is a pair composed of an input feature vector and a desired output label. A supervised learning technique analyzes the training dataset and generates a model or inferred function which is called a classifier (if the output label is discrete) or a regression function (if the output value is continuous). The inferred function or classifier model should predict the actual output for any valid input observation. This requires the inferred function to generalize from example data to unknown example in a reasonable way.

To solve a supervised learning problem, we have to do the following steps.

- Determine the nature of the training dataset. At the beginning, one has to decide what kind of data is to be used as a training dataset. In the case of virus classification for instance the feature vector might be mean value of filter responses of Gabor filters (Figure 6.1).

- Collect a training set. The training data set should cover all possible examples such that the inferred function will be able to determine the labels of unknown data. Thus, a set of input examples is gathered also with corresponding outputs labels or values from human experts or from a system.

- Feature selection or feature extraction is the important task to represent the learned function. The accuracy of the inferred function strongly depends on how features are being chosen. An input object is represented by a feature vector which consists of a number of features that are properties of the object. More features could decrease the
accuracy rate of the classifier because of curse of dimensionality [YDY08] which can increase the noise and require a higher number of training samples. However, the feature vector should contain enough features to accurately predict the output.

- Determine the learning algorithm which will fit best for the dataset. For instance, support vector machines, KNN or a decision tree may be used as the classifier.

- Apply a learning algorithm on the training set. Some learning algorithms need to adjust certain parameters to optimize performance. Cross validation is used to adjust these parameters by optimizing performance on a subset of the training set.

- The final step is to evaluate the performance of the learning algorithm. After parameter adjustment, run the learning algorithm on a test set that is completely separated from the training set and measure the performance of the learning algorithm.

There are several supervised learning algorithms with different characteristics and learning capabilities. There is no learning algorithm that fits all kinds of supervised problems.

### 2.2 Test and Training

As mentioned in the previous section, supervised learning begins with evaluation of the training set and appropriate learning algorithm selection. An initial data set is used for this purpose. In supervised learning this dataset is represented by $n$ dimensional feature vectors. This dataset is called the training dataset.

The performance of the learning algorithm is natural to measure in terms of error rates of all classes and as an average error rate. The learning algorithm predicts the class of each sample; if prediction is same as the actual class that is counted as a success; if misclassified it is an error. The misclassification rate is just the ratio of errors made over a whole set of instances and it evaluates the overall performance of the learning algorithm.

Of course, what we are more interested in is the performance on new data, not the performance measured on old data. Training dataset is only used to build the classifier model. Hence the question is, the misclassification rate on training dataset likely to be a good indicator of the misclassification rate on new data? Indeed the answer is no because for new dataset misclassification rate might be different from misclassification rate on training dataset. So performance need to be measured on independent test dataset. If the classifier performance on the training dataset is much better than on the test dataset then over-fitting is likely the cause. Figure 2.1 shows an example of the overfitting problem.

To evaluate the performance of a classifier on new data, we need to evaluate its misclassification rate on a dataset that follows the same probability distribution as the training data and played no role in the development of the classifier. This independent dataset is called the test set.

It is important that training data should not include the test set in any way in the formation of the classifier. For instance, several learning algorithms follow two steps, a first step to generate a basic learning model and a second to optimize the control parameters involved in that learning model. Different data sets may be needed in both stages. Or you might train several different learning algorithms on the training data and then measure performance on a independent test dataset to see which one performs better.

In that case, we can use one more dataset called a validation set. The training data is used to build the classifiers using one or more learning algorithms. The validation set is used to optimize the control parameters of those learning algorithms or choose one learning algorithm. The test set is used to evaluate the misclassification rate of the final optimized classifier. Each
Figure 2.1: Blue points are indicating the training set (left) and test set (right) follows the same probability distribution. Two classifier models fit to training data; both fitted classifier models are plotted in orange and green with both training and test data set. The minimum square error (MSE) for the orange curve in training data set is less but is very high for test data set. The green curve does not overfit the training dataset much, as its MSE for training set is less than the orange curve model. Source: http://en.wikipedia.org/wiki/File:Traintest.svg

of the three set should be independent from each other. The validation sets must be separated from the training set to achieve better optimization or selection of algorithm, the test set must be separated from both to predict the actual misclassification rate.

2.3 Cross Validation

Sometimes problems arise when there is only a small amount of data available. In many cases, training data is classified manually and an independent test set is required to obtain the error rate. In a real scenario, there might be a limited number of instances for training and testing. There is a method called holdout procedure to separate certain amount of data for testing and use rest of the part for training (part of that would be used for validation, if required). Basically, it is common to separate one third of the instances for testing set and two third of the instances for the training set.

Of course the data set used for training and testing must contain the same amount of instances from each class. We have to ensure that random sampling is done in way that guarantees training and test sets consist of right proportion from each class. This method is known as stratified holdout. In general, this is a good way to ensure that all classes are not unevenly represented in training and test sets.

Sometime, a particular sample taken for holdout can cause bias in error estimation. To mitigate any bias we can repeat the training and test phase several times with different random samples. In each iteration, two-thirds of the data is chosen for training using stratified holdout and rest of the data is used for testing. Each iteration’s error rates are averaged and it gives an overall misclassification rate. To make different random samples from the given data set, there is a statistical method called cross-validation [Sch13]. You can decide the number of partitions or folds of the data in cross-validation. There are several types of cross-validation techniques used to evaluate the performance of the model such as K-fold, Leave-one-out and Repeated random sub-sampling etc. K-fold cross validation is commonly used in machine learning.

In k-fold validation, the original dataset is randomly divided into k parts in which the class
is represented in almost the same proportions as in the original dataset. Each part is held out one at a time as the validation data for testing, the remaining k-1 parts are used as training data. Then the misclassification rate is estimated on held out set. This procedure is repeated k times such that each fold is used exactly once as validation set. The error rates from the iterations can be averaged to produce an overall error rate. The advantage of cross validation is that each instance is part of validation set exactly once and all instances are used for both training and testing. In general, K is often set to 10 leading to 10-fold cross-validation.

2.4 Evaluation

In this thesis, classification results are evaluated using confusion matrices. A confusion matrix allows visualization of correct and incorrect predictions made by the classification algorithm compared to the actual class labels in the dataset. The confusion matrix is \( N \times N \), where \( N \) is the number of classes in the dataset. In the matrix each column represents the number of objects in a predicted class and each row represents the number of objects in an actual class. Figure 2.2, shows an example of a confusion matrix for two classes (Positive and Negative). Four positive samples were predicted as negative and six negative samples were predicted as positive. In total there were 38 positive and 40 negative samples in the test set.

<table>
<thead>
<tr>
<th>Actual class</th>
<th>Predicted class</th>
<th>Positive</th>
<th>Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Positive</td>
<td>34</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Negative</td>
<td>6</td>
<td>34</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.2: Confusion matrix for two classes
Chapter 3

Multi-Instance Learning

3.1 Introduction

In many classification problems objects are too rich and contain a lot of information, so it is not easy to represent the object by a single feature vector. A typical example is the problem of the virus image dataset. Some viruses like influenza are elongated and we can retrieve multiple patches from those and it can be represented by several feature vectors. Figure 3.1 is showing an elongated virus with multiple representative circular patches. We can get multiple filter responses for each patch after applying a set of Gabor or Zernike moment filters.

![Multi instance image patches](image)

Figure 3.1: Multi instance image patches, Each square box represents an instance of the virus patch.

In standard classification, an object is represented by a single feature vector. This feature vector consists of descriptive measures derived from the object, such as the mean values of the response images after applying a set of filters. As a result, much information might be lost from the object and the representation of the object might become inadequate. In this case we can extend the assumption of having a single feature vector per object. In Multi Instance Learning (MIL), each object is represented by a set (called a bag) of feature vectors (called “instances”). Class label (either positive or negative class) is only assign to the bag not the individual instances. The class label of a new bag is determined by the instances that belong to that bag.

MIL classification might help to develop a better prediction system. For example for a medical classification problem: An X-ray image of a patient is used to identify abnormalities in the lung. Multiple patches can be extracted from the x-ray image and it is hard to label all those patches individually. This problem could be solved by MIL, where individual patches are not label but the set (bag) of patches (instances) is labeled healthy or ill. If none of the instances in the bag are indicated as “ill”, then the patient is classified as healthy.

Since In MIL classification the samples are represented by a set of instances or feature vectors, the datasets tend to be larger and the learning process more difficult. Unfortunately the number of instances in MIL datasets is commonly limited. Since instances are not labeled,
only bags are classified, so the performance of a MIL classifier depends directly on the number of bags but not on instances. MIL datasets often have low-sample-size limitation. Classifier performance evaluation such as area under ROC curve has shown to be more suitable for low-sample-size dataset.

Furthermore, to be able to select the best classifier for a specific problem often depends on the number of samples in the dataset, e.g. number of samples in the training set. It is possible to train a more complex classifier if a larger training set is available. Reducing the number of samples in the training set may produce a different optimal classifier. It is therefore good to observe the performance of classifiers for different sizes of the training set.

3.2 Mult Instance Learning and Classifiers

Let’s assume that sample $i$ is represented by a bag $B_i$, and consist of $m_i$ unlabeled instances $x \in \mathbb{R}^p : B_i = \{x_{ij}, j = 1, \ldots, m_i\}$. [Feature vector length p]

Again assume that $Y_i \in \{w^+, w^-\}$ denotes the class label (positive or negative) that is assigned to each bag $B_i$ generally depends on the number of positive instances $X_{ij}$ in that bag. Some classifiers assign a positive label to a bag when it contains at least one positive instance. Sometimes bag contains lots of instances or the classification is noisy, then a certain number of instances are chosen from each bag. In the training phase the label of instances $y_{ij}$ is unknown. A classifier has to identify representative instances in each bag that would help to determine the bag class label. At last a trained classifier predicts the class label based on instances in the bag.

MIL function:

$$\hat{Y} = f(B) = f(\{x_j\}).$$  \hspace{1cm} (3.1)

A Selected number of classifiers of which some used in this thesis discussed in the next sections.

3.2.1 Simple MI

This classifier just transforms the MI dataset to a typical mono-instance dataset and use traditional learning classifiers such as support vector machines or K-NN to come up with learning model. There are several ways to transform MI data to mono-instance dataset such as arithmetic average of all instances in the bag, choosing the geometric center as a candidate instance for particular bag or using min-max combined features of bag[Wek13a].

3.2.2 Axis-Parallel Rectangle

Axis-Parallel Rectangle [DL97] is the first classifier within the multi-instance concept to solve MI problems. The aim of this method is to construct an axis-parallel hyper-rectangle (APR) in the feature space such that at least one instance from each positive bag falls inside the boundary. Meanwhile all instances from negative bags will be removed. Three optimization schemes are suggested in [DL97] to find such a decision boundary or APR: a “standard” scheme constructs the smallest boundary that covers all the instance from a positive bag; an “outside-in” scheme constructs the smallest rectangular decision boundary or APR such that all instances in the positive bags are included and then shrinks the boundary to remove false positives; an “inside-out” starts from a seed point and then grows an axis-parallel hyper-rectangle until it covers at least one instance per bag and does not cover any instances from a negative bag.
3.2.3 Diverse Density

Diverse Density (DD) is a general framework that was published in [MLP98] for solving MI learning problems. The objective of the Diverse Density method is to identify a core point in features space that is as close as possible to at least one instance of all positive bags and as far as possible from instances of all negative bags. The optimal core point has the highest diverse density which describes how many instances of positive bags that are close to the core point and how many instances of negative bags that are far away from that point. The diverse density approach optimizes the core point and radius (using a Gaussian Model) to get the highest diverse density. A new bag is classified as positive if any instances fall inside the diverse density area.

A probabilistic base Diverse Density is described below. Assume that a positive bag is denoted as $B_i^+$ and we use $B_{ij}^+$ to denote the jth instance of the bag. Assume further that each instance is represented by a feature vector and that the kth feature vector value is denoted $B_{ijk}^+$. Similarly we denote a negative bag as $B_i^-$ and the jth instance in that bag as $B_{ij}^-$. The core point is denoted as $t$ and for which the diverse density is the highest is defined as $DD(t) = p(t|B_i^+, \ldots, B_n^+, B_1^-, \ldots, B_m^-)$ over the feature space. Using Bayes theorem and considering uniform prior over core point, $DD(t)$ is equivalent to maximizing the following likelihood:

$$\arg\max_t p(B_1^+, \ldots, B_n^+, B_1^-, \ldots, B_m^-|t),$$

Assume further that all bags are conditionally independent for any core point t. This assumption decomposes the previous likelihood probability:

$$\arg\max_t \prod_i p(B_i^+|t) \prod_i p(B_i^-|t),$$

Now we can apply Bayes rule assuming uniform prior:

$$\arg\max_t \prod_i p(t|B_i^+) \prod_i p(|t|B_i^-),$$

which is basically the general form of diverse density.

3.2.4 Citation k-NN

Citation k-NN is a variant of the popular k Nearest Neighbor (k-NN) [Wik12] classifier. This is only possible if the distance between bags is possible to measure. In [Wan00], the distance between two bags is measured using the minimum Hausdorff distance. The distance between two bags is the shortest distance between any two instances from each bag.

$$\text{Dist}(A, B) = \text{Min}(\text{Dist}(a_i, b_j))$$

(3.2)

Where A and B are two bags and $a_i$ denotes an instance from bag A and $b_j$ denote instance from bag B. In K-NN generally bag label used to predict the unknown bag label. Sometimes in a MIL dataset, most of the labels of the K nearest neighbors of a new bag are not correctly labeled; because of this majority voting approach in K-NN a wrong prediction of bag label might result. The citation criterion is used to overcome this issue. In this approach not only the nearest neighbors (known as references) of unlabeled bags are considered, but also the number of bags that consider the unlabeled bag as their neighbor (known as citers). This is a more robust approach compared to K-NN, since both citers and references of an unlabeled bag are considered to predict the bag label.
3.2.5 Support Vector Machine (SVM) for MI

To modify the Support Vector Machine (SVM) for MI was proposed in [ATH03]. Two approaches were proposed, *mi-SVM* and *MI-SVM* for instance-level and bag-level classification respectively. In mi-SVM, instance levels \( y_i \) are considered as unobserved variables subject to constraints based on their bag labels \( Y_I \). The objective of *mi-SVM* is to maximize the instance margin jointly over unlabeled instances. Following is a kernelized discriminant function:

\[
\min_{\{y_i\}} \min_{w, b, \xi} \frac{1}{2} \|w\|^2 + C \sum_i \xi_i,
\]

where \( W \) is normal vector to hyperplane, \( b \) is the bias, \( C \) is a regularization parameter and \( \xi \) is slack variable. *mi-SVM*: where the second part of the constraint imposes a relationship between bag labels and instance labels. In comparison, the objective of MI-SVM is to maximize the bag margin. This kernelized discriminant function:

\[
\min_{w, b, \xi} \frac{1}{2} \|w\|^2 + C \sum_I \xi_I
\]

**MI-SVM:**

\[
\text{subject to } \forall I : Y_I \max_{i \in I} (\langle w, x_i \rangle + b) \geq 1 - \xi_I, \xi_I \geq 0.
\]

Note that, the margin of each instance is considered in *mi-SVM* and to maximize the margin it is possible to set the instance label variables subject to constraints of their bag labels. In comparison, in *MI-SVM* only one instance from each bag is considered to define the margin of the bag.

**Sequential Minimal Optimization (SMO)**

To make SVM training faster, SMO was proposed in [Pla98]. A very large scale of quadratic programming (QP) optimization problem needs to be solved to train a SVM classifier. SMO splits the QP problem into smallest possible QP sub-problems which are solved analytically. Therefore, time-consuming numerical QP optimization as an inner loop is not required to train a SVM. Less memory is required for SMO, because no extra matrix storage is needed which allows SMO to train SVM with large training dataset.

**Polynomial Kernel**

Polynomial kernel is a non-stationary kernel function used with SVM. Polynomial kernel is useful for a problem if training dataset is normalized. Polynomial kernel is defined as [wik13]

\[
K(x, y) = (x^T y + c)^d,
\]

where \( x \) and \( y \) are MI instance vectors in the input space, the constant term \( c \) and \( d \) is polynomial degree. Figure 3.2 illustrates the mapping of input vectors in a feature space.
Radial Basis Function

The (Gaussian) radial basis function (RBF) is a kernel function commonly used with SVM. The RBF is defined as [BSV04]

\[ K(x, x') = \exp \left( -\frac{\|x - x'\|^2}{2\sigma^2} \right), \quad (3.4) \]

where \( x \) and \( x' \) are two samples, \( \sigma \) is a adjustable parameter. An alternative definition could be

\[ K(x, x') = \exp \left( -\gamma(\|x - x'\|^2) \right), \quad (3.5) \]

where \( \gamma = \frac{1}{2\sigma^2} \).
Chapter 4

Weka Multi-Instance Learning Tool

4.1 Introduction

Weka stands for Waikato Environment for Knowledge Analysis is commonly used in machine learning completely written in Java. The University of Waikato is the founder of Weka and it is an open source project under the GNU General Public License. The Weka suite contains a collection of learning algorithms and data visualization tools for prediction and data analysis. Weka provides a friendly graphical user interface to use all functionalities together with java API. This project was initially developed for analyzing data from the agricultural domain and was written in C but the current version of Weka is fully Java-based and it is used in different areas such education, research etc. Some important features of Weka include:

- Open source project under the GNU General Public License.
- Platform independent, since it is written in completely in Java, so it is compatible in any modern operating system.
- A wide collection of machine learning algorithms and visualization tools.
- Easy to use because of its graphical user interfaces.
- Well documented Java API for programmers is provided.

Weka provides several machine learning techniques, in particular for data preprocessing, classification, regression, clustering, feature selection and visualization. Weka learning techniques generate predictive models on the assumption that the data-set is available in a specific format of flat file or in relational database where each sample is represented by a fixed length of attributes (nominal, ordinal or numerical). Weka supports access to different relational databases using JDBC connectivity. It has no support for multi-relational data analysis but there is a third party tool for transforming a group of linked relational tables into a single table that is compatible with Weka.

Weka has three environments Explorer, Experimenter and Knowledge Flow. The Weka Experimenter provides all machine learning functionality. The Weka Experimenter environment provides support to create, update, execute and analyze experiments in a more flexible way than is possible when analyzing the schemes individually. For instance, the user can generate an experiment that executes many schemes against multiple datasets, and then examine the results to choose the optimal scheme (statistically). The Knowledge Flow provides the same functionality as the Explorer but the graphical interface is a more data-flow inspired design. The user can drag and drop different learning components on a layout canvas and joint them together to create knowledge flow for preprocessing and predictive modeling.
4.2 Data Format

*Weka* data format is called *ARFF* (Attribute-Relation File Format). *ARFF* file is an ASCII text file that contains instances and an attribute list. ARFF files are divided into two sections. The first section, the header information, describes the name of the relation, attribute list, their types and class label. The second section, the data information, describes the instances. *Weka* has two data formats named as propositional format and multi-instance format, where propositional format is used for general classifier and multi-instance format is used for Multi-instance classifier. The Multi-instance format has the following header information:

- Bag-id: nominal attribute; each bag identifier
- Bag: relational attribute names and type
- Class: the class name of the bags.

The next two subsections illustrate the proposition format and MI format respectively for the virus dataset.

4.2.1 Propositional Format:

This *ARFF* consist of all the attributes, virus bag (which is the bag id), virus class label and data (contains the data of the instances).

```arff
@relation virus
@attribute virus_bag(1,2,3,4,5,6) // instance id
@attribute bag relational
@attribute f1 real // attribute name and type
@attribute f2 real
@attribute f3 real
@attribute f4 real
@attribute f5 real
@attribute f6 real
@end bag
@attribute class ("Adenovirus","Astrovirus","CCHF") // class name
@data
1,"8.11462,5.00690,2.24918,1.19259,1.14880,1.15052","Adenovirus"
2,"6.72820,4.00578,2.04798,1.28311,1.06585,1.01312","Adenovirus"
3,"6.47191,4.10594,2.09722,1.34959,1.17124,1.10618","Astrovirus"
4,"7.75440,4.76798,2.04335,1.05598,1.01962,1.10797","Astrovirus"
5,"8.01840,4.92295,2.13153,1.04862,1.02386,1.12150","CCHF"
6,"7.99419,4.95309,2.03102,0.97458,1.02072,1.12560","CCHF"
```

4.2.2 Multi-instance Format:

MI data is represented using relational attribute where each example is described using three attributes: bag-id, bag, class. The bag contains instances of each example, consisting of the attributes $f1$ to $f3$. Each instance in the bag is separated by \n and data is surrounded by the quotes.
4.3 Filter: Preprocessing the Data

The *Weka* data filters are concerned with class label that transform dataset by filtering attribute or instances, resampling the dataset, filtering examples and so on. Filtering helps to select attribute by filter out an attribute from the list and measure performance of the classifier. The *Weka* filters provide good support for data preprocessing, which is a key part of machine learning. The filter package is arranged into supervised and unsupervised filtering. Again each filter is subdivided into attribute and sample filtering. Most *Weka* filters implements the Java Interface OptionHandler, which helps you to set all properties by String array instead of setting them each manually. For instance, to remove the first attribute of a dataset you can use the following filter.

```java
weka.filters.supervised.attribute.Remove

with the following option

-R 1
```

Following example shows how to filter out the first attribute from the dataset. You can use the new dataset to measure classifier performance without the first attribute:

```java
import weka.filters.supervised.attribute.Remove;
import weka.filters.Filter;
import weka.core.Instances;
...
String [] filterOption = new String[2];
filterOption [0] = "-R";  // range /
filterOption [1] = "1";   // first attribute /
Remove removeFilter = new Remove();  // new instance of filter /
removeFilter.setOptions(filterOption);   // set options /
removeFilter.setInputFormat(dataset);   //
/* apply filter */
Instances newData = Filter.useFilter(dataset, removeFilter);
```
4.4 Classifier

Weka supports several Multi-Instance classifiers. The following classifiers are available in the Weka Java `weka.classifiers.mi` package [Wek13d].

- **CitationKNN**: Multi-instance variant of KNN classifier [Wik12].
- **MDD**: Diverse Density algorithm [MLP98].
- **MIBoost**: Geometric mean of posterior of instance is considered [FS96].
- **MIEMDD**: EM version MDD, It is a MI general framework implemented based on Dietterich’s Diverse Density (DD) algorithm [ZG01].
- **MISMO**: It is an implementation of John Platt’s sequential minimal optimization algorithm for SVM[Pla98] [KSBM01].
- **MISVM**: It is an implementation of Stuart Andrew’s mi_SVM[ATH03].

4.4.1 Building a Classifier

You can build a classifier in two different ways:

**Batch**

Training a classifier on a dataset in Weka is very simple, for instance we can train the CitationKNN algorithm on a given multi-instance dataset. The training of a classifier is done by the `buildClassifier(instances)` method.

```java
import weka.classifiers.mi.CitationKNN;

String[] options = weka.core.Utils.splitOptions("-R 4 -C 2 -H 1");

CitationKNN classifier = new CitationKNN(); /* new instance of classifier */
classifier.setOptions(options); /* set the options */
classifier.buildClassifier(dataset); /* build classifier */
```

**Incremental**

MI classifiers which implement the java interface `weka.classifiers.UpdateableClassifier` can be trained incrementally. Since data will be loaded incrementally into memory, this is good for bulk amount of data analysis. This procedure of incremental training of a classifier is quite easy:

- Call `buildClassifier(Instances)` method of classifier with dataset structure.
- Call `updateClassifier(Instance)` with new instances `weka.classifiers.bayes.NaiveBayesUpdateable` to incrementally update the classifier.

4.5 Evaluation

Cross validation is a very important technique if you don’t have independent test sets to evaluate the classifier on. So you can use 10 folded cross validation which is more commonly used in machine learning. Weka provides class for cross validation called Evaluation. Here is an example of cross validation.
import java.util.Random;
import weka.classifiers.Evaluation;

Evaluation evaluation = new Evaluation(trainingDataSet);
evaluation.crossValidateModel(classifier, 10, new Random(1));

If you have an independent test set then evaluate the classifier on test set. Here is an example,
CitationKNN classifier is trained with training set and then evaluated on the test set.

Instances trainDataSet = ...
Instances testDataSet = ...
/* train classifier */
CitationKNN classifier = new CitationKNN(); /* new instance of classifier */
classifier.buildClassifier(trainDataSet);
/* evaluate CitationKNN and print some statistics */
Evaluation evaluation = new Evaluation(trainDataSet);
evaluation.evaluateModel(classifier, testDataSet);
System.out.println(evaluation.toSummaryString(" ", false));
Chapter 5

Texture Features

In this thesis, two filter methods are used to generate feature vectors. These two methods are Gabor filter banks and regional Zernike moments and are described in the next section.

5.1 Gabor Filter

Gabor filter is used in image analysis after it generalized to 2d by Granlund [Gra78]. In the spatial domain, 2D Gabor filter \( \psi \), is defined as is in [BF07] :

\[
\psi(x, y, F, \theta, \gamma, \eta) = \frac{F^2}{\pi \gamma \eta} \exp \left(-F^2 \left(\frac{x'}{\gamma}\right)^2 + \left(\frac{y'}{\eta}\right)^2\right) \cdot \exp \left(i2\pi Fx'\right),
\]

(5.1)

\( x' = x \cos \theta + y \sin \theta, \)

\( y' = -x \sin \theta + y \cos \theta. \)

The angle between the direction of the wave is denoted by \( \theta \) and \( F \) is the frequency of the wave and the x-axis respectively. The Gaussian envelope is described by the standard deviation perpendicular to the wave, \( \eta \) and standard deviation parallel to the wave \( \gamma \). A Gabor filter bank is a collection of Gabor filters with different scales, frequencies and orientations, shown in the following Figure 5.1.

![Figure 5.1: An example of Gabor filter bank, Source:http://www.cb.uu.se/~gustaf/](http://www.cb.uu.se/~gustaf/)
5.2 Zernike Moments

Zernike moments are very useful method to generate rotation-invariant features which are basically magnitudes of the Zernike moments of the image. Compare to classical moments, Zernike moments are mappings of an image patch onto a set of complex orthogonal basis functions, which magnitudes can be used as rotation-invariant useful features [JFMM09]. Since Zernike basis functions are orthogonal to each other, Zernike moments can generate features of the image with no overlapping or redundancy of the data between the regional Zernike moments [SK12]. In this thesis regional Zernike moments are used to generate features from virus and texture dataset using different order of moments. Zernike moments with different polynomial orders that is shown in the Figure 5.2.

![Zernike Moments Filter Bank](http://www.cb.uu.se/~gustaf/)

Figure 5.2: Filter bank of zernike moments with different orders. Source: http://www.cb.uu.se/~gustaf/
Chapter 6

Experiment and Result

6.1 Data set

The Virus Texture Dataset 1.0 and the Kylberg Texture Dataset 1.0 have been used in this thesis to evaluate the performance of different Multi-Instance classifiers and compare the results with single instance classifier results. Both datasets are publicly available at [Kyl13].

6.1.1 Virus Data Set

The images in the virus dataset have been acquired in a transmission electron microscop (TEM). There are 13 virus types in this dataset. In this thesis, virus texture patches have been extracted from the images in two ways. In the first method, virus patches were extracted automatically using the process described in [KUH+11]. In the second method, patches have been extracted using manually marked coordinates of the center axis of each object. Each patch has been extracted in such a way that they do not overlap each other. Figure 6.2 shows an example of the manual patch extraction. Virus dataset has the following properties:

- 13 virus classes: Adenovirus, Astrovirus, CCHF, Cowpox, Ebola, Influenza, Lassa, Marburg (Musoke), Norovirus, Papilloma, Rift Valley, Rotavirus, WestNile (Dengue and Orf virus class is not included in the dataset), see Figure 1.1b
- Each class has 100 unique patches.
- Each patch size is 41x41 pixels.

---

Figure 6.1: Method 1: Generate Pixel base MI bag
Figure 6.2: Method 2: Generate MI bag, (a): Influenza Virus. (b): Each sample patch is instance of the bag. (c): Influenza Skeleton.

6.1.2 Kylberg Texture Dataset v. 1.0

The dataset contains images of different textured surfaces includes fabrics, stone, rice grains, sesame seeds and lentils. This dataset has the following features:

- Total number of texture classes are 28, see Figure 1.1a
- Each class has 160 unique patches.
- Size of each texture of patch: 576x576 pixels.
6.2 Experiments

In this thesis, different Weka MI classifiers performance are evaluated on two datasets. In this section I have evaluated the Weka MISMO [Pla98] support vector machine (SVM) classifier on two dataset using both MIRBFKernel[Wek13b] and MIPolyKernel[Wek13c] kernel, since this classifier result is better than other classifier results. The feature vector is generated using two methods. In the first method the response at each pixel position is considered as an instance of the bag and all response values of that position is consider as the feature vector. This is shown in figure 6.1. In the second method, each sample patch of the virus image is considered as instance of bag and the virus image is defined as a Bag, as shown in figure 6.2. Each classifier is trained by 10-fold stratified cross-validation and cross validation training set size is varied from 40% to 100% of the total training set.

6.2.1 Evaluation on Virus Data (Gabor Filter Response)

Table 6.1 shows the result of MI-SMO classifier on the virus dataset. In this experiment three virus classes (Adenovirus,Astrovirus,CCHF) are used in the dataset. Using the first Method (figure 6.1), Gabor filter responses are used to generate instances. Each response pixel is considered as an instance of the bag. Each pixel position’s response values across the Gabor filter responses are considered as feature vector of an instance. This result suggests that more training data might improve the classification rate. Initially the experiment is run with 180 training bags and classification rate is 43%, but after increasing the training bags to 420 classification rate increase to 95%. Training dataset should cover all possible samples so that classifier can determine more unknown class label of test samples. Since all response pixels are used as instances, the classifier training takes a long time and more instances can mislead the classifier since most of them are false examples.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Classifier Option</th>
<th>TrainBag</th>
<th>TestBag</th>
<th>CorrectBag</th>
</tr>
</thead>
<tbody>
<tr>
<td>MISO SMO</td>
<td>MIPolyKernel-C250007-E1.0</td>
<td>180</td>
<td>90</td>
<td>40</td>
</tr>
<tr>
<td>MISO SMO</td>
<td>MIPolyKernel-C250007-E3.0</td>
<td>180</td>
<td>90</td>
<td>48</td>
</tr>
<tr>
<td>MISO SMO</td>
<td>MIPolyKernel-C250007-E5.0</td>
<td>180</td>
<td>90</td>
<td>45</td>
</tr>
<tr>
<td>MISO SMO</td>
<td>MIRBFKernel-C250007-G0.01</td>
<td>180</td>
<td>90</td>
<td>47</td>
</tr>
<tr>
<td>MISO SMO</td>
<td>MIRBFKernel-C250007-G0.25</td>
<td>180</td>
<td>90</td>
<td>45</td>
</tr>
<tr>
<td>MISO SMO</td>
<td>MIPolyKernel-C250007-E1.0</td>
<td>270</td>
<td>90</td>
<td>47</td>
</tr>
<tr>
<td>MISO SMO</td>
<td>MIPolyKernel-C250007-E3.0</td>
<td>270</td>
<td>90</td>
<td>45</td>
</tr>
<tr>
<td>MISO SMO</td>
<td>MIPolyKernel-C250007-E5.0</td>
<td>270</td>
<td>90</td>
<td>43</td>
</tr>
<tr>
<td>MISO SMO</td>
<td>MIRBFKernel-C250007-G0.01</td>
<td>270</td>
<td>90</td>
<td>49</td>
</tr>
<tr>
<td>MISO SMO</td>
<td>MIRBFKernel-C250007-G0.25</td>
<td>270</td>
<td>90</td>
<td>42</td>
</tr>
<tr>
<td>MISO SMO</td>
<td>MIPolyKernel-C250007-E1.0</td>
<td>360</td>
<td>90</td>
<td>75</td>
</tr>
<tr>
<td>MISO SMO</td>
<td>MIPolyKernel-C250007-E3.0</td>
<td>360</td>
<td>90</td>
<td>73</td>
</tr>
<tr>
<td>MISO SMO</td>
<td>MIPolyKernel-C250007-E5.0</td>
<td>360</td>
<td>90</td>
<td>70</td>
</tr>
<tr>
<td>MISO SMO</td>
<td>MIRBFKernel-C250007-G0.01</td>
<td>360</td>
<td>90</td>
<td>72</td>
</tr>
<tr>
<td>MISO SMO</td>
<td>MIRBFKernel-C250007-G0.25</td>
<td>360</td>
<td>90</td>
<td>67</td>
</tr>
<tr>
<td>MISO SMO</td>
<td>MIPolyKernel-C250007-E1.0</td>
<td>420</td>
<td>90</td>
<td>78</td>
</tr>
<tr>
<td>MISO SMO</td>
<td>MIPolyKernel-C250007-E3.0</td>
<td>420</td>
<td>90</td>
<td>84</td>
</tr>
<tr>
<td>MISO SMO</td>
<td>MIPolyKernel-C250007-E5.0</td>
<td>420</td>
<td>90</td>
<td>81</td>
</tr>
<tr>
<td>MISO SMO</td>
<td>MIRBFKernel-C250007-G0.01</td>
<td>420</td>
<td>90</td>
<td>86</td>
</tr>
<tr>
<td>MISO SMO</td>
<td>MIRBFKernel-C250007-G0.25</td>
<td>420</td>
<td>90</td>
<td>80</td>
</tr>
</tbody>
</table>

Table 6.1: Result of MI-SMO classifier on the virus dataset using First Method (figure 6.1)

To reduce the vast number of instances per bag, instances are chosen at random. Table 6.2 shows the result of the experiment. However, it shows that random instance selection does not
improve the classifier result. The reason is probably that sometimes in the random selection process representative instances are discarded.

### 6.2.2 Evaluation on Virus Data (Zernike Moment)

In this thesis I got better classification result using \textit{regional Zernike moment} by the second method (figure 6.2) instead of Gabor filter. In this experiment thirteen virus classes are used in the dataset. Table 6.3 shows the best result using polynomial Kernel. In this case each patch of an object is considered as an instance and the object is represented as a bag of instances. It gives better result than random instance selection.

<table>
<thead>
<tr>
<th>No</th>
<th>Classifier Name</th>
<th>Options</th>
<th>Training Bag</th>
<th>TestBag</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MIPolyKernel</td>
<td>-E 1.0</td>
<td>58.33% (210)</td>
<td>56.49% (205)</td>
</tr>
<tr>
<td>2</td>
<td>MIPolyKernel</td>
<td>-E 3.0</td>
<td>64.16% (221)</td>
<td>66.18% (235)</td>
</tr>
<tr>
<td>3</td>
<td>MIPolyKernel</td>
<td>-E 5.0</td>
<td>62.22% (224)</td>
<td>64.16% (231)</td>
</tr>
<tr>
<td>4</td>
<td>MIRBFKernel</td>
<td>-G 0.01</td>
<td>35.00% (126)</td>
<td>34.72% (125)</td>
</tr>
<tr>
<td>5</td>
<td>MIRBFKernel</td>
<td>-G 0.09</td>
<td>56.83% (183)</td>
<td>52.77% (136)</td>
</tr>
<tr>
<td>6</td>
<td>MIRBFKernel</td>
<td>-G 0.2</td>
<td>58.61% (221)</td>
<td>57.77% (220)</td>
</tr>
<tr>
<td>7</td>
<td>MIRBFKernel</td>
<td>-G 0.3</td>
<td>59.10% (213)</td>
<td>59.72% (210)</td>
</tr>
<tr>
<td>8</td>
<td>MIRBFKernel</td>
<td>-G 0.4</td>
<td>58.62% (221)</td>
<td>61.38% (222)</td>
</tr>
<tr>
<td>9</td>
<td>MIRBFKernel</td>
<td>-G 0.5</td>
<td>61.38% (221)</td>
<td>62.22% (224)</td>
</tr>
<tr>
<td>10</td>
<td>MIRBFKernel</td>
<td>-G 0.6</td>
<td>60.27% (217)</td>
<td>63.61% (229)</td>
</tr>
<tr>
<td>11</td>
<td>MIRBFKernel</td>
<td>-G 0.7</td>
<td>66.83% (219)</td>
<td>64.44% (212)</td>
</tr>
</tbody>
</table>

Table 6.2: Classification result of randomly selected instances.

<table>
<thead>
<tr>
<th>No</th>
<th>Classifier Kernel</th>
<th>Options</th>
<th>Training Data:1800 bag</th>
<th>Training Data:1800 bag</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Influenza</td>
<td>MI</td>
</tr>
<tr>
<td>1</td>
<td>MIPolyKernel</td>
<td>-E 1.0</td>
<td>75%</td>
<td>56%</td>
</tr>
<tr>
<td>2</td>
<td>MIPolyKernel</td>
<td>-E 3.0</td>
<td>76%</td>
<td>52%</td>
</tr>
<tr>
<td>3</td>
<td>MIPolyKernel</td>
<td>-E 5.0</td>
<td>76%</td>
<td>54%</td>
</tr>
<tr>
<td>4</td>
<td>MIRBFKernel</td>
<td>-G 0.2</td>
<td>76%</td>
<td>54%</td>
</tr>
<tr>
<td>5</td>
<td>MIRBFKernel</td>
<td>-G 0.3</td>
<td>76%</td>
<td>48%</td>
</tr>
<tr>
<td>6</td>
<td>MIRBFKernel</td>
<td>-G 0.4</td>
<td>76%</td>
<td>50%</td>
</tr>
<tr>
<td>7</td>
<td>MIRBFKernel</td>
<td>-G 0.5</td>
<td>80%</td>
<td>50%</td>
</tr>
<tr>
<td>8</td>
<td>MIRBFKernel</td>
<td>-G 0.6</td>
<td>80%</td>
<td>52%</td>
</tr>
<tr>
<td>9</td>
<td>MIRBFKernel</td>
<td>-G 0.7</td>
<td>80%</td>
<td>52%</td>
</tr>
</tbody>
</table>

Table 6.4: Compare result of MI and single instance classification.

Multiple patches are extracted from the viruses which are elongated. A Single patch is extracted from circular shaped virus. In the next experiment only elongated viruses are used
since those have only multiple instances and created two datasets with multi instances and single instance from each patch using the second method (figure 6.2). Single instance dataset is created by randomly select an instance from each patch. This experiment result is shown in table 6.4. This experiment suggests that MI classification gives better result over single instance classification.

6.2.3 Evaluation on Texture Dataset

Only few experiments have been conducted on texture image dataset, since virus image dataset is the main concern of this thesis. Using the first Method (figure 6.1), Gabor filter responses are used to generate instances. Each pixel position’s response values across the Gabor filter responses are considered as feature vector of an instance. In this experiment, only 12 texture classes are used in the dataset. In Table 6.5 shows the classifier results of the different SVM kernels for varying the number of randomly chosen instances. This experiment suggests that increasing number of instances and training bag might increase the classifier performance. For better prediction classifier should be trained with sufficient amount of example training data.

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Classifier Option</th>
<th>InstanceSelectionMethod</th>
<th>TrainBag</th>
<th>TestBag</th>
<th>CorrectBag</th>
</tr>
</thead>
<tbody>
<tr>
<td>MISO</td>
<td>MIPolyKernel-C250007-E.1.0&quot;</td>
<td>Rand-40</td>
<td>840</td>
<td>220</td>
<td>183</td>
</tr>
<tr>
<td>MISO</td>
<td>MIPolyKernel-C250007-E.3.0&quot;</td>
<td>Rand-40</td>
<td>840</td>
<td>227</td>
<td></td>
</tr>
<tr>
<td>MISO</td>
<td>MIPolyKernel-C250007-E.5.0&quot;</td>
<td>Rand-40</td>
<td>840</td>
<td>207</td>
<td></td>
</tr>
<tr>
<td>MISO</td>
<td>MIRBF Kernel-C250007-G.0.01&quot;</td>
<td>Rand-40</td>
<td>840</td>
<td>210</td>
<td></td>
</tr>
<tr>
<td>MISO</td>
<td>MIRBF Kernel-C250007-G.0.09&quot;</td>
<td>Rand-40</td>
<td>840</td>
<td>220</td>
<td></td>
</tr>
<tr>
<td>MISO</td>
<td>MIPolyKernel-C250007-E.1.0&quot;</td>
<td>Rand-60</td>
<td>840</td>
<td>216</td>
<td></td>
</tr>
<tr>
<td>MISO</td>
<td>MIPolyKernel-C250007-E.3.0&quot;</td>
<td>Rand-60</td>
<td>840</td>
<td>231</td>
<td></td>
</tr>
<tr>
<td>MISO</td>
<td>MIPolyKernel-C250007-E.5.0&quot;</td>
<td>Rand-60</td>
<td>840</td>
<td>207</td>
<td></td>
</tr>
<tr>
<td>MISO</td>
<td>MIRBF Kernel-C250007-G.0.01&quot;</td>
<td>Rand-60</td>
<td>840</td>
<td>220</td>
<td></td>
</tr>
<tr>
<td>MISO</td>
<td>MIRBF Kernel-C250007-G.0.09&quot;</td>
<td>Rand-60</td>
<td>840</td>
<td>220</td>
<td></td>
</tr>
<tr>
<td>MISO</td>
<td>MIPolyKernel-C250007-E.1.0&quot;</td>
<td>Rand-80</td>
<td>840</td>
<td>231</td>
<td></td>
</tr>
<tr>
<td>MISO</td>
<td>MIPolyKernel-C250007-E.3.0&quot;</td>
<td>Rand-80</td>
<td>840</td>
<td>216</td>
<td></td>
</tr>
<tr>
<td>MISO</td>
<td>MIPolyKernel-C250007-E.5.0&quot;</td>
<td>Rand-80</td>
<td>840</td>
<td>213</td>
<td></td>
</tr>
<tr>
<td>MISO</td>
<td>MIRBF Kernel-C250007-G.0.01&quot;</td>
<td>Rand-80</td>
<td>840</td>
<td>220</td>
<td></td>
</tr>
<tr>
<td>MISO</td>
<td>MIRBF Kernel-C250007-G.0.09&quot;</td>
<td>Rand-80</td>
<td>840</td>
<td>220</td>
<td></td>
</tr>
</tbody>
</table>

Table 6.5: Classifier results for different SVM kernels on texture dataset.
Chapter 7

Summary

In this thesis, I have shown (Table: 6.4) that MI learning can be useful to classify elongated virus object using the second method (figure 6.2). In a classification problem it is very interesting to see how the classifier performs for varying sizes of training data. The experiment reports in this thesis show that the classifier performance rate is increased by increasing the size of the training set. However, the datasets that are used in this thesis have limited number of instances and very few virus classes are elongated to extract multiple patches. So this is a problem to optimize MI the classifier since most of the virus classes have only a single instance. The virus dataset that is extracted automatically has some issues about texture quality, pattern and false samples. For example Astrovirus and Influenza patches have rather similar texture. In that case, miss classification rate between similar texture classes is very high.

The results also show that there is not so much improvement after increasing the number of instances per bag using pixel based feature vector where each pixel is considered as an instance. The best performances are archived with fewer instances as shown in table 6.3 using the second method (figure 6.2) where each patch of an object is considered as an instance. When more instances are present in the bag it is harder to find out representative instances of the positive bag. The experiments in this thesis suggest that to use an MI classifier is preferable instead of a standard classifier if it is possible to extract multiple patches from the given objects.
Bibliography


In Proc. 17th International Conf. on Machine Learning, pages 1119–1125. Morgan 


[YDY08] Wen-Hui Yang, Dao-Qing Dai, and Hong Yan. Feature extraction and uncorrelated 
discriminant analysis for high-dimensional data. Knowledge and Data Engineering, 

[ZG01] Qi Zhang and Sally A. Goldman. Em-dd: An improved multiple-instance learning 
technique. In Advances in Neural Information Processing Systems 14, pages 1073– 