Summer 2013

Clubfoot Image Classification

Amanda Marie De Hoedt
University of Iowa

CLUBFOOT IMAGE CLASSIFICATION

by

Amanda Marie De Hoedt

A thesis submitted in partial fulfillment
of the requirements for the
Master of Science degree in Biomedical Engineering
in the Graduate College at
The University of Iowa

August 2013

Thesis Supervisor: Professor Thomas Casavant

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

MASTER'S THESIS

This is to certify that the Master's thesis of

Amanda Marie De Hoedt

has been approved by the Examining Committee
for the thesis requirement for the Master of Science
degree in Biomedical Engineering at the August 2013 graduation.

Thesis Committee:

Thomas Casavant, Thesis Supervisor

Terry Braun

Jose Morcuende

Todd Scheetz

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF EQUATIONS

CHAPTER 1: INTRODUCTION

Clubfoot is a congenital foot disorder that, left untreated, can limit a person's mobility by making it difficult and painful to walk [1]. Although inexpensive and reliable treatment exists, clubfoot often goes untreated in the developing world, where 80% of cases occur [2]. Untreated clubfoot can have a lasting impact on an individual's life; lack of mobility limits job prospects and clubfoot can carry a negative social stigma.

Many nonprofit and non-governmental organizations are partnering with hospitals and clinics in the developing world to provide treatment for patients with clubfoot, and to train medical personnel in the use of the Ponseti Method, the non-surgical serial casting method widely used as treatment for clubfoot [3].

Many, if not most, of the clubfoot hospitals and clinics working with these organizations function with limited infrastructure that is often taken for granted in developed nations. Clinics and hospitals may have limited or no access to high-speed internet, they may have unreliable power sources, and there may be few medical personnel [4; 5]. Civil infrastructure such as road quality or access to bridges may also be prohibitive for patients, who must travel (sometimes for many hours) to a hospital or clinic each week for treatment.

As a component of these partnerships, clinics and hospitals are collecting patient information with a web-based application that also has offline capabilities [6]. Some of this patient information, such as photographs, requires expert quality assessment. Such assessment may occur at a later date by a staff member in the hospital, or it may occur in a completely different location through the web interface.

Photographs capture the state of a patient at a specific point in time. If a photograph is not taken correctly, and as a result, has no clinical utility, the photograph cannot be recreated because that moment in time has passed.

These observations have motivated the desire to perform real-time classification of clubfoot images as they are being captured in a possibly remote and challenging environment. In the short term, successful classification could provide immediate feedback to those taking patient photos, helping to ensure that the image is of good quality and the foot is oriented correctly at the time of image capture. In the long term, this classification could be the basis for automated image analysis that could reduce the workload of a busy staff, and enable broader provision of treatment.

After two years of work on this classification problem, a greatly enhanced understanding of the challenges associated with clubfoot image classification has been acquired. Furthermore, modest success in orientation classification of this highly variable data set has been achieved. The methodology and results for this classification are outlined in the following chapters.

CHAPTER 2: BACKGROUND

This chapter provides basic definitions of terms and concepts used throughout the thesis. It also serves as a brief introduction to a number of algorithms and tools. These tools were used in the software development and data analyses outlined in chapter 3.

2.1 Clubfoot

As clubfoot is the subject of this research, it is important to understand what clubfoot is, how it is treated, the barriers to treatment, and the work that has been done to eliminate these barriers.

2.1.1 Definitions and History of Clubfoot

Clubfoot is a congenital foot disorder that affects approximately 1 in every 1,000 children worldwide [7]. Individuals with clubfoot experience bone and soft tissue deformation in at least one foot, which inhibits their ability walk [1].

Figure 1: Clubfoot is a congenital foot disorder that affects approximately 1 in every 1000 children worldwide. Individuals with clubfoot experience bone and soft tissue deformation in at least one foot, which inhibits their ability walk.

If left untreated, clubfoot can prohibit individuals from being productive members of society by limiting their mobility and causing social stigma [2]. This is especially pronounced in the developing world, where 80% of clubfoot occurs, and access to treatment is limited.

Although little is known about the causes of clubfoot, an inexpensive and effective treatment is available. The Ponseti Method, developed by Dr. Ignacio Ponseti at the University of Iowa Hospitals and Clinics, is a serial casting treatment in which the foot is manipulated into the correct position over a period of several weeks [8]. It is analogous to the process of straightening teeth in the field of orthodontics. Each week, the foot is manipulated and held in place with a cast. The casting stage of treatment takes about five to eight weeks. Following manipulation, the child wears a brace for several years so the feet do not relapse.

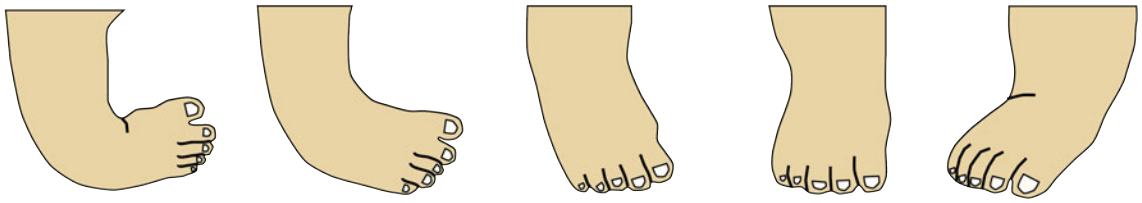

Figure 2: Clubfoot can be treated with the Ponseti Method. This method uses serial casting over a period of several weeks to gradually correct the foot.

2.1.2 International Clubfoot Registry

Many organizations and groups exist to reduce the prevalence of clubfoot by increasing the availability of treatment and by training medical personnel in the Ponseti Method. In collaboration with miraclefeet and the Ponseti International Association, the International Clubfoot Registry was developed [6; 9; 10]. This online tool collects patient

information and supports clinical, training, and research needs. In response to the limited infrastructure resources available in the developing world, an offline application was further developed. Now, with the growing prevalence of mobile devices in all parts of the world, the development of a mobile application has been proposed [11].

2.1.3 Clubfoot Image Utility

One of the important functionalities of the International Clubfoot Registry is the ability for clinics to upload photographs of their patients' feet before, during, and after treatment. Photographs are important because they can be used to make a diagnosis, monitor the progression of a patient's treatment, and evaluate the ability of medical personnel to successfully treat the disorder.

2.2 Tools and Techniques

In an effort to classify clubfoot photos, a number of tools and techniques in the fields of image processing, machine vision, and machine learning were used. First, characteristic elements, or image features, had to be extracted from the digital images. Then the computer "learned" relationships between these features and properties of the image. As a result, features could be used to infer information about novel images. This section will present a broad overview of the tools and techniques that were used for feature extraction and machine learning as part of this research.

2.2.1 Image Feature Extraction

Digital images are represented as an array of numbers that correspond to color and intensity. There are many methods for manipulating these numerical arrays to find points of interest in an image. MATLAB is a computer application that provides a powerful set of array-based tools that facilitate, or even implement, these methods [12]. VLFeat is an open-source library that integrates with MATLAB and implements many common machine vision

methods [13]. This section will present an overview of feature extraction methods relevant to this thesis.

2.2.1.1 Scale Invariant Feature Transform

Scale Invariant Feature Transform (SIFT) descriptors were described by David Lowe in 1999 and the methodology for extracting SIFT descriptors was patented in 2004 by The University of British Columbia [14; 15]. Descriptors are found by convolving an image with Gaussian filters at different image scales. At each scale, the differences between the Gaussian images are calculated, and then key points are detected.

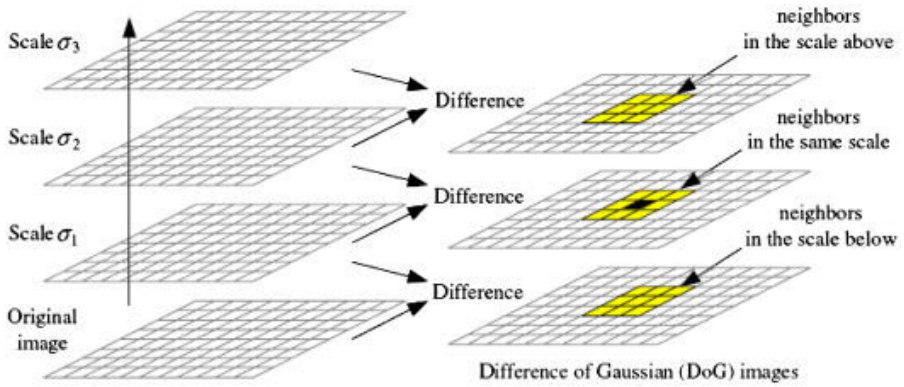

Figure 3: Scale Invariant Feature Transform (SIFT) descriptors are found by convolving an image with Gaussian filters at different image scales. At each scale, the differences between the Gaussian images are calculated. Key points are detected by finding local maxima and minima. SIFT features are scale and rotation invariant.

Key points are local maxima and minima found by comparing each pixel's value to the value of each of its neighbors within an image scale. Key points are filtered to remove those with low contrast and those found along edges, while keeping key points located at corners. Each key point is assigned a gradient magnitude and orientation.

There are many publicly-available implementations of Lowe's SIFT, for example, VLFeat's MATLAB-based implementation, which was used in this analysis. In the VLFeat

implementation, SIFT features are stored and referenced using a different format than that used in Lowe's implementation, but the resulting key points and descriptors are nearly equivalent.

SIFT features were chosen for this analysis due to their robustness. The clubfoot data set has a high degree of variability; photographs were taken with different cameras, under different lighting conditions, at different angles, with young, mobile children. SIFT descriptors' invariance to image scale and rotation, and robust performance with changes to illumination and noise made it an excellent candidate.

2.2.1.2 K-Means Clustering

K-means clustering is an algorithm used to group data points into K clusters based on similarity [17]. The algorithm was introduced by Stuart Lloyd and has four basic steps.

1) K initial "centroids" are generated within the domain of the data.
2) Data points are mapped to the nearest centroid, creating K clusters of data points.
3) The centroid of each cluster is calculated, and the centroid values are updated accordingly.
4) Steps 2 and 3 are repeated until convergence.

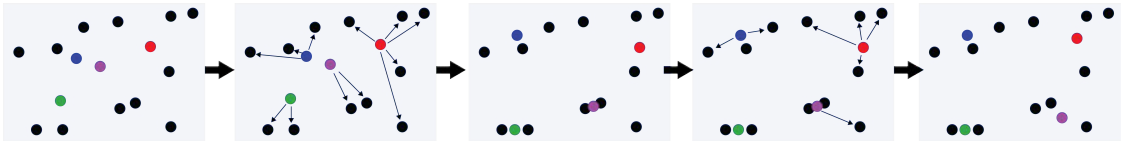

Figure 4: K-means clustering is an algorithm used to group data points into k clusters based on similarity. Centroids are represented as colored circles and data points are represented as dark circles.

K-means clustering is used to group clubfoot image features; VLFeat implements Elkan k-means clustering for faster performance compared to Euclidean k-means clustering [18; 19].

2.2.1.3 Bag-of-Words

Bag-of-words (BOW) is a technique that was originally developed for analysis of textual data, but it can also be applied to visual data by treating descriptors as words [20]. Using this technique, image feature vectors are expressed by a histogram of the occurrences of representative descriptors within the image [21].

There are three main steps for a bag-of-words implementation. First, descriptors are obtained from the entire data set, or a representative subset of the data. From this set of descriptors, a representative sample is chosen to create an unordered bag-of-words. This can be achieved using an algorithm such as k-means clustering. Finally, all of the descriptors in an image are found and mapped to the bag-of-words features using a distance metric. This results in a histogram showing the prevalence of each bag-of-words feature within the sample space [22].

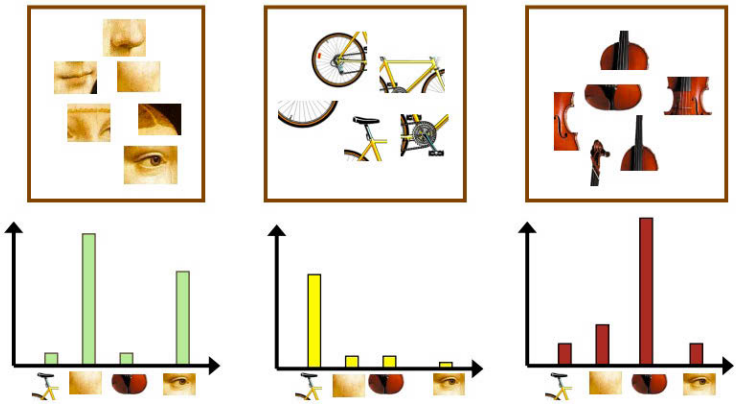

Figure 5: Using a bag-of-words model, image feature vectors are expressed by a histogram of the occurrences of representative descriptors within the image. Different levels of descriptor occurrence can provide information about the contents of an image.

A bag-of-words was generated for the clubfoot data set using VLFeat implemented in MATLAB.

2.2.1.4 Pyramid Histogram Of Visual Words

Pyramid Histogram of Visual Words (PHOW) feature vectors are spatial pyramid representations of SIFT bag-of-word descriptors [23].

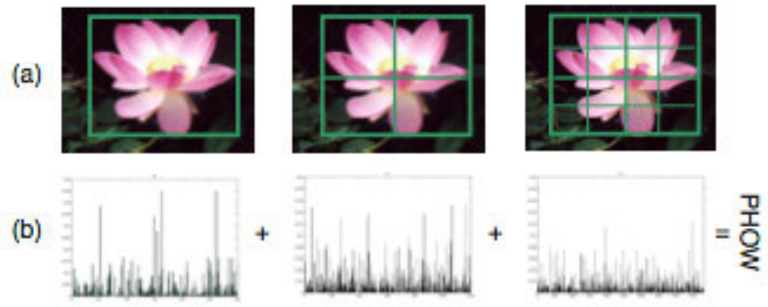

Figure 6: Pyramid Histogram of Visual Words (PHOW) feature vectors are spatial pyramid representations of SIFT bag-of-word descriptors.

Images are segmented with grids of varying sizes (1x1, 2x2, 4x4, etc.) and a bag-of-words histogram is constructed for each image space. The histograms are appended together to create a PHOW feature vector. By computing histograms at different levels, PHOW feature vectors are able to provide location information at varying granularities. VLFeat implements PHOW feature extraction according to the methodology outlined by Bosch et al. The PHOW feature vectors extracted using this implementation were used to classify images in the clubfoot data set.

2.2.1.5 Blind Image Quality Indices

Blind Image Quality Indices (BIQI) are numerical values that represent image degradation [24]. The BIQI value ranges from 0 to 100, with 0 representing the best quality

and 100 representing the worst. BIQI uses a no-reference image quality assessment algorithm that estimates the amount of distortion in an image. Types of distortion estimated by BIQI are image compression, white noise, Gaussian blur, and Fast fading. The degree of computed distortion determines the overall score assigned to an image.

BIQI scores were calculated for the images in the clubfoot data set in an effort to measure the images' degradation and overall quality.

2.2.1.6 Edge Detection

Edge detection is a technique used in image processing that is aimed at finding edges within an image. Typically, these techniques find edges by searching for intensity changes, or gradients, within an image [25]. This is achieved by convolving an image with one or more matrices, called "filters" or "kernels" [26]. Convolution assigns areas of continuity low values while areas with variation are assigned high values. Thus, edges are found.

Sobel edge detection, implemented in MATLAB was used to find edges in the clubfoot data set. The Sobel operator uses two 3x3 filters to find gradients in two directions.

-1	0	+1
-2	0	+2
-1	0	+1

+1	+2	+1
0	0	0
-1	-2	-1

Figure 7: Sobel edge detection uses two filters to find gradients along the x and y axes of an image.

2.2.1.7 Procrustes Analysis

Procrustes analysis is a statistical methodology used for comparing the similarity of shapes [27]. It achieves this in two steps. First, a test shape and template shape are superimposed using linear translation, scaling, and rotation. The second step is to determine

the Procrustes distance between the superimposed images. This is calculated by finding the sum of squared distances between points in the shapes.

Equation 1: The Procrustes distance is calculated by finding the sum of squared distances between points in two superimposed shapes.

$$d = \sqrt{\sum_{j=1}^{n} (x_{j1} - x_{j2})^2 + (y_{j1} - y_{j2})^2}$$

Procrustes distance is built-in functionality of MATLAB, and it was collected as a numerical measure of an image's orientation.

2.2.2 Machine Learning

Machine Learning has been one of the fastest growing fields, and widely deployed methods, for a large array of problems which require extraction of knowledge from many instances or examples, due to a lack of more traditional analytic modeling techniques. This research deployed a machine learning approach that can be used to automatically judge the acceptability of photos taken of the feet of children being treated in a remote, off-line situation – usually with a smart phone camera.

The "learning" process is complex, but at its core is the need to determine the distinguishing features embedded in an image, and then partition images based on the most distinguishing features. Once features have been extracted from an image or set of images, the computer must "learn" how these features relate to image properties. Learning these relationships allows the computer to predict properties of novel images.

Waikato Environment for Knowledge Analysis (WEKA) is a software tool that aids in data analysis and classification by providing implementations of many different types of

classifiers [28]. This section will provide a cursory overview of the field of machine learning as it applies to the specific classification problem being addressed in this thesis.

2.2.2.1 General principles

Machine learning has fundamental building blocks that are used for data classification [29]. The process starts with obtaining examples, or a data set from a real-world set of data. Each instance in a data set will have a set of features, or attributes. One of these attributes is the class attribute. The class attribute is the property that needs to be predicted, and is the "answer" to the classification.

The data set is used to train a classifier so it is able to predict the class attribute given the set of features. Once a classifier has been built, it can be used to predict the class of novel data.

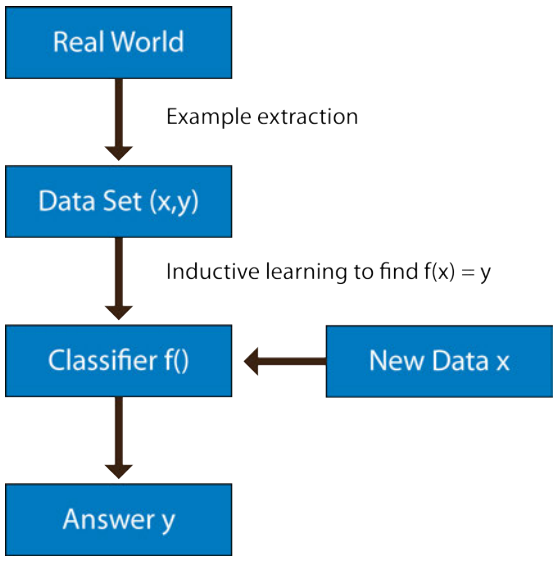

Figure 8: Machine learning is a process that builds a classifier trained on examples of real-world data. The classifier learns relationships between the data set's features and class attribute. It can then predict the class of novel data.

Data that is used to train a classifier should not be input to the classifier as a novel example for testing or prediction purposes. This is because the classifier has seen both the features and the class attributes of all the examples it was trained on, so it "knows" the answer to the classification problem. Predicting the class of an instance used for training is called overtraining, and inhibits the most valuable attribute of powerful predictors – generalization.

2.2.2.2 Classifiers

There are many types of classifiers that can be used in machine learning. Naïve Bayes and Support Vector Machine classifiers were found to perform well on the data set used in this research, and their performance was compared to that of a majority classifier.

2.2.2.2.1 Naïve Bayes Classifier

The Naïve Bayes classifier is a core methodology that has been used for at least 55 years [30]. It assumes the independence of attributes to calculate the probability that, given a set of attribute values, a condition occurs. This is summarized as Baye's Rule:

Equation 2: Naive Bayes classification makes predictions using Baye's Rule.

$$P(c_i|x) = \frac{P(x|c_i)P(c_i)}{P(x)}$$

WEKA implements a Naïve Bayes classifier following the methodology described by George H. John et al [31].

2.2.2.2.2 Support Vector Machine Classifier

A support vector machine constructs a set of hyperplanes that maximally separates a data set, allowing for classification [32]. Support vectors, or data points, are found along the

margin surrounding the hyperplane. Data points falling on opposite sides of the hyperplane have different predicted classes.

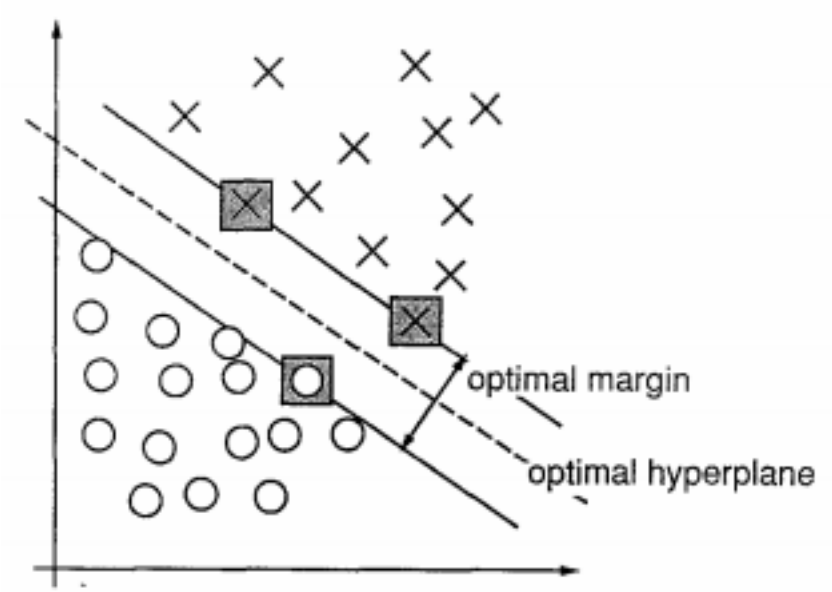

Figure 9: A support vector machine constructs a set of hyperplanes that maximally separates a data set, allowing for classification.

A SVM can be extended to higher-dimension spaces, and can take advantage of non-linear projections [33]. WEKA achieves this using the sequential minimal optimization algorithm presented by John C. Platt [34].

2.2.2.2.3 Majority Classifier

A majority classifier predicts the probability of the majority class in the data set. In WEKA's implementation, the majority class is found by taking the mean or mode of the data. Since the clubfoot data set used in this analysis has nominal class labels, the classifier predicts the mode. The accuracy of a majority classifier can be used as a baseline to measure the success of other classification types.

2.2.2.3 Classification Schemes

Multiple classifiers have been shown to improve classification results when trying to group images into meaningful categories [35]. Multiple classifiers are organized in a hierarchical fashion, which can be fixed or determined automatically [36; 37].

Classifiers in a hierarchical structure often have binary predictions, but it is not a requirement and some classification schemes take advantage of multi-class classifiers. Hierarchies are organized according to the nature of the data set being classified, and as a result, there are no standardized organization schemes for multiple classifiers. Several hierarchical classification schemes were developed and analyzed to predict the orientation and quality of images in the clubfoot data set.

2.2.2.4 10-Fold Cross Validation

Different classifiers and classification schemes will have better or worse performance on different sets of data. To find the classifier and classification scheme that works best on a particular data set, testing needs to be performed in a way that most accurately reflects the real environment for the deployed classifier among unknown instances, and that prevents overtraining.

10-fold cross validation breaks the data into 10 equal-sized, random, and stratified subsets of data [38]. The classifier is first trained on subsets 1-9, and then tested with subset 10. The process is repeated, building a total of 10 classifiers, where each subset serves as the testing set a single time. Classification of images in the clubfoot data set was tested using 10-fold cross validation.

Figure 10: 10-fold cross validation breaks the data into 10 equal-sized, random, and stratified subsets of data. In ten rounds of classification, a classifier is trained with nine of the subsets and tested with the remaining subset.

CHAPTER 3: METHODS AND RESULTS

3.1 Dataset

Dr. Jose Morcuende provided a set of 2,791 photos of clubfoot patients. These photos were taken by medical personnel in the University of Iowa Children's Hospital Clubfoot Clinic and by family members in the patients' homes. The dataset was organized for use by clinicians in the clinic, and needed to be "cleaned" and reorganized for analysis.

Duplicate images were found using the freeware tool, VisiPics [39]. With a strict filter setting, 662 images were found to have one or more duplicates. 672 images were removed from the set to eliminate all copies of these images.

Some images contained patient identifying information within the photo, such as names, date of birth, or faces. A total of 61 images with identifying information were manually removed from the dataset.

These filters reduced the data set to 2,058 photos, which were renamed using a shell script.

3.2 Assigning Orientation Scores

Orientation refers to the orientation of the foot within the photo. Photos with different orientations of the feet provide different clinical utility. Being able to automatically detect orientation could help ensure photos' utility.

3.2.1 Nominal Orientation Assignment

Photos were manually assigned an orientation attribute representing the orientation of the feet in the image. An image could be assigned one of six orientation attributes: "front", "back", "side", "front floor", "back floor", or "other".

Figure 11: Images were be assigned one of six orientation attributes: "front", "back", "side", "front floor", "back floor", or "other".

Table 1: Orientation assignments of 2,058 clubfoot images.

Orientation	Number of Images
Front	582
Back	380
Side	168
Front Floor	216
Back Floor	197
Other	515
Total	2058

Maria Miller, Clinic Nurse Coordinator at the University of Iowa Hospitals and Clinics, helped to define these categories by providing the instructions that are given to medical personnel and family members who are responsible for taking photographs of the feet.

3.2.2 Numeric Procrustes Values

Procrustes distance measures the difference between two shapes [27]. Images from the clubfoot data set were converted into shapes by applying MATLAB's Sobel edge detector to each image [12].

One image from each of the orientation categories "front", "back", "side", "front floor", and "back floor" was selected to be used as a template photo representing an orientation category. Template photos were selected for low background noise and clear, distinct lines outlining the feet.

Table 2: Template images for Procrustes analysis.

Orientation	Template Photo Number
Front	938
Back	164
Side	2685
Front Floor	1482
Back Floor	498

Each of the remaining images in the data set was mapped to each of the template images using Procrustes analysis. This generated five distance measures for each image, corresponding to the degree of difference between the image and each of the orientation category templates. Higher scores corresponded to a greater difference between the image and the template.

Table 3: Average Procrustes distances for matching image type to template.

Image Type	Average Procrustes Value For Matching Image Type to Template				
	Template				
	Front	Back	Side	Front Floor	Back Floor
Front	.57	.58	.61	.67	.60
Back	.58	.57	.59	.68	.59
Side	.64	.62	.59	.73	.62
Front Floor	.63	.60	.60	.69	.60
Back Floor	.63	.61	.61	.69	.61
Other	.62	.60	.62	.72	.59

It was expected that the average Procrustes distance measure for each image type would be minimized when calculated using the template of the same image type. This was true for two of the five orientations: front and side. Although back images that were matched with a back template yielded an average Procrustes distance measure lower than the measure generated when compared to other templates, the distance measure was not significantly different from the average distance measure of back images matched with the front template.

The Procrustes distance measures were not used as features for machine learning classification.

3.3 Assigning Quality Scores

Quality of an image in this project has two components. The first is the degree of image degradation, which can be the result of many factors including noise, artifacts, and blur. The second component of image quality is the clinical utility of the photo.

3.3.1 Numeric Values

In order to numerically represent an image's quality, a scoring system was developed. Two undergraduate students, Emily McDougall and Ashley Home, and one graduate student

used this system to rate each image's quality according to several features representing both image and diagnostic quality. Six criteria that could affect the quality of an image were defined. Each photo received a score ranging from (low) 1-5 (high) for each of the criteria. Example images were provided, along with a table for entering scores.

Table 4: Criteria for scoring quality of images.

Criteria	**Good (5)**	**Average (3)**	**Poor (1)**
Background	Solid	Not solid; Good contrast	Not solid; Poor contrast
Hands/Objects	None or minimal hands	Minimal hands	Hands or other objects
Exposure	Good	Good	Low/High
Noise	Low	Moderate	High
Blur	Low	Moderate	High
Percent of photo containing hands/legs	65-85%	55-65%	<55% or >85%

Pearson product-moment correlation coefficients showed that the scores assigned by the three individuals had medium to large positive correlation for all but the exposure and noise categories, which had low positive correlation [40].

Table 5: Pearson product-moment correlation coefficients of scorers' quality assignments.

	Pearson Product-Moment Correlation Coefficients		
	Emily-Ashley	**Emily-Amanda**	**Ashley-Amanda**
Background	.66	.61	.83
Hands/Objects	.65	.62	.71
Exposure	.47	.47	.40
Noise	.34	.33	.35
Blur	.62	.68	.69
Percent of photo containing hands/legs	.55	.42	.56
Total	.59	.58	.72

3.3.2 Aggregate Value

Two of the three students provided an aggregate, or nominal quality score of "good", "average", or "poor" for each image. This value represented their overall (somewhat subjective) opinion of the image's quality taking into consideration all of the numerical quality criteria with an unspecified weighting of the various underlying quality parameters.

Because one student did not provide a nominal quality assignment, a Naïve Bayes classifier was used to predict what their assignment would have been given their numeric quality scores and the mapping of underlying to overall scores of the other two students. This classifier was built using a training set consisting of the scores generated by the other two students. It used numeric quality scores as attributes and the nominal quality assignment as the class.

To assess the performance of the classifier, 10-fold cross validation was performed and was found to have an accuracy of 77%.

Steps were not taken to improve this classification because these predicted nominal scores were not directly used in future data analysis or classification. They were only used to judge quality classification performance against a pseudo-gold standard.

3.3.3 Nominal Quality Assignment

When predicting image quality in subsequent analyses, the nominal quality assignment was used as the class attribute. It was important to choose the most accurate and robust nominal quality assignments since they would serve as the class definition. Several methods were considered.

3.3.3.1 Appending

The dataset could be duplicated to reflect both students' nominal quality assignments. This would preserve their assignments, but could lead to classifier overtraining.

3.3.3.2 Averaging

Students' nominal assignments could be assigned a numerical value, allowing for the two scores to be averaged. This method was not chosen because rounding in either direction would create a bias in the data and diminish the value of each student's score.

3.3.3.3 2-Matching

The data set could be reduced to only include images where the two student's assigned nominal scores matched. This would preserve the nominal assignments while eliminating the concern of overtraining.

3.3.3.4 3-Matching

This is a more stringent extension of 2-matching. In this case, the data set could be reduced to only include images where the two student's assigned nominal scores and the third student's predicted nominal scores matched. This again preserved nominal assignments while eliminating the concern of overtraining. Furthermore, it reinforced that the nominal quality metric was accurate because the numeric scores of many images were considered in the prediction of the final student's scores.

The subset of data in which the three nominal quality scores matched was ultimately used in subsequent analysis of classifiers.

Table 6: The distribution of orientation assignments with respect to appended quality assignments.

	Front	**Back**	**Side**	**Front Floor**	**Back Floor**	**Other**	
Good	306	188	21	31	19	93	658
Average	582	405	150	176	170	397	1880
Poor	276	167	165	225	205	540	1578
	1164	760	336	432	394	1030	4116

Table 7: The distribution of orientation assignments with respect to 2-matching quality assignments.

	Front	**Back**	**Side**	**Front Floor**	**Back Floor**	**Other**	
Good	92	8	51	3	7	30	191
Average	180	44	129	50	46	116	565
Poor	80	75	43	69	57	200	524
	352	127	223	122	110	346	1280

Table 8: The distribution of orientation assignments with respect to 3-matching quality assignments.

	Front	**Back**	**Side**	**Front Floor**	**Back Floor**	**Other**	
Good	90	7	39	3	7	24	170
Average	81	22	75	12	26	61	277
Poor	38	20	39	18	30	110	225
	209	49	153	33	63	195	702

3.3.4 Blind Image Quality Indices

The Blind Image Quality Indices were calculated for each image in an effort to measure image degradation [24; 41]. The BIQI value ranges from 0 to 100, with 0 representing the best quality and 100 representing the worst.

The BIQI average scores were calculated for the set of 702 good, average, and poor images that had three matching nominal values.

Table 9: Average Blind Image Quality Indices (BIQI) scores for images of different qualities.

	Average BIQI
Good	48.5
Average	47.8
Poor	46.9

Performing a Student's t-test revealed that the mean BIQI scores were not significantly different. Thus, BIQI was not used as a determinant of quality.

3.4 Feature Extraction

Bag-of-words (BOW) feature extraction was implemented in MATLAB using the VLFeat toolbox [12; 19]. A bag of words was created by extracting Pyramid Histogram of Visual Words (PHOW) descriptors [23] from 50 randomly selected images and using k-means clustering to find K centroids that represent the extracted features. For each image, the PHOW features were mapped to this bag-of-words. Histograms of word frequency were generated at 2x2 and 4x4 scales. An image's feature vector was comprised of the appended histograms.

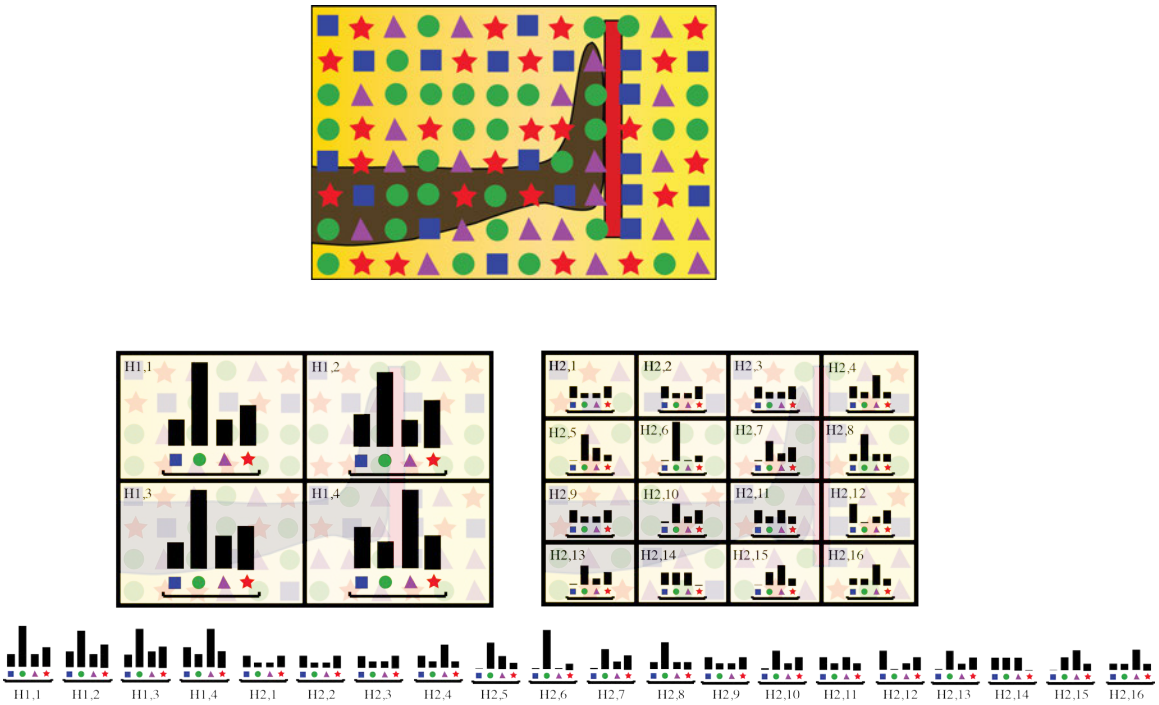

Figure 12: For each image, PHOW features were mapped to a bag-of-words. Histograms of word frequency were generated at 2x2 and 4x4 scales. An image's feature vector was comprised of the appended histograms.

3.4.1 Bag-of-Words Size

Features were calculated with a bag-of-words size of K=150 and K=600. Because PHOW features are multi-scale, this results in a feature vector that is 3,000 and 12,000 units long, respectively.

To compare the relative effectiveness of these two different sizes, a Naïve Bayes classifier was built to predict image orientation and quality using PHOW feature vector attributes. The classifier was assessed using 10 runs of 10-fold cross validation.

Table 10: Performance of orientation and quality classification using feature vectors built from a bag-of-words containing 150 and 600 words.

	Orientation Accuracy (%)		**Quality Accuracy (%)**	
Run	**150 BOW**	**600 BOW**	**150 BOW**	**600 BOW**
1	38.2	42.7	35.9	34.2
2	38.5	43	34.8	35.9
3	37.7	41.9	37.3	37.2
4	37.2	39.5	36.2	36.6
5	37.9	43.3	37.7	37.7
6	37.9	42.2	36.2	35.8
7	37.5	41.2	37.7	37.7
8	38	41.5	36.1	37.2
9	37	42.6	34.4	34.9
10	38.2	42.2	36.3	37
Average	37.81	42.01	36.26	36.42

Comparing the accuracy of the two classifications using a Student's t-test, it was found that orientation performance was significantly better when k=600 (p=1.6E-9). Performance of quality prediction was not significantly different.

3.5 Classifiers

Waikato Environment for Knowledge Analysis (WEKA) is a software tool that aids in data analysis and classification by providing implementations of many different types of classifiers [28]. Classifier types implemented in WEKA include Naïve Bayes, decision trees, nearest neighbor, logistic regression, support vector machine, ensemble, and majority classifiers [31; 34; 42-46].

Table 11: Classifier types implemented in the WEKA software application.

Classifier Type	WEKA Implementation
Naïve Bayes:	Naïve Bayes
Decision Tree:	J48
Nearest Neighbor:	IBk
Logistic Regression:	Logistic
Support Vector Machine:	SMO
Ensemble:	Random Forest
	JRip
Majority:	ZeroR

Different types of classifiers may be more or less successful than others at classification, depending on the nature of the data set, and the different capabilities of each classifier.

3.5.1 Optimal Classifier Selection

To compare the performance of different types of classifiers, orientation and quality classifiers were built using orientation and nominal quality scores as class variables, and using PHOW feature vectors generated with a k=150 bag-of-words. The classifiers' accuracies were assessed with 10 runs of 10-fold cross validation.

Table 12: The performance of different classifiers with respect to orientation and quality prediction.

Classifier	Orientation Accuracy (%)	Quality Accuracy (%)
Naïve Bayes	37.8	36.3
J48	30.2	34.9
Ibk, k=10	33.7	38.3
Logistic	25.5	36.9
SMO	41.6	39
Random Forest, I=10	36.5	37.2
JRip	33.2	37.2
Majority	29.8	39.5

The support vector machine (SMO) had the best performance, and Naïve Bayes classification was also a top performer.

3.6 Classification Schemes

While predicting image orientation was the initial goal of this project, predicting quality became a significant focus of this research as the nature of the difficulties in classifying orientation appeared to be dependent on image quality. Image quality was believed to have an effect upon a classifier's ability to classify orientation. Thus several hierarchical schemes were developed to address these considerations.

Classification was implemented in Java using the WEKA plugin. Each was assessed with 10 runs of 10-fold cross validation. Results were then compared to a majority classifier. Based on the results of BOW size and classifier testing, images were classified using a k=600 BOW PHOW feature vector, and an orientation or nominal quality score as the class variable. SMO and Naïve Bayes classifiers were used in this analysis.

3.6.1 Basic Orientation and Quality

Two basic orientation classifiers were developed to predict quality and classification. These classifiers function independently of each other. These simply predicted the quality or orientation class using a PHOW feature vector.

Both the Naïve Bayes and SMO classifiers in the basic orientation scheme performed better than a majority classifier. The SMO basic quality classifier also performed better than the majority classifier ($p=0.007$), although it did not have substantially better performance.

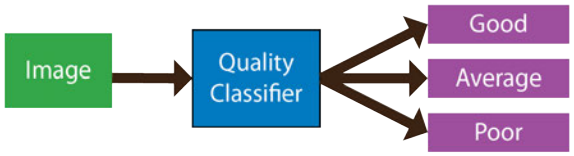

Figure 13: The basic quality classification scheme predicted an image's nominal quality value given a PHOW feature vector.

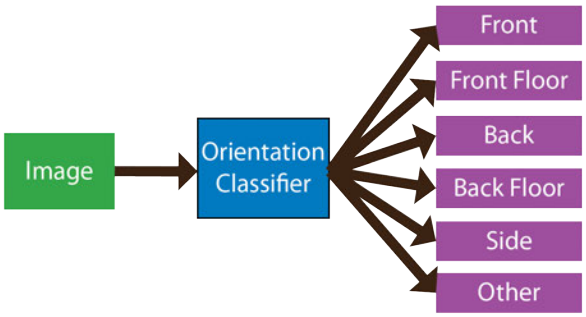

Figure 14: The basic orientation classification scheme predicted an image's orientation given a PHOW feature vector.

Table 13: A Naïve Bayes classifier was used in the basic classification scheme to predict an image's quality and orientation given a PHOW feature vector that was generated with a bag-of-words containing 600 features.

Naïve Bayes 600 BOW Basic Classification Scheme		
Run	**Orientation Accuracy (%)**	**Quality Accuracy (%)**
1	42.7	34.2
2	43.0	35.9
3	41.9	37.3
4	39.5	36.6
5	43.3	37.7
6	42.2	35.8
7	41.2	37.7
8	41.5	37.2
9	42.6	34.9
10	42.2	37.0
Average	**42.0**	**36.4**

Table 14: A SMO classifier was used in the basic classification scheme to predict an image's quality and orientation given a PHOW feature vector that was generated with a bag-of-words containing 600 features.

SMO 600 BOW Basic Classification Scheme		
Run	**Orientation Accuracy (%)**	**Quality Accuracy (%)**
1	47.6	40.9
2	47.9	40.7
3	45.9	41.5
4	48.3	40.0
5	47.0	40.2
6	45.0	38.9
7	47.4	41.2
8	46.7	41.2
9	47.3	40.0
10	48.3	39.5
Average	**47.1**	**40.4**

3.6.2 Quality on All

Three quality-on-all classifiers were built to determine whether features from images of a certain quality (good, average, or poor) were generally better at classifying all image types. It was hypothesized that building a classifier with feature vectors from either good or poor images may have better performance. Good images might have this result because there may be fewer "noisy" features. Poor images might have this result because the most important features would have to be distilled during training.

Three different classifiers were built using the PHOW feature vectors of images within a single nominal quality class (good, average, or poor). Each classifier was used to classify every image in the data set. This classification scheme shows the performance of classifiers built with certain quality images.

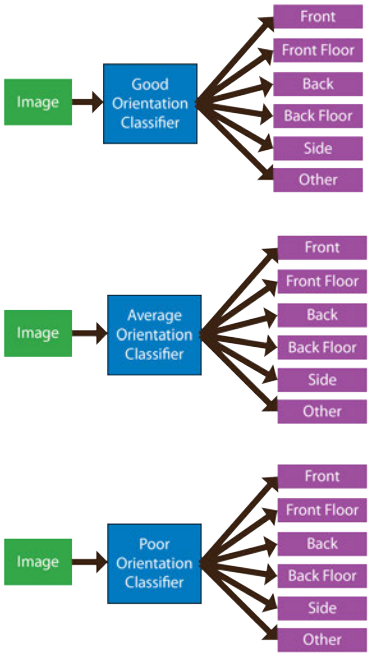

Figure 15: The quality-based classifier was used to classify all images in the data set using a classifier trained only on images of a certain quality. The results of this classifier would show if features from images of a certain quality (good, average, or poor) were generally better at classifying all image types.

Table 15: Three Naïve Bayes classifiers were used in the Quality on All classification scheme to predict an image's orientation given a PHOW feature vector that was generated with a bag-of-words containing 600 features.

Naïve Bayes 600 BOW Quality on All Classification Scheme			
Run	**Good Orientation Accuracy (%)**	**Average Orientation Accuracy (%)**	**Poor Orientation Accuracy (%)**
1	35.5	38.9	34.5
2	36.8	38.9	35.6
3	34.3	38.2	34.8
4	35.6	38.6	35.5
5	35.5	37.7	35.6
6	34.2	38.7	35.3
7	33.3	39.9	35.6
8	35.2	38.5	35.8
9	34.5	38.6	34.9
10	35.2	37.6	35.3
Average	**35.0**	**38.6**	**35.3**

Table 16: Three SMO classifiers were used in the Quality on All classification scheme to predict an image's orientation given a PHOW feature vector that was generated with a bag-of-words containing 600 features.

SMO 600 BOW Quality on All Classification Scheme			
Run	**Good Orientation Accuracy (%)**	**Average Orientation Accuracy (%)**	**Poor Orientation Accuracy (%)**
1	36.9	41.2	37.5
2	36.8	41.7	37.9
3	36.3	40.6	38.3
4	37.0	41.9	39.3
5	34.9	40.7	37.9
6	35.9	41.9	37.7
7	35.6	41.6	38.9
8	36.6	41.0	38.2
9	35.5	40.7	38.7
10	37.2	41.2	38.0
Average	**36.3**	**41.3**	**38.2**

Although the overall performance of all three classifiers was better than a majority classification, they also performed worse than the basic classification scheme.

The classifiers' performance corresponded to the number of photos available for training. There were 170 good quality images, 277 average quality images, and 255 poor quality images. The lower performance of all classifiers reflects the diminished size of the training set.

These results do not support the argument that a using a classifier built with feature vectors of only high or low-quality images is better at classifying all types of images.

3.6.3 Quality on Quality

Three quality-on-quality classifiers were built to determine if features from images of a certain quality (good, average, or poor) were generally better at classifying images of the same type. If this were true, it would motivate hierarchical classification where images were first classified according to quality and then classified according to orientation based on those results.

Three different classifiers, good, average, and poor, were built with PHOW feature vectors from only good, average, or poor images. Each classifier was used to classify only images of the same quality.

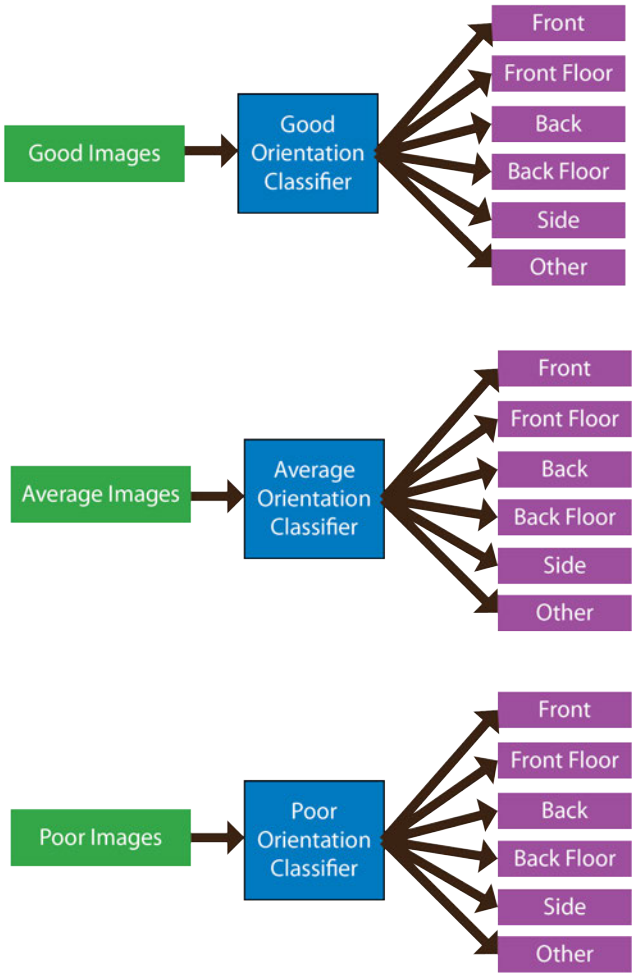

Figure 16: A Quality on Quality classification scheme built three orientation classifiers using images of a certain quality (good, average, poor). The classifiers were used to classify only images of the same quality. The results of this classifier would show if features from images of a certain quality (good, average, or poor) were generally better at classifying images of the same type.

Table 17: Three majority classifiers were used in the Quality on Quality classification scheme to predict an image's orientation.

Majority Quality on Quality Classification Scheme			
Overall Orientation Accuracy (%)	**Good Orientation Accuracy (%)**	**Average Orientation Accuracy (%)**	**Poor Orientation Accuracy (%)**
29.8	52.9	29.2	43.1

Table 18: Three Naïve Bayes classifiers were used in the Quality on Quality classification scheme to predict an image's orientation given a PHOW feature vector that was generated with a bag-of-words containing 600 features.

Naïve Bayes 600 BOW Quality on Quality Classification Scheme				
Run	**Overall Orientation Accuracy (%)**	**Good Orientation Accuracy (%)**	**Average Orientation Accuracy (%)**	**Poor Orientation Accuracy (%)**
1	44.2	42.4	40.4	52.9
2	46.0	45.1	40.4	56.5
3	44.0	42.0	39.0	55.3
4	45.3	43.1	40.1	57.1
5	45.3	44.7	39.4	55.9
6	45.9	41.6	41.2	60.0
7	46.3	42.0	41.9	60.0
8	45.7	42.7	41.2	57.6
9	44.2	41.6	39.0	56.5
10	45.4	44.7	40.1	55.3
Average	**45.2**	**43.0**	**40.3**	**56.7**

Table 19: Three SMO classifiers were used in the Quality on Quality classification scheme to predict an image's orientation given a PHOW feature vector that was generated with a bag-of-words containing 600 features.

SMO 600 BOW Quality on Quality Classification Scheme				
Run	**Overall Orientation Accuracy (%)**	**Good Orientation Accuracy (%)**	**Average Orientation Accuracy (%)**	**Poor Orientation Accuracy (%)**
1	46.7	46.3	41.5	55.9
2	45.3	44.3	41.2	53.5
3	45.9	44.3	40.4	57.1
4	47.3	45.1	43.3	57.1
5	45.4	44.7	39.0	57.1
6	45.7	43.9	41.9	54.7
7	45.9	43.5	42.6	54.7
8	45.4	43.1	40.8	56.5
9	44.9	45.5	37.9	55.3
10	46.7	46.7	41.9	54.7
Average	**45.9**	**44.7**	**41.1**	**55.7**

Using a quality-on-quality classification scheme, overall performance of both the Naïve Bayes and SMO classifiers was better than a majority classifier. Using this scheme, the Naïve Bayes classifiers performed better than if using a basic classification scheme ($p=1.07E-6$). On the other hand, the SMO classifiers performed better under a basic classification scheme ($p=0.009$).

Looking at the performance of the individual classifiers compared to a majority classification, it appears that classifying images with a classifier built with images of the same quality is only advantageous for images of average and poor quality. It is possible that the diminished training set size of the good image class contributed to its poor performance.

The results of this classification scheme show how a quality-based orientation classifier would perform if quality could be accurately predicted every time. The results of the Naïve Bayes classifier motivate the development of a hierarchical classification scheme.

3.6.4 Quality-Based

A hierarchical classification scheme was developed to first classify images according to quality, and then classify images' orientation based on the predicted quality. This classification scheme consists of four classifiers built using PHOW feature vectors. There is one basic quality classifier and three quality-based orientation classifiers.

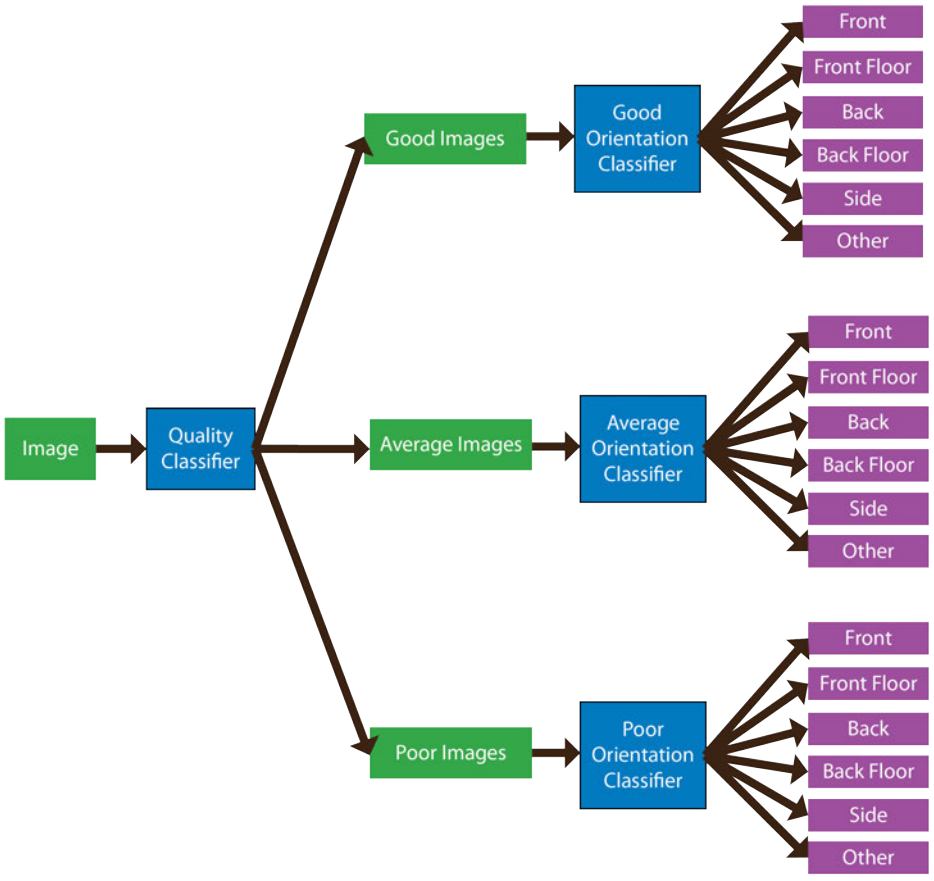

Figure 17: The hierarchical quality-based classification scheme was developed to first classify images according to quality, and then classify images' orientation based on the predicted quality. This classification scheme consists of four classifiers: one basic quality classifier and three quality-based orientation classifiers.

Table 20: Naïve Bayes classifiers were used in the quality-based classification scheme to predict an image's quality and orientation given a PHOW feature vector that was generated with a bag-of-words containing 600 features.

Naïve Bayes 600 BOW Quality-Based Classification Scheme					
Run	**Quality Accuracy (%)**	**Overall Orientation Accuracy (%)**	**Good Orientation Accuracy (%)**	**Average Orientation Accuracy (%)**	**Poor Orientation Accuracy (%)**
1	34.2	40.2	42.8	37.2	40.9
2	35.9	41.2	44.8	35.7	43.9
3	37.3	38.6	39.3	36.8	40.0
4	36.6	39.6	41.4	33.7	44.6
5	37.7	39.7	39.5	34.8	46.6
6	35.8	40.3	39.7	38.5	43.6
7	37.7	41.3	39.5	41.9	43.1
8	37.2	41.2	42.2	38.0	43.8
9	34.9	40.2	40.6	36.3	44.9
10	37.0	41.5	44.7	34.7	46.3
Ave.	**36.4**	**40.4**	**41.5**	**36.8**	**43.8**

Table 21: SMO classifiers were used in the quality-based classification scheme to predict an image's quality and orientation given a PHOW feature vector that was generated with a bag-of-words containing 600 features.

SMO 600 BOW Quality-Based Classification Scheme					
Run	**Quality Accuracy (%)**	**Overall Orientation Accuracy (%)**	**Good Orientation Accuracy (%)**	**Average Orientation Accuracy (%)**	**Poor Orientation Accuracy (%)**
1	40.9	43.2	40.2	43.9	47.2
2	40.7	43.9	41.0	44.6	47.3
3	41.5	42.9	41.4	40.9	51.2
4	40.0	45.0	41.2	44.3	54.5
5	40.2	43.3	40.3	42.1	53.4
6	38.9	43.2	40.9	41.3	51.9
7	41.2	44.3	42.0	43.8	50.4
8	41.2	43.9	42.0	43.6	49.1
9	40.0	41.3	38.0	41.0	48.5
10	39.5	43.6	40.3	42.9	51.9
Ave.	**40.4**	**43.5**	**40.7**	**42.8**	**50.5**

Both the SMO and Naïve Bayes classifiers performed better than a majority classifier, but did not have better performance than did a basic classification scheme. This is because of the relatively poor performance of the quality classifier. If the performance of the quality classifier was better, it is expected that the Naïve Bayes classifier would have better performance, based on the quality-on-quality analysis.

3.6.5 Voting

The poor performance of the basic quality classifier resulted in less-than-optimal performance of a quality-based classification scheme. Rather than using the basic quality classifier, the voting classification scheme used three binary quality classifiers that predicted whether or not an image was of a certain quality (good, average, poor). The binary quality classifiers were trained using PHOW features of all the images and a boolean class label.

A voting mechanism was then used to determine an image's quality based on the output of the three binary quality classifiers. The voting mechanism worked in a similar way as a grade point average calculation. It converted each nominal quality value into a numerical value, or weight. It then used these weights to average the output of the three binary quality classifiers.

Equation 3: A voting mechanism was used to determine an image's quality based on the output of three binary quality classifiers. It converted each nominal quality value into a numerical value, or weight. It then used these weights to average the output of the three binary quality classifiers.

$$w_g = 3$$
$$w_a = 2$$
$$w_p = 1$$
$$c_i = output\ of\ binary\ classifier$$
$$if\ c_g + c_a + c_p = 0, \quad \text{vote} = 1$$
$$if\ c_g + c_a + c_p \neq 0, \quad \text{vote} = Floor\left(\frac{c_g \times w_g + c_a \times w_a + c_p \times w_p}{c_g + c_a + c_p}\right)$$

Table 22: Quality assignments of the voting mechanism, given all possible combinations of binary classifier outputs.

Good	Average	Poor	Vote	Quality
0	0	0	1	Poor
0	0	1	1	Poor
0	1	0	2	Average
1	0	0	3	Good
0	1	1	1	Poor
1	1	0	2	Average
1	0	1	2	Average
1	1	1	2	Average

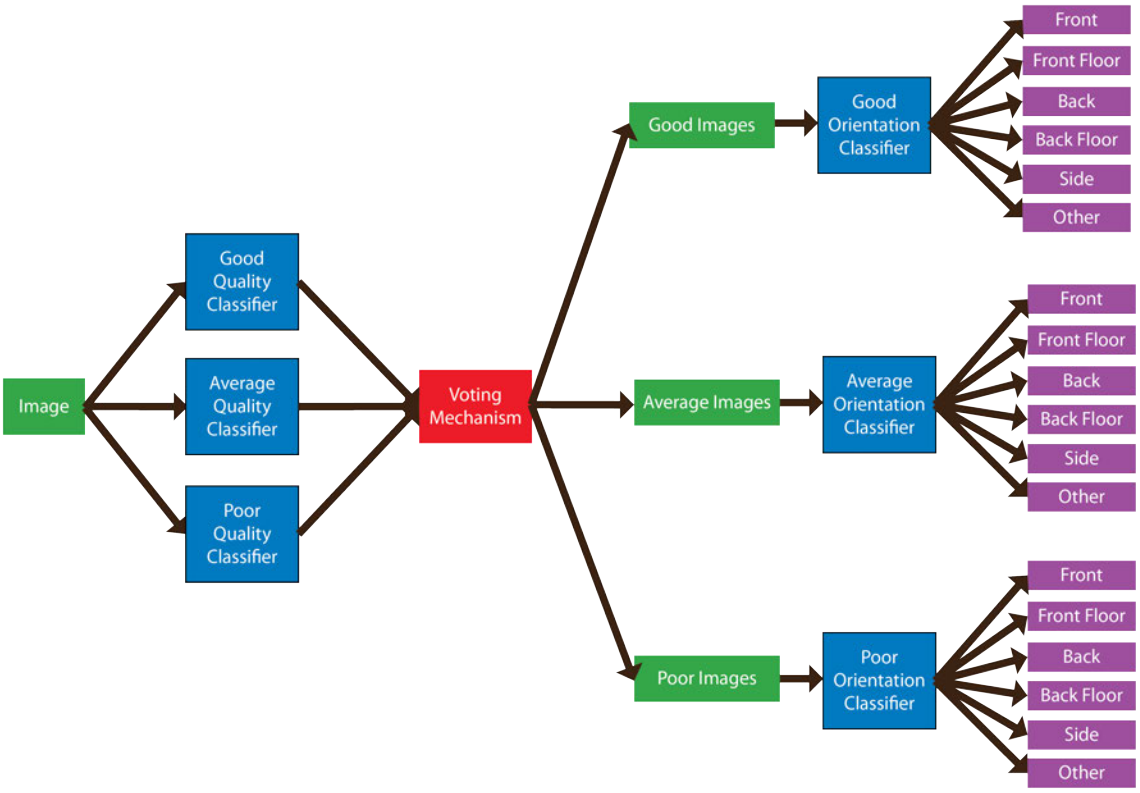

Figure 18: A voting classification scheme first classified images according to quality, and then classified images' orientation based on the predicted quality. It performed quality classification using three binary quality classifiers that predicted whether or not an image was of a certain quality (good, average, poor) and a voting mechanism to average the output of the binary classifiers. Orientation was then predicted using three quality-based orientation classifiers.

Table 23: Three majority classifiers were used in the voting classification scheme to predict an image's quality and orientation.

Majority Voting Classification Scheme		
Binary Quality Accuracy (%)	**Quality Accuracy (%)**	**Orientation Accuracy (%)**
59.6	39.5	29.8

Table 24: Naïve Bayes classifiers were used in the voting classification scheme to predict an image's quality and orientation given a PHOW feature vector that was generated with a bag-of-words containing 600 features.

Naïve Bayes 600 BOW Voting Classification Scheme			
Run	**Binary Quality Accuracy (%)**	**Quality Accuracy (%)**	**Orientation Accuracy (%)**
1	57.8	12.0	34.3
2	57.5	12.4	34.3
3	58.6	11.4	34.3
4	58.4	11.3	34.6
5	58.1	13.1	34.9
6	57.6	11.8	33.3
7	59.4	12.1	35.3
8	58.9	11.7	34.2
9	57.7	11.3	33.2
10	59.1	11.1	33.8
Ave.	**58.3**	**11.8**	**34.2**

Table 25: SMO classifiers were used in the voting classification scheme to predict an image's quality and orientation given a PHOW feature vector that was generated with a bag-of-words containing 600 features.

SMO 600 BOW Voting Classification Scheme			
Run	**Binary Quality Accuracy (%)**	**Quality Accuracy (%)**	**Orientation Accuracy (%)**
1	62.5	17.1	36.5
2	63.1	15.0	37.9
3	62.9	14.5	37.0
4	62.3	12.4	37.2
5	61.6	13.1	36.0
6	61.5	13.4	36.5
7	63.2	14.5	38.0
8	62.8	13.1	37.0
9	62.3	15.2	37.0
10	62.1	13.2	36.3
Ave.	**62.4**	**14.2**	**36.9**

The binary classifiers have an overall majority classification accuracy of 59.6%. The Naïve Bayes classifier did not perform better than the majority classifier, but the SMO classifier did perform better ($p=0.002$). Meanwhile, both the Naïve Bayes and SMO classifiers have very poor performance with nominal classification. This indicates that the voting mechanism used did not combine the output of the binary quality classifiers in a meaningful way.

As a result of poor quality classification, the accuracy of the orientation classifiers is lower than expected for both the Naïve Bayes and SMO classifiers.

CHAPTER 4: CONCLUSION

Clubfoot is a congenital foot disorder that is prevalent in the developing world, and can be treated with an inexpensive, non-surgical method called the Ponseti Method [2]. Treatment providers collect patient photos for diagnosis, patient monitoring, and physician quality assessment. Classification of patient photos could provide immediate feedback to those taking patient photos, helping to ensure that the image is of good quality and the foot is oriented correctly at the time of image capture. Classification could also serve as the basis for automated image analysis that could reduce the workload of a busy staff. Methodology was developed for such a classification using image processing and machine learning techniques.

Pyramid Histogram of Visual Words (PHOW) feature vectors were extracted from a set of clubfoot images, and were used to classify the data set according to image quality and foot orientation [23]. Different classifier types and classification schemes were systematically tested to achieve the best possible classification results.

Using two independent SMO quality and orientation classifiers yielded the best results, with significantly better performance than a majority classifier ($p_{quality}$=0.007, $p_{orientation}$=1.72E-12). Naïve Bayes hierarchical classification could also yield similar performance if quality could be better predicted, as evidenced by the results of the quality-on-quality classification.

Table 26: The performance of Naïve Bayes and SMO quality and orientation classification using different classification schemes.

	Quality Accuracy (%)		**Orientation Accuracy (%)**	
Classification Scheme	**Naïve Bayes**	**SMO**	**Naïve Bayes**	**SMO**
Basic	36.4	40.4	42	47.6
Quality on Quality	N/A	N/A	45.2	45.9
Quality-Based	36.4	40.4	40.4	43.5
Voting	11.8	14.2	34.2	36.9

The success of the independent SMO classifiers is modest; respective classification accuracies of 40.4% and 47.6% for quality and orientation prediction are 0.9% and 17.8% higher than the majority classifier's performance of 39.5% and 29.8%. Still, this level of performance is not robust enough for clinical use.

It is possible that other feature types may perform better or worse at quality and orientation classification of clubfoot images. The methodology and classification framework outlined in this thesis provides a sound way for testing such future hypothesis.

APPENDIX A: PROCRUSTES SCORE

```
% --------------------------------------------------------------------
function scores = procrustes_scores(path)
% --------------------------------------------------------------------

%Get image location
files = dir(path);

%get number of files
num_images = numel(files) - 3;

%get the templates
f = getTemplate(path, 'front');
ff = getTemplate(path, 'front_floor');
b = getTemplate(path, 'back');
bf = getTemplate(path, 'back_floor');
s = getTemplate(path, 'side');

%template_min = minimum(size(f,2), size(ff,2), size(b,2), size(bf,2),
size(s,2));

%initialize score array
scores = zeros(num_images, 6);

for i = 4:numel(files)
     filename = files(i).name
     filepath = strcat(path, '/', filename);

     %get image number
     filenumber = strrep(filename, 'quality', '');
     filenumber = strrep(filenumber, '.JPG', '');
     number = str2num(filenumber);

     %Perform edge detection
    BW = getEdges(filepath);

     %Convert to double
     im = im2double(BW);

     %compare image to templates
     if size(im,2) > size(f,2)
       d_f = procrustes(im,f);

     else
       d_f = procrustes(f,im);
     end


     if size(im,2) > size(ff,2)
       d_ff = procrustes(im,ff);
     else
       d_ff = procrustes(ff,im);
     end


     if size(im,2) > size(b,2)
       d_b = procrustes(im,b);
     else
       d_b = procrustes(b,im);
     end
```

```
      if size(im,2) > size(bf,2)
        d_bf = procrustes(im,bf);
      else
        d_bf = procrustes(bf,im);
      end


      if size(im,2) > size(s,2)
        d_s = procrustes(im,s);
      else
        d_s = procrustes(s,im);
      end

      %set scores
      scores(i-3, 1) = number;
      scores(i-3, 2) = d_f;
      scores(i-3, 3) = d_ff;
      scores(i-3, 4) = d_b;
      scores(i-3, 5) = d_bf;
      scores(i-3, 6) = d_s;

end

xlswrite('procrustesScores', scores);

% --------------------------------------------------------------------
function im = standarizeImage(im)
% --------------------------------------------------------------------

im = im2single(im) ;
if size(im,1) > 220, im = imresize(im, [220 NaN]) ; end

% --------------------------------------------------------------------
function im = getGrayscaleImage(im)
% --------------------------------------------------------------------

s = size(im);

%convert to grayscale
if(size(s,2) == 3)
      im = rgb2gray(im);
else
      %do nothing
end

% --------------------------------------------------------------------
function im = getEdges(filepath)
% --------------------------------------------------------------------
      %load image
      image = imread(filepath);

      %standardize image
      image = standarizeImage(image);

      %convert to black and white
      image = getGrayscaleImage(image);

      %perform edge detection
      im = edge(image);
      %BW = edge(image, 'canny');

% --------------------------------------------------------------------
```

```
function im = getTemplate(path, orientation)
% ------------------------------------------------------------------------

%load image file
     if(strcmp(orientation, 'front'))
       filename =  'quality938.JPG';
     elseif(strcmp(orientation, 'front_floor'))
       filename =  'quality1482.JPG';
     elseif(strcmp(orientation, 'back'))
       filename =  'quality164.JPG';
     elseif(strcmp(orientation, 'back_floor'))
       filename =  'quality498.JPG';
     elseif(strcmp(orientation, 'side'))
       filename =  'quality2685.JPG';
     else
       filename =  '';
     end

     filepath = strcat(path, '/', filename);

%Perform edge detection
    BW = getEdges(filepath);

%Convert to double
       im = im2double(BW);
```

APPENDIX B: CLASSIFICATION SCHEMES

```
/*********************************************
 * Name: ClubfootClassification
 * Author: Amanda De Hoedt
 * Description: Classifies clubfoot image data
 * Input: args[0] = name of data set
 *        args[1] = classification scheme
 *        args[3] = classifier type
 *********************************************/

import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;
import weka.filters.Filter;
import weka.filters.unsupervised.attribute.Remove;

public class ClubfootClassification {

  Instances originalData;
  DataSource source;

  public static void main(String[] args) throws Exception{

    //Load original data file
    DataSet originalData = new DataSet();

    if(args[0] != ""){
      //set the data set according to the value
      String filename = getFileName(args[0]);
      originalData.setDataSource(filename);
    }

    else
    {
      String filename = "";
      //filename = getFileName("Average600");
      //filename = getFileName("Three150");
      //filename = getFileName("Three600");
      //filename = getFileName("Two600");
      //filename = getFileName("600");
      //filename = getFileName("350");

      originalData.setDataSource(filename);
    }

    originalData.loadFile();
    originalData.setAttribute("Type");

    Instances data = new Instances(originalData.getDataSet());

    //Remove the numerical quality attributes
    Instances preProcessedData = filterNumbericalQualityAttributes(data);
    data = null;

    if(args[1].equals("QualityOnQuality")){
      QualityClassifierOnQualityData qualityClassifier = new
QualityClassifierOnQualityData(preProcessedData, args[2]);
      qualityClassifier.run();
      qualityClassifier = null;
    }
    if(args[1].equals("QualityOnAll")){
      QualityBasedOrientationClassifier cbClassifier = new
```

```
QualityBasedOrientationClassifier(preProcessedData, args[2]);
      cbClassifier.run();
      cbClassifier = null;
    }
    //BASIC CLASSIFICATION
    if(args[1].equals("Basic")){
      BasicClassifier basic = new BasicClassifier(preProcessedData, args[2]);
      basic.run();
      basic = null;
    }
    //BASIC ORIENTATION CLASSIFICATION
    if(args[1].equals("BasicOrientation")){
      BasicOrientationClassifier basicOrientation = new
BasicOrientationClassifier(preProcessedData, args[2]);
      basicOrientation.run();
      basicOrientation = null;
    }
    //QUALITY-BASED CLASSIFICATION
    if(args[1].equals("QualityBased")){
      QualityBasedClassifier qualityBased = new
QualityBasedClassifier(preProcessedData, args[2]);
      qualityBased.run();
      qualityBased = null;
    }
    //VOTING CLASSIFICATION
    if(args[1].equals("Voting")){
      VotingClassifier voting = new VotingClassifier(preProcessedData,
args[2]);
      voting.run();
      voting = null;
    }

  }//end main

  private static Instances filterNumbericalQualityAttributes(Instances d)
throws Exception{
    Remove remove = new Remove();
    String[] options = new String[2];
    options[0] = "-R";  // "range"
    options[1] = "1-7"; //range of attributes
    remove.setOptions(options);
    remove.setInputFormat(d);
    Instances preProcessedData = Filter.useFilter(d, remove);

    return preProcessedData;

  }

  private static String getFileName(String s){
    if(s.equals("Average600")){
      //Quality metric averaged across 2 users + 1 predicted
      return "qualityScoresSiftLongAveragedQualityAllNoDupsWeka.arff";
    }
    else if(s.equals("Three150")){
      //Quality metric same across 2 users + 1 predicted 150 BOW Size
      return "qualityScores150WordSiftLongThreeSameQualityAllNoDupsWeka.arff";
    }
    else if(s.equals("Three600")){
      //Quality metric same across 2 users + 1 predicted
      return "qualityScoresSiftLongThreeSameQualityAllNoDupsWeka.arff";
    }
    else if(s.equals("Two600"))
    {
```

```
      //Quality metric same across 2 users
      return "qualityScoresSiftLongTwoSameQualityAllNoDupsWeka.arff";
    }
    else if(s.equals("600")){
      //Individual Quality metrics
      return "qualityScoresSiftLongAllNoDupsWeka.arff";
    }
    else if(s.equals("350")){
      //Individual Quality metrics + short SIFT features
      return "qualityScoresSiftAllNoDupsWeka.arff";
    }
    return "";
  }

}//end class



/**********************************************
 * Name: DataSet
 * Author: Amanda De Hoedt
 * Description: Performs functions related to data sets
 **********************************************/
import weka.core.Attribute;
 import weka.core.Instances;
import weka.core.converters.ConverterUtils.DataSource;


public class DataSet {
DataSource source;
Instances data;

public static void main(String[] args)
{
  //do main stuff here
}

public void setDataSource(String path) throws Exception{
  source = new DataSource(path);
  System.out.print("Data Source Set." + '\n');
}

public void loadFile() throws Exception {
  //Load data file
  System.out.print("Loading data file." + '\n');
  data = source.getDataSet();
  System.out.print("Data file loaded successfully." + '\n');
}

public void setAttribute(String attributeName) {
  if (data.classIndex() == -1){
    Attribute classAttribute = data.attribute(attributeName);
    data.setClassIndex(classAttribute.index());
  }
}

public void createDataSetFromExisting(Instances oldData){
  data = new Instances(oldData);
}

public Instances getDataSet(){
  return data;
}
```

```
}


/*********************************************
 * Name: ClassificationScheme
 * Author: Amanda De Hoedt
 * Description: Defines common elements of
 *              classification scheme
 *              functionality. Performs functions
 *              related to data classification
 *********************************************/
import weka.classifiers.Classifier;
import weka.classifiers.Evaluation;
import weka.classifiers.bayes.NaiveBayes;
import weka.classifiers.functions.Logistic;
import weka.classifiers.functions.MultilayerPerceptron;
import weka.classifiers.functions.SMO;
import weka.classifiers.functions.SimpleLinearRegression;
import weka.classifiers.lazy.IBk;
import weka.classifiers.rules.JRip;
import weka.classifiers.rules.ZeroR;
import weka.classifiers.trees.J48;
import weka.classifiers.trees.RandomForest;
import weka.core.Attribute;
import weka.core.Instances;
import weka.filters.Filter;
import weka.filters.unsupervised.attribute.Remove;
import weka.filters.unsupervised.instance.RemoveWithValues;


public abstract class ClassificationScheme {
  private Instances data;
  private Attribute subjectiveQualityAttribute;
  private Attribute orientationAttribute;
  private int runs;
  private String type;

  public ClassificationScheme(Instances d){
    setRuns(10);
    setData(new Instances(d));
    setClassAttributes(d);
  }

  public void setClassificationType(String s)
  {
    type = s;
  }

  public String getClassificationType()
  {
    return type;
  }

  public Classifier getClassifier() throws Exception
  {
    Classifier classifier = null;

    if(type.equals("NaieveBayes")){
      classifier = (Classifier) new NaiveBayes();
    }
    else if(type.equals("MultilayerPerceptron")){
```

```
      classifier = (Classifier) new MultilayerPerceptron();
    }
    else if(type.equals("J48")){
      classifier = (Classifier) new J48();
    }
    else if(type.equals("Logistic")){
      classifier = (Classifier) new Logistic();
    }
    else if(type.equals("SMO")){
      classifier = (Classifier) new SMO();
    }
    else if(type.equals("RandomForest")){
      classifier = (Classifier) new RandomForest();
    }
    else if(type.equals("JRip")){
      classifier = (Classifier) new JRip();
    }
    else if(type.equals("SimpleLinearRegression")){
      classifier = (Classifier) new SimpleLinearRegression();
    }
    else if(type.equals("ZeroR")){
      classifier = (Classifier) new ZeroR();
    }
    else if(type.equals("IBk")){
      classifier = (Classifier) new IBk();
      String[] ibkOptions = new String[2];
      ibkOptions[0] = "-K";  // "range"
      ibkOptions[1] = "10"; //range of attributes
      classifier.setOptions(ibkOptions);
    }

    return classifier;
  }

  public Instances removeBinaryQuality(String quality, Instances d) throws
Exception{

    Attribute rAttribute = d.attribute(quality);

    Remove remove = new Remove();
    String[] options = new String[2];
    options[0] = "-R";  // "range"
    options[1] = String.valueOf(rAttribute.index()+1); //range of attributes
    remove.setOptions(options);
    remove.setInputFormat(d);
    Instances preProcessedData = Filter.useFilter(d, remove);

    return preProcessedData;
  }

  public void setClassAttributes(Instances d){
    setSubjectiveQualityAttribute(d.attribute("Subjective"));
    setOrientationAttribute(d.attribute("Type"));
  }

  public void setClassAttributes(Instances s, Instances o){
    setSubjectiveQualityAttribute(s.attribute("Subjective"));
    setOrientationAttribute(o.attribute("Type"));
  }

  public void printResults(Evaluation e){
    String strSummaryQuality = e.toSummaryString();
    System.out.print(strSummaryQuality);
```

```
    try {
      String strClassDetails = e.toClassDetailsString();
      System.out.println();
      System.out.print(strClassDetails);
      System.out.println();
    } catch (Exception e1) {
      // TODO Auto-generated catch block
      //e1.printStackTrace();
    }

     // Get the confusion matrix
     double[][] cmMatrixQuality = e.confusionMatrix();
     for(int row_i=0; row_i<cmMatrixQuality.length; row_i++){
       for(int col_i=0; col_i<cmMatrixQuality.length; col_i++){
         System.out.print(cmMatrixQuality[row_i][col_i]);
         System.out.print("|");
       }
       System.out.println();
     }
  }

  public static String getQualityFromInt(int i){
    if(i == 1){
      return "average";
    }
    else if(i == 2){
      return "poor";
    }
    else if(i == 3){
      return "good";
    }
    return null;
  } //end getQualityFromInt

  public static Integer getIntFromQuality(String s){
    if(s == "average"){
      return 1;
    }
    else if(s == "poor"){
      return 2;
    }
    else if(s == "good"){
      return 3;
    }
    return null;
  } //end getQualityFromInt

  public Instances filterOnQuality(String q, Instances train) throws Exception{

    setClassAttributes(train);

    String[] optionsSelectGood = new String[4];
    optionsSelectGood[0] = "-C";  // "attribute"
    optionsSelectGood[1] =
String.valueOf(subjectiveQualityAttribute.index()+1); //subjective attribute
    optionsSelectGood[2] = "-L";  // "label"
    optionsSelectGood[3] = Integer.toString(getIntFromQuality(q));

    RemoveWithValues selectGood = new RemoveWithValues();
    selectGood.setOptions(optionsSelectGood);
    selectGood.setInvertSelection(true);
    selectGood.setInputFormat(train);
```

```
    Instances filteredTrain = Filter.useFilter(train, selectGood);

    return filteredTrain;
  }

  public Instances getData() {
    return data;
  }

  public void setData(Instances data) {
    this.data = data;
  }

  public int getRuns() {
    return runs;
  }

  public void setRuns(int runs) {
    this.runs = runs;
  }

  public Attribute getOrientationAttribute() {
    return orientationAttribute;
  }

  public void setOrientationAttribute(Attribute orientationAttribute) {
    this.orientationAttribute = orientationAttribute;
  }

  public Attribute getSubjectiveQualityAttribute() {
    return subjectiveQualityAttribute;
  }

  public void setSubjectiveQualityAttribute(Attribute
subjectiveQualityAttribute) {
    this.subjectiveQualityAttribute = subjectiveQualityAttribute;
  }

}


/**********************************************
 * Name: QualityBasedOrientationClassifier
 * Author: Amanda De Hoedt
 * Description: Defines the structure of the
 *              quality-on-all classification
 *              scheme and performs
 *              classification
 **********************************************/
import java.util.Random;

import weka.classifiers.Classifier;
import weka.classifiers.Evaluation;
import weka.classifiers.bayes.NaiveBayes;
import weka.core.Instances;
import weka.filters.Filter;
import weka.filters.unsupervised.instance.RemoveWithValues;


public class QualityBasedOrientationClassifier extends ClassificationScheme{

public QualityBasedOrientationClassifier(Instances d, String s){
  super(d);
```

```
  System.out.print("Setting up quality-based orientation classification
scheme." + '\n');

  try {
    d = removeBinaryQuality("Good",d);
    d = removeBinaryQuality("Average",d);
    d = removeBinaryQuality("Poor",d);
  } catch (Exception e) {
    e.printStackTrace();
  }
  setData(new Instances(d));
  setSubjectiveQualityAttribute(d.attribute("Subjective"));
  setClassificationType(s);
}

public void run() throws Exception {
  System.out.print("***RUNNING QUALITY BASED ORIENTATION CLASSIFICATION SCHEME
WITH " + getClassificationType() + " classifier***" + '\n');

  //Define classifier
  Classifier cModel = getClassifier();

  //Run multiple times
  for(int r = 0; r < getRuns(); r++){
    //Print the run
    System.out.println("\n Run: " + (r+1));

    //Create cross-validation folds
    Random rand = new Random(r+1);
    int folds = 10;

    setSubjectiveQualityAttribute(getData().attribute("Subjective"));
    Instances randData = new Instances(getData());
    randData.randomize(rand);
    randData.stratify(folds);

    //for each quality level (good, average, poor)
    for (int m = 1; m <=3; m++){
      Evaluation eTest = new Evaluation(randData);
      for (int n = 0; n < folds; n++){
        //get training and testing sets
        Instances train = randData.trainCV(folds, n);
        Instances test = randData.testCV(folds, n);

        //filter train for images of certain quality
        //Remove the numerical quality attributes
        String[] optionsSelectGood = new String[4];
        optionsSelectGood[0] = "-C";  // "attribute"
        optionsSelectGood[1] = "1"; // first attribute (Subjective)
        optionsSelectGood[2] = "-L";  // "label"
        optionsSelectGood[3] = Integer.toString(m);

        RemoveWithValues selectGood = new RemoveWithValues();
        selectGood.setOptions(optionsSelectGood);
        selectGood.setInvertSelection(true);
        selectGood.setInputFormat(train);
        Instances filteredTrain = Filter.useFilter(train, selectGood);

        //uncomment line below to use all quality levels
        //Instances filteredTrain = new Instances(train);

        //Set class index
        filteredTrain.setClassIndex(1);
```

```
        //build classifier with filtered training set
            Classifier clsCopy = Classifier.makeCopy(cModel);
            clsCopy.buildClassifier(filteredTrain);

        //predict orientation using classifier
        eTest.evaluateModel(clsCopy, test);

        if(n == 9){
          //print the quality name
          String title = getQualityFromInt(m);
          System.out.println("\n" + title + " quality run: " + (n+1));
          printResults(eTest);

        }
      }
    }
  }
}


}


/************************************************
 * Name: QualityClassifierOnQualityData
 * Author: Amanda De Hoedt
 * Description: Defines the structure of the
 *              quality-on-quality classification
 *              scheme and performs
 *              classification
 ************************************************/
import java.util.Random;

import weka.classifiers.Classifier;
import weka.classifiers.Evaluation;
import weka.classifiers.bayes.NaiveBayes;
import weka.core.Instances;


public class QualityClassifierOnQualityData extends ClassificationScheme{


  public QualityClassifierOnQualityData(Instances d, String s){
    super(d);
    System.out.print("Setting up quality classifier on quality data." + '\n');

    try {
      d = removeBinaryQuality("Good",d);
      d = removeBinaryQuality("Average",d);
      d = removeBinaryQuality("Poor",d);
    } catch (Exception e) {
      e.printStackTrace();
    }
    setData(new Instances(d));
    setSubjectiveQualityAttribute(d.attribute("Subjective"));
    setOrientationAttribute(d.attribute("Type"));
    setClassificationType(s);
  }

  public void run() throws Exception {

    System.out.print("***RUNNING QUALITY CLASSIFIER ON QUALITY DATA
```

```
CLASSIFICATION SCHEME WITH " + getClassificationType() + " classifier***" +
'\n');

    //Define classifier
    //Classifier cModelQuality = (Classifier) new NaiveBayes();
    /*
    Classifier cModelOrientationGood = (Classifier) new NaiveBayes();
    Classifier cModelOrientationAve = (Classifier) new NaiveBayes();
    Classifier cModelOrientationPoor = (Classifier) new NaiveBayes();
    */
    Classifier cModelOrientationGood = getClassifier();
    Classifier cModelOrientationAve = getClassifier();
    Classifier cModelOrientationPoor = getClassifier();

    //Run multiple times
    for(int r = 0; r < getRuns(); r++){
      //Print the run
      System.out.println("\n Run: " + (r+1));

      //Create cross-validation folds
      //int seed = 4;
      Random rand = new Random(r+1);
      int folds = 10;

      Instances randData = new Instances(getData());
      randData.randomize(rand);
      randData.stratify(folds);

      //Define evaluators
      setSubjectiveQualityAttribute(getData().attribute("Subjective"));
      setOrientationAttribute(getData().attribute("Type"));
      randData.setClassIndex(getOrientationAttribute().index());
      Evaluation eTestOrientation = new Evaluation(randData);
      Evaluation eTestOrientationGood = new Evaluation(randData);
      Evaluation eTestOrientationAve = new Evaluation(randData);
      Evaluation eTestOrientationPoor = new Evaluation(randData);
      //randData.setClassIndex(subjectiveQualityAttribute.index());
      //Evaluation eTestQuality = new Evaluation(randData);

      for (int n = 0; n < folds; n++){
        //get training and testing sets
        Instances train = randData.trainCV(folds, n);
        Instances test = randData.testCV(folds, n);

        Instances orientationTesting = new Instances(test);

        setClassAttributes(train);

        //filter to train for images of certain quality
          Instances filteredTrainGood = filterOnQuality("good", train);
          Instances filteredTrainAve = filterOnQuality("average", train);
          Instances filteredTrainPoor = filterOnQuality("poor", train);

        //Set class index
          setClassAttributes(train);
          filteredTrainGood.setClassIndex(getOrientationAttribute().index());
          filteredTrainAve.setClassIndex(getOrientationAttribute().index());
          filteredTrainPoor.setClassIndex(getOrientationAttribute().index());

        //build classifier with filtered training set
          Classifier cModelOrientationGoodCopy =
Classifier.makeCopy(cModelOrientationGood);
          cModelOrientationGoodCopy.buildClassifier(filteredTrainGood);
```

```
          Classifier cModelOrientationAveCopy =
Classifier.makeCopy(cModelOrientationAve);
          cModelOrientationAveCopy.buildClassifier(filteredTrainAve);
          Classifier cModelOrientationPoorCopy =
Classifier.makeCopy(cModelOrientationPoor);
          cModelOrientationPoorCopy.buildClassifier(filteredTrainPoor);


        //filter to train for images of certain quality
          Instances filteredTestGood = filterOnQuality("good",
orientationTesting);
          Instances filteredTestAve = filterOnQuality("average",
orientationTesting);
          Instances filteredTestPoor = filterOnQuality("poor",
orientationTesting);

        //Set class index
          filteredTestGood.setClassIndex(getOrientationAttribute().index());
          filteredTestAve.setClassIndex(getOrientationAttribute().index());
          filteredTestPoor.setClassIndex(getOrientationAttribute().index());

        //predict quality using classifier
          //eTestQuality.evaluateModel(cModelQualityCopy, qualityTesting);
          eTestOrientation.evaluateModel(cModelOrientationGoodCopy,
filteredTestGood);
          eTestOrientationGood.evaluateModel(cModelOrientationGoodCopy,
filteredTestGood);
          eTestOrientation.evaluateModel(cModelOrientationAveCopy,
filteredTestAve);
          eTestOrientationAve.evaluateModel(cModelOrientationAveCopy,
filteredTestAve);
          eTestOrientation.evaluateModel(cModelOrientationPoorCopy,
filteredTestPoor);
          eTestOrientationPoor.evaluateModel(cModelOrientationPoorCopy,
filteredTestPoor);

        //based on quality prediction, use orientation classifier to predict
orientation

          //Print the Quality results
        //System.out.println("\n Quality run: " + (n+1));
        //printResults(eTestQuality);

          if(n == 9){
          //Print the Quality results
          System.out.println("\n Overall Run: " + (r+1) + " Orientation run: "
+ (n+1));
          printResults(eTestOrientation);

          System.out.println("\n Overall Run: " + (r+1) + " Orientation Good
run: " + (n+1));
          printResults(eTestOrientationGood);

          System.out.println("\n Overall Run: " + (r+1) + " Orientation Ave
run: " + (n+1));
          printResults(eTestOrientationAve);

          System.out.println("\n Overall Run: " + (r+1) + " Orientation Poor
run: " + (n+1));
          printResults(eTestOrientationPoor);
          }

      }
```

```
    }
  }

}



/*********************************************
 * Name: BasicClassifier
 * Author: Amanda De Hoedt
 * Description: Defines the structure of the basic
 *              classification scheme and
 *              performs classification
 *********************************************/
import java.util.Random;

import weka.classifiers.Classifier;
import weka.classifiers.Evaluation;
import weka.classifiers.bayes.NaiveBayes;
import weka.classifiers.functions.LinearRegression;
import weka.classifiers.functions.Logistic;
import weka.classifiers.functions.MultilayerPerceptron;
import weka.classifiers.functions.SMO;
import weka.classifiers.functions.VotedPerceptron;
import weka.classifiers.lazy.IBk;
import weka.classifiers.trees.J48;
import weka.classifiers.trees.RandomForest;
import weka.core.Instances;
import weka.filters.Filter;
import weka.filters.unsupervised.attribute.Remove;
import weka.filters.unsupervised.instance.RemoveWithValues;
import weka.core.Attribute;

public class BasicClassifier extends ClassificationScheme{

  public BasicClassifier(Instances d, String s){
    super(d);
    System.out.print("Setting up basic classification scheme." + '\n');

    //filter unneeded binary quality columns
    try {
      d = removeBinaryQuality("Good",d);
      d = removeBinaryQuality("Average",d);
      d = removeBinaryQuality("Poor",d);
    } catch (Exception e) {
      e.printStackTrace();
    }
    setData(new Instances(d));
    setClassAttributes(d);
    setClassificationType(s);
  }

  public void run() throws Exception{
    System.out.print("***RUNNING BASIC CLASSIFICATION SCHEME WITH " +
getClassificationType() + " classifier***" + '\n');

    //Define classifier
    Classifier cModelOrientation = getClassifier();
    Classifier cModelQuality = getClassifier();

    //Run multiple times
    for(int r = 0; r < getRuns(); r++){
```

```
        //Print the run
        System.out.println("\n Run: " + (r+1));

        //Create cross-validation folds
        //int seed = 4;
        Random rand = new Random(r+1);
        int folds = 10;

        Instances randData = new Instances(getData());
        randData.randomize(rand);
        randData.stratify(folds);

        setClassAttributes(randData);
        randData.setClassIndex(getOrientationAttribute().index());
        Evaluation eTestOrientation = new Evaluation(randData);
        randData.setClassIndex(getSubjectiveQualityAttribute().index());
        Evaluation eTestQuality = new Evaluation(randData);

        //For each fold
        for (int n = 0; n < folds; n++){

          //get training and testing sets
          Instances train = randData.trainCV(folds, n);
          Instances test = randData.testCV(folds, n);

          setClassAttributes(train);

          //Filter to remove the subjective quality attributes
          Remove removeQuality = new Remove();
          String[] qualityOptions = new String[2];
          qualityOptions[0] = "-R";  // "range"
          qualityOptions[1] =
String.valueOf(getSubjectiveQualityAttribute().index()+1); //range of
attributes
          removeQuality.setOptions(qualityOptions);
          removeQuality.setInputFormat(getData());

          //Filter to remove the orientation attributes
          Remove removeOrientation = new Remove();
          String[] orientationOptions = new String[2];
          orientationOptions[0] = "-R";  // "range"
          orientationOptions[1] =
String.valueOf(getOrientationAttribute().index()+1); //range of attributes
          removeOrientation.setOptions(orientationOptions);
          removeOrientation.setInputFormat(getData());

          //Apply the filters
          Instances orientationTraining = Filter.useFilter(train, removeQuality);
          Instances orientationTesting = Filter.useFilter(test, removeQuality);
          Instances qualityTraining = Filter.useFilter(train, removeOrientation);
          Instances qualityTesting = Filter.useFilter(test, removeOrientation);

          //Set class index
          setClassAttributes(qualityTraining, orientationTraining);

          orientationTraining.setClassIndex(getOrientationAttribute().index());
          orientationTesting.setClassIndex(getOrientationAttribute().index());
          qualityTraining.setClassIndex(getSubjectiveQualityAttribute().index());
          qualityTesting.setClassIndex(getSubjectiveQualityAttribute().index());

          //Build classifier with filtered training set
            Classifier cModelOrientationCopy =
Classifier.makeCopy(cModelOrientation);
```

```
          Classifier cModelQualityCopy = Classifier.makeCopy(cModelQuality);

          cModelOrientationCopy.buildClassifier(orientationTraining);
          cModelQualityCopy.buildClassifier(qualityTraining);

        //Predict orientation using classifier
          eTestOrientation.evaluateModel(cModelOrientationCopy,
orientationTesting);
          eTestQuality.evaluateModel(cModelQualityCopy, qualityTesting);

          //Print the Orientation results
          if(n==9){
            System.out.println("\n Orientation run: " + (n+1));
          printResults(eTestOrientation);
          }

        //Print the Quality results
        if(n==9){
          System.out.println("\n Quality run: " + (n+1));
          printResults(eTestQuality);
        }

      }
    }

  }//end function run

}



/**********************************************
 * Name: QualityBasedClassifier
 * Author: Amanda De Hoedt
 * Description: Defines the structure of the
 *              quality-based classification
 *              scheme and performs
 *              classification
 **********************************************/
import java.util.Random;

import weka.classifiers.Classifier;
import weka.classifiers.Evaluation;
import weka.classifiers.bayes.NaiveBayes;
import weka.core.Instance;
import weka.core.Instances;
import weka.filters.Filter;
import weka.filters.unsupervised.attribute.Remove;


public class QualityBasedClassifier extends ClassificationScheme{

  public QualityBasedClassifier(Instances d, String s){
    super(d);
    System.out.print("Setting up quality-based classification scheme." + '\n');
    try {
      d = removeBinaryQuality("Good",d);
      d = removeBinaryQuality("Average",d);
      d = removeBinaryQuality("Poor",d);
    } catch (Exception e) {
      e.printStackTrace();
    }
    setData(new Instances(d));
```

```
    setSubjectiveQualityAttribute(d.attribute("Subjective"));
    setOrientationAttribute(d.attribute("Type"));
    setClassificationType(s);
  }

  public void run() throws Exception {

    System.out.print("***RUNNING QUALITY-BASED CLASSIFICATION SCHEME WITH " +
getClassificationType() + " classifier***" + '\n');

    //Define classifier
    Classifier cModelQuality = getClassifier();
    Classifier cModelOrientationGood = getClassifier();
    Classifier cModelOrientationAve = getClassifier();
    Classifier cModelOrientationPoor = getClassifier();

    //Run multiple times
    for(int r = 0; r < getRuns(); r++){
      //Print the run
      System.out.println("\n Run: " + (r+1));

      //Create cross-validation folds
      //int seed = 4;
      Random rand = new Random(r+1);
      int folds = 10;

      setSubjectiveQualityAttribute(getData().attribute("Subjective"));
      setOrientationAttribute(getData().attribute("Type"));
      Instances randData = new Instances(getData());
      randData.randomize(rand);
      randData.stratify(folds);

      //Define evaluators
      randData.setClassIndex(getOrientationAttribute().index());
      Evaluation eTestOrientation = new Evaluation(randData);
      Evaluation eTestOrientationGood = new Evaluation(randData);
      Evaluation eTestOrientationAve = new Evaluation(randData);
      Evaluation eTestOrientationPoor = new Evaluation(randData);
      randData.setClassIndex(getSubjectiveQualityAttribute().index());
      Evaluation eTestQuality = new Evaluation(randData);

      for (int n = 0; n < folds; n++){
        //get training and testing sets
        Instances train = randData.trainCV(folds, n);
        Instances test = randData.testCV(folds, n);

        Instances orientationTesting = new Instances(test);

        setClassAttributes(train);

        //Filter to remove the orientation attributes
        Remove removeOrientation = new Remove();
        String[] orientationOptions = new String[2];
        orientationOptions[0] = "-R";  // "range"
        orientationOptions[1] =
String.valueOf(getOrientationAttribute().index()+1); //range of attributes
        removeOrientation.setOptions(orientationOptions);
        removeOrientation.setInputFormat(getData());

        //Apply the filters
        Instances qualityTraining = Filter.useFilter(train, removeOrientation);
        Instances qualityTesting = Filter.useFilter(test, removeOrientation);
```

```
        //Set class index
        setClassAttributes(qualityTraining);
        qualityTraining.setClassIndex(getSubjectiveQualityAttribute().index());
        qualityTesting.setClassIndex(getSubjectiveQualityAttribute().index());

        //Build classifier with filtered training set
          Classifier cModelQualityCopy = Classifier.makeCopy(cModelQuality);
          cModelQualityCopy.buildClassifier(qualityTraining);

        //filter to train for images of certain quality
          Instances filteredTrainGood = filterOnQuality("good", train);
          Instances filteredTrainAve = filterOnQuality("average", train);
          Instances filteredTrainPoor = filterOnQuality("poor", train);

        //Set class index
          setClassAttributes(train);
          filteredTrainGood.setClassIndex(getOrientationAttribute().index());
          filteredTrainAve.setClassIndex(getOrientationAttribute().index());
          filteredTrainPoor.setClassIndex(getOrientationAttribute().index());

        //build classifier with filtered training set
          Classifier cModelOrientationGoodCopy =
Classifier.makeCopy(cModelOrientationGood);
          cModelOrientationGoodCopy.buildClassifier(filteredTrainGood);
          Classifier cModelOrientationAveCopy =
Classifier.makeCopy(cModelOrientationAve);
          cModelOrientationAveCopy.buildClassifier(filteredTrainAve);
          Classifier cModelOrientationPoorCopy =
Classifier.makeCopy(cModelOrientationPoor);
          cModelOrientationPoorCopy.buildClassifier(filteredTrainPoor);


          setClassAttributes(orientationTesting);
          //for each instance of the testing set, get the quality prediction
          //System.out.println("\n Total Number of instances: " +
qualityTesting.numInstances());
          for(int i = 0; i < qualityTesting.numInstances(); i++){
            Instance inst = qualityTesting.instance(i);
            double result = cModelQualityCopy.classifyInstance(inst);
            //System.out.println("\n Result " + i + " : " + result);

            //set value of corresponding instance in orientation test set

orientationTesting.instance(i).setValue(getSubjectiveQualityAttribute().index()
, result);
          }

        //filter to train for images of certain quality
          Instances filteredTestGood = filterOnQuality("good",
orientationTesting);
          Instances filteredTestAve = filterOnQuality("average",
orientationTesting);
          Instances filteredTestPoor = filterOnQuality("poor",
orientationTesting);

          //Set class index
          filteredTestGood.setClassIndex(getOrientationAttribute().index());
          filteredTestAve.setClassIndex(getOrientationAttribute().index());
          filteredTestPoor.setClassIndex(getOrientationAttribute().index());

        //predict quality using classifier
          eTestQuality.evaluateModel(cModelQualityCopy, qualityTesting);
          eTestOrientation.evaluateModel(cModelOrientationGoodCopy,
```

```
filteredTestGood);
          eTestOrientationGood.evaluateModel(cModelOrientationGoodCopy,
filteredTestGood);
          eTestOrientation.evaluateModel(cModelOrientationAveCopy,
filteredTestAve);
          eTestOrientationAve.evaluateModel(cModelOrientationAveCopy,
filteredTestAve);
          eTestOrientation.evaluateModel(cModelOrientationPoorCopy,
filteredTestPoor);
          eTestOrientationPoor.evaluateModel(cModelOrientationPoorCopy,
filteredTestPoor);

        //based on quality prediction, use orientation classifier to predict
orientation

          //Print the Quality results
          if(n == 9)
          {
          System.out.println("\n Quality run: " + (n+1));
          printResults(eTestQuality);

          //Print the Quality results
          System.out.println("\n Orientation run: " + (n+1));
          printResults(eTestOrientation);

          System.out.println("\n Orientation Good run: " + (n+1));
          printResults(eTestOrientationGood);

          System.out.println("\n Orientation Ave run: " + (n+1));
          printResults(eTestOrientationAve);

          System.out.println("\n Orientation Poor run: " + (n+1));
          printResults(eTestOrientationPoor);
          }

      }
    }
  }

}


/**********************************************
 * Name: VotingClassifier
 * Author: Amanda De Hoedt
 * Description: Defines the structure of the
 *              voting classification scheme and
 *              performs classification
 **********************************************/
import java.util.Random;

import weka.classifiers.Classifier;
import weka.classifiers.Evaluation;
import weka.classifiers.bayes.NaiveBayes;
import weka.core.Attribute;
import weka.core.Instance;
import weka.core.Instances;
import weka.filters.Filter;
import weka.filters.unsupervised.attribute.Remove;


public class VotingClassifier extends ClassificationScheme{
```

```
  public VotingClassifier(Instances d, String s){
    super(d);
    System.out.print("Setting up voting classification scheme." + '\n');
    setSubjectiveQualityAttribute(d.attribute("Subjective"));
    setOrientationAttribute(d.attribute("Type"));
    setData(new Instances(d));
    setClassificationType(s);
  }

public void run() throws Exception {

  System.out.print("***RUNNING VOTING CLASSIFICATION SCHEME WITH " +
getClassificationType() + " classifier***" + '\n');

  //Define classifiers
  Classifier cModelQuality = getClassifier();
  Classifier cModelQualityGood = getClassifier();
  Classifier cModelQualityAve = getClassifier();
  Classifier cModelQualityPoor = getClassifier();
  Classifier cModelOrientationGood = getClassifier();
  Classifier cModelOrientationAve = getClassifier();
  Classifier cModelOrientationPoor = getClassifier();

  //Run multiple times
  for(int r = 0; r < getRuns(); r++){
    //Print the run
    System.out.println("\n Run: " + (r+1));

    //Create cross-validation folds
    Random rand = new Random(r+1);
    int folds = 10;

    setSubjectiveQualityAttribute(getData().attribute("Subjective"));
    setOrientationAttribute(getData().attribute("Type"));
    Instances randData = new Instances(getData());
    randData.randomize(rand);
    randData.stratify(folds);

    //Define evaluators
    randData.setClassIndex(getOrientationAttribute().index());
    Evaluation eTestOrientation = new Evaluation(randData);

    randData.setClassIndex(getSubjectiveQualityAttribute().index());
    Evaluation eTestQuality = new Evaluation(randData);

    Attribute binaryQualityAttribute = randData.attribute("Good");
    randData.setClassIndex(binaryQualityAttribute.index());
    Evaluation eTestBinaryQuality = new Evaluation(randData);

    for (int n = 0; n < folds; n++){
      //get training and testing sets
      Instances train = randData.trainCV(folds, n);
      Instances test = randData.testCV(folds, n);

      Instances orientationTraining = removeAllBinaryQuality(train);
      Instances orientationTesting = removeAllBinaryQuality(test);

      Instances votingQualityTraining = removeAllBinaryQuality(train);
      votingQualityTraining =
removeOrientationAttribute(votingQualityTraining);
      Instances votingQualityTesting = removeAllBinaryQuality(test);
      votingQualityTesting = removeOrientationAttribute(votingQualityTesting);
      setClassAttributes(votingQualityTraining);
```

```
votingQualityTraining.setClassIndex(getSubjectiveQualityAttribute().index());

      setClassAttributes(train);

      //MAKE THREE QUALITY CLASSIFIERS
      Instances qualityGoodTrain = getFilteredQualitySet("Good", train);
      Instances qualityAveTrain = getFilteredQualitySet("Average", train);
      Instances qualityPoorTrain = getFilteredQualitySet("Poor", train);

      Instances qualityGoodTest = getFilteredQualitySet("Good", test);
      Instances qualityAveTest = getFilteredQualitySet("Average", test);
      Instances qualityPoorTest = getFilteredQualitySet("Poor", test);

      //Build classifiers with filtered training set
      Classifier cModelQualityCopy = Classifier.makeCopy(cModelQuality);
        Classifier cModelGoodQualityCopy =
Classifier.makeCopy(cModelQualityGood);
        Classifier cModelAveQualityCopy =
Classifier.makeCopy(cModelQualityAve);
        Classifier cModelPoorQualityCopy =
Classifier.makeCopy(cModelQualityPoor);
        cModelQualityCopy.buildClassifier(votingQualityTraining);
        cModelGoodQualityCopy.buildClassifier(qualityGoodTrain);
        cModelAveQualityCopy.buildClassifier(qualityAveTrain);
        cModelPoorQualityCopy.buildClassifier(qualityPoorTrain);

        //MAKE THREE ORIENTATION CLASSIFIERS
      //filter to train for images of certain quality
        Instances filteredTrainGood = filterOnQuality("good",
orientationTraining);
        Instances filteredTrainAve = filterOnQuality("average",
orientationTraining);
        Instances filteredTrainPoor = filterOnQuality("poor",
orientationTraining);

      //Set class index
        setClassAttributes(orientationTraining);
        filteredTrainGood.setClassIndex(getOrientationAttribute().index());
        filteredTrainAve.setClassIndex(getOrientationAttribute().index());
        filteredTrainPoor.setClassIndex(getOrientationAttribute().index());

      //build classifier with filtered training set
        Classifier cModelOrientationGoodCopy =
Classifier.makeCopy(cModelOrientationGood);
        Classifier cModelOrientationAveCopy =
Classifier.makeCopy(cModelOrientationAve);
        Classifier cModelOrientationPoorCopy =
Classifier.makeCopy(cModelOrientationPoor);
        cModelOrientationGoodCopy.buildClassifier(filteredTrainGood);
        cModelOrientationAveCopy.buildClassifier(filteredTrainAve);
        cModelOrientationPoorCopy.buildClassifier(filteredTrainPoor);

        setClassAttributes(orientationTesting);
        //for each instance of the testing set, get the quality prediction
        //System.out.println("\n Total Number of instances: " +
test.numInstances());
        for(int i = 0; i < test.numInstances(); i++){

          Instance instG = qualityGoodTest.instance(i);
          Instance instA = qualityAveTest.instance(i);
          Instance instP = qualityPoorTest.instance(i);
```

```
          //Get scores from good, average, and poor quality classifiers
          double resultG = cModelGoodQualityCopy.classifyInstance(instG);
          double resultA = cModelAveQualityCopy.classifyInstance(instA);
          double resultP = cModelPoorQualityCopy.classifyInstance(instP);

          //get quality prediction from vote
          String result = getVotingResults(resultP, resultA, resultG);
          //System.out.println("\n Result " + i + ": " + "G: " + resultG + " A:
" + resultA + " P: " + resultP + " Vote: " + result);

          double vote = (double)getIntFromVoteQuality(result);

          //set value of corresponding instance in orientation test set
          setClassAttributes(orientationTesting);

orientationTesting.instance(i).setValue(getSubjectiveQualityAttribute().index()
, vote);

          //set value of corresponding instance in quality test set
          setClassAttributes(votingQualityTesting);

votingQualityTesting.instance(i).setValue(getSubjectiveQualityAttribute().index
(), vote);
        }

      //filter to train for images of certain quality
        Instances filteredTestGood = filterOnQuality("good",
orientationTesting);
        Instances filteredTestAve = filterOnQuality("average",
orientationTesting);
        Instances filteredTestPoor = filterOnQuality("poor",
orientationTesting);

      //Set class index
        setClassAttributes(orientationTesting);
        filteredTestGood.setClassIndex(getOrientationAttribute().index());
        filteredTestAve.setClassIndex(getOrientationAttribute().index());
        filteredTestPoor.setClassIndex(getOrientationAttribute().index());

        setClassAttributes(votingQualityTesting);

votingQualityTesting.setClassIndex(getSubjectiveQualityAttribute().index());

      //predict quality using classifier
        eTestQuality.evaluateModel(cModelQualityCopy, votingQualityTesting);
        eTestBinaryQuality.evaluateModel(cModelGoodQualityCopy,
qualityGoodTest);
        eTestBinaryQuality.evaluateModel(cModelAveQualityCopy, qualityAveTest);
        eTestBinaryQuality.evaluateModel(cModelPoorQualityCopy,
qualityPoorTest);
        eTestOrientation.evaluateModel(cModelOrientationGoodCopy,
filteredTestGood);
        eTestOrientation.evaluateModel(cModelOrientationAveCopy,
filteredTestAve);
        eTestOrientation.evaluateModel(cModelOrientationPoorCopy,
filteredTestPoor);

      //based on quality prediction, use orientation classifier to predict
orientation

        if(n == 9){
          //Print the Quality results
        System.out.println("\n Binary Quality run: " + (n+1));
```

```
        printResults(eTestBinaryQuality);

        //Print the Quality results
        System.out.println("\n Subjective Quality run: " + (n+1));
        printResults(eTestQuality);

        //Print the Quality results
        System.out.println("\n Orientation run: " + (n+1));
        printResults(eTestOrientation);
        }

    }
  }
}

private static String getVotingResults(double resultP, double resultA, double
resultG){
  int pWeight = 1;
  int aWeight = 2;
  int gWeight = 3;
  double result;

  double num = (resultP * pWeight) + (resultA * aWeight) + (resultG * gWeight);
  double denom = resultP + resultA + resultG;

  if(denom == 0)
  {
    result = 1;
  }
  else
  {
    result = Math.floor(num/denom);
  }

  if(result == 1)
  {
    return "poor";
  }
  else if(result == 2)
  {
    return "average";
  }
  else if(result == 3)
  {
    return "good";
  }
  else
  {
    return "void";
  }
}

  private static Integer getIntFromVoteQuality(String s){
    if(s == "average"){
      return 0;
    }
    else if(s == "poor"){
      return 1;
    }
    else if(s == "good"){
      return 2;
    }
    return null;
```

```
  } //end getQualityFromInt

  private Instances getFilteredQualitySet(String q, Instances train) throws
Exception{
    String[] qualities = new String[3];
    qualities[0] = "Good";
    qualities[1] = "Average";
    qualities[2] = "Poor";

    train = removeOrientationAttribute(train);
    train = removeSubjectiveQualityAttribute(train);

    //Remove binary values for other qualities
    for(int i = 0; i < qualities.length; i++)
    {
      if(qualities[i] != q)
      {
        train = removeBinaryQuality(qualities[i],train);
      }
    }

    //Set class index
    Attribute binaryAttribute = train.attribute(q);
    train.setClassIndex(binaryAttribute.index());

    return train;
  }


  private Instances removeOrientationAttribute(Instances d) throws Exception{
    setOrientationAttribute(d.attribute("Type"));

    //Filter to remove the orientation attributes
    Remove removeOrientation = new Remove();
    String[] orientationOptions = new String[2];
    orientationOptions[0] = "-R";  // "range"
    orientationOptions[1] =
String.valueOf(getOrientationAttribute().index()+1); //range of attributes
    removeOrientation.setOptions(orientationOptions);
    removeOrientation.setInputFormat(d);

    //Apply the filters
    Instances filtered = Filter.useFilter(d, removeOrientation);

    return filtered;
  }

  private Instances removeSubjectiveQualityAttribute(Instances d) throws
Exception{
    setSubjectiveQualityAttribute(d.attribute("Subjective"));

    //Filter to remove the orientation attributes
    Remove removeOrientation = new Remove();
    String[] orientationOptions = new String[2];
    orientationOptions[0] = "-R";  // "range"
    orientationOptions[1] =
String.valueOf(getSubjectiveQualityAttribute().index()+1); //range of
attributes
    removeOrientation.setOptions(orientationOptions);
    removeOrientation.setInputFormat(d);

    //Apply the filters
    Instances filtered = Filter.useFilter(d, removeOrientation);
```

```
    return filtered;
  }

  private Instances removeAllBinaryQuality(Instances d) throws Exception{
      d = removeBinaryQuality("Good",d);
      d = removeBinaryQuality("Average",d);
      d = removeBinaryQuality("Poor",d);
      return d;
  }

}
```

REFERENCES

1. Foster A. & N. Davis. 2007. Congenital talipes equinovarus (clubfoot). Surgery (Oxford). 25: 171-175.

2. Penny J. N. 2005. The neglected clubfoot. Techniques in orthopaedics. 20: 153-166.

3. Dobbs M. B. & C. A. Gurnett. 2009. Update on clubfoot: etiology and treatment. Clin. Orthop. 467: 1146-1153.

4. Urdea M., L. A. Penny, S. S. Olmsted *et al.* 2006. Requirements for high impact diagnostics in the developing world. Nature. 444: 73-79.

5. Wootton R. 1997. The possible use of telemedicine in developing countries. J. Telemed. Telecare. 3: 23-26.

6. Coordinated Laboratory for Computational Genomics. 2013. International Clubfoot Registry.

7. Wynne-Davies R. 1972. Genetic and environmental factors in the etiology of talipes equinovarus. Clin. Orthop. 84: 9-13.

8. Ponseti I. V. (. 1996. Congenital clubfoot : fundamentals of treatment / Ignacio V. Ponseti.

9. miraclefeet. 2013. miraclefeet.

10. Ponseti International Association. 2013. The Ponseti International Association.

11. International Telecommunication Union. 2013. The World in 2013: ICT Facts and Figures. ITU.

12. The MathWorks. 2012. Matlab. R2012a (7.14.0.739).

13. Vedaldi A. & B. Fulkerson. 2010. VLFeat: An open and portable library of computer vision algorithms. *In* Proceedings of the International Conference on Multimedia. : 1469-1472. ACM.

14. Lowe D. G. 1999. Object recognition from local scale-invariant features. *In* Computer Vision, 1999. the Proceedings of the Seventh IEEE International Conference On. Vol. 2: 1150-1157. Ieee.

15. Lowe D. G., inventor; The University Of British Columbia, assignee. 2004. Method and Apparatus for Identifying Scale Invariant Features in an Image and use of Same for Locating an Object in an Image.

16. Niu P., X. Wang, H. Jin *et al.* 2011. A feature-based robust digital image watermarking scheme using bandelet transform. Optics & Laser Technology. 43: 437-450.

17. Lloyd S. 1982. Least squares quantization in PCM. Information Theory, IEEE Transactions on. 28: 129-137.

18. Elkan C. 2003. Using the triangle inequality to accelerate k-means. *In* Machine Learning-International Workshop then Conference. Vol. 20: 147.

19. Vedaldi A. & B. Fulkerson. 2008. VLFeat: An open and portable library of computer vision algorithms. VLFeat: An open and portable library of computer vision algorithms (2008).

20. Sivic J. & A. Zisserman. 2009. Efficient visual search of videos cast as text retrieval. Pattern Analysis and Machine Intelligence, IEEE Transactions on. 31: 591-606.

21. Nowak E., F. Jurie & B. Triggs. 2006. Sampling strategies for bag-of-features image classification. *In* Computer Vision–ECCV 2006. Anonymous : 490-503. Springer.

22. Fei-Fei L. 2010. Part 1: Bag-of-words models. : 1-69.

23. Bosch A., A. Zisserman & X. Muoz. 2007. Image classification using random forests and ferns. *In* Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference On. : 1-8. IEEE.

24. Moorthy A. & A. Bovik. 2009. A modular framework for constructing blind universal quality indices. IEEE Signal Process. Lett.

25. Davis L. S. 1975. A survey of edge detection techniques. Computer graphics and image processing. 4: 248-270.

26. Maini R. & H. Aggarwal. 2009. Study and comparison of various image edge detection techniques. International Journal of Image Processing (IJIP). 3: 1-11.

27. Stegmann M. B. & D. D. Gomez. 2002. A brief introduction to statistical shape analysis. Informatics and Mathematical Modelling, Technical University of Denmark, DTU. : 15.

28. The University of Waikato. 2010. Waikato Environment for Knowledge Analysis.

29. Street N. 2011. Knowledge Discovery: Machine Learning.

30. Lewis D. D. 1998. Naive (Bayes) at forty: The independence assumption in information retrieval. *In* Machine Learning: ECML-98. Anonymous : 4-15. Springer.

31. John G. H. & P. Langley. 1995. Estimating continuous distributions in Bayesian classifiers. *In* Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence. : 338-345. Morgan Kaufmann Publishers Inc.

32. Cortes C. & V. Vapnik. 1995. Support-vector networks. Mach. Learning. 20: 273-297.

33. Smola A. J. & B. Schölkopf. 2004. A tutorial on support vector regression. Statistics and computing. 14: 199-222.

34. Platt J. 1998. Sequential minimal optimization: A fast algorithm for training support vector machines.

35. Vailaya A., A. Jain & H. J. Zhang. 1998. On image classification: City images vs. landscapes. Pattern Recognit. 31: 1921-1935.

36. Huang J., S. R. Kumar & R. Zabih. 1998. An automatic hierarchical image classification scheme. *In* Proceedings of the Sixth ACM International Conference on Multimedia. : 219-228. ACM.

37. Sablatnig R., P. Kammerer & E. Zolda. 1998. Hierarchical classification of paintings using face-and brush stroke models. *In* Pattern Recognition, 1998. Proceedings. Fourteenth International Conference On. Vol. 1: 172-174. IEEE.

38. Street N. 2011. Knowledge Discovery: Training and Testing Methods.

39. Guillaume FOUET. 2013. VisiPics. 1.31.

40. Mukaka M. 2012. A guide to appropriate use of Correlation coefficient in medical research. Malawi Medical Journal. 24: 69-71.

41. Moorthy A. & A. Bovik. 2009. BIQI software release. URL {http://live.ece.utexas.edu/research/quality/biqi.zip}.

42. Quinlan J. R. 1993. C4. 5: Programs for Machine Learning. Morgan Kaufmann.

43. Aha D. W., D. Kibler & M. K. Albert. 1991. Instance-based learning algorithms. Mach. Learning. 6: 37-66.

44. Le Cessie S. & J. Van Houwelingen. 1992. Ridge estimators in logistic regression. Applied statistics. : 191-201.

45. Breiman L. 2001. Random forests. Mach. Learning. 45: 5-32.

46. Cohen W. W. 1995. Fast effective rule induction. *In* Machine Learning-International Workshop then Conference. : 115-123. Morgan Kaufmann Publishers, Inc.