**MODIFIED NAIVE BAYES ALGORITHM TO ANALYZE AND CLASSIFY**

**EMOTIONS MAPPED TO MUSIC DATA**

by

COREY RYAN MCINTOSH

B.S., University of Colorado, 2010

A thesis submitted to the Graduate Faculty of the

University of Colorado at Colorado Springs

in partial fulfillment of the

requirements for the degree of

Master of Science

Department of Computer Science

2016

© Copyright By Corey Ryan McIntosh 2016

All Rights Reserved

This thesis for the Master of Science degree by

Corey Ryan McIntosh

has been approved for the

|  |
| --- |
| Date |

Department of Computer Science

by

|  |
| --- |
| Sudhanshu K. Semwal, Chair |

|  |
| --- |
| Jonathan Ventura |

|  |
| --- |
| Glen Whitehead |

McIntosh, Corey Ryan (M.S., Computer Science)

Modified Naive Bayes Algorithm to Analyze and

Classify Emotions Mapped to Music Data

Thesis directed by Professor Sudhanshu K. Semwal

Emotions and the analysis of what produces them are a field that is largely relegated to psychologists and it is very hard to predict how someone will emote or react to music. Emotions are very similar to attraction. Attraction may be very hard for people to describe exactly what characteristics make them attracted to another person, but most definitely one knows it when they are attracted to another person. In the same way, people tend to be able to define the emotion that they are feeling, but they typically don’t know how they arrived there. This uncertainty lends itself to a very difficult undertaking with regard to our study: the emotional mapping of music. This focus is an important topic in today’s society, and an important part of MIR (Music Information Retrieval). Companies have relied almost exclusively on professionally trained people to determine genre, mood, and emotion of songs. Pandora, an industry leader, providing Internet radio, still to this day, utilizes this technique despite technological advances. As computers have become more efficient and can now handle millions of iterations with ease, several algorithms have emerged to provide automatic classification. Even though multiple different techniques have been tried, there is no consensus as to which one of them is the most useful and accurate. There is even some difficulty in determining what exactly should be predicted. In our study, a predictive approximation algorithm utilizing a modified Naive Bayes model based on statistical methodologies provides an alternative to the current algorithms for determining the emotion that music will elicit. The accuracy with which these traits map to emotions is compared against data from other research in the area to determine the validity and usefulness of the model proposed in this thesis. Ultimately, our implementation shows that the topic of emotion classification for music remains unresolved, with no resolution as to the best technique to use.

**TABLE OF CONTENTS**

CHAPTER

INTRODUCTION 1

Purpose of the Study 2

The Study 2

Data limitations 3

Other limitations of this Thesis 6

Necessary Mathematics and Algorithms 8

Ensemble Methods 8

Probabilistic Graph Models 9

Naïve Bayes Model 10

One-Way Anova Statistic 13

REVIEW OF THE LITERATURE AND RELATED WORK 15

Classifiers -- Review 15

Supervised Learning Algorithms Research 18

Emotional Mappings Research 20

METHODOLOGY 22

Utilized Software 22

Testing 26

Test Types 26

Test Setup 27

Test Algorithms 28

Process Details -- Study Procedure 28

Modified Naïve Bayes Classifier 29

RESULTS, DISCUSSION AND FURTHER RESEARCH 31

Findings 31

Discussion 34

Conclusion 38

BIBLIOGRAPHY/REFERENCES 40

A. Modified Naïve Bayes Classifier 42

B. Node.js Conversion Script 79

**TABLES**

Table

**1.** Accuracy by Classifier with four classes 31

2. Accuracy by Classifier with six classes 31

3. Accuracy by Classifier with nine classes 32

4. Soybean Classification 36

**FIGURES**

Figure

1. Directed Graph Model 9

2. Naïve Bayes Probability Tree 11

3. F-Critical Region 14

4. Linear SVM 17

5. Weka GUI Choices 24

6. Weka Explorer Preprocess 24

7. Percentile Chart 30

8. Output for Weka Experimenter Analysis for nine classes 33

9. Output for Weka Experimenter Analysis for six classes 33

10. Output for Weka Experimenter Analysis for four classes 34

**CHAPTER 1**

**INTRODUCTION**

Julian Treasure said, “The fascinating thing about music is that you recognize it fast and associate it fast[[1]](#footnote-1).” The fact is that the brain acts as an amazing classifier for within seconds it can determine the tempo, dynamics, genre, and the emotion that the music is trying to portray. While computers have been gaining a lot of ground in the area of machine learning and attempting to somewhat mimics the human brain, there is still a long way to go when competing with the brain. In the world of machine learning, there are a wide variety of classifiers that have been used in many number of fields to analyze data and produce a class declaration for that data. Some common classifier algorithms or supervised learning algorithms are: Naïve Bayes, Random Forest, Bagged Trees, Linear SVM, and Decision Trees. These algorithms typically ingest training data to build a model that can then effectively classify a future dataset or instance. Some of these algorithms work better on certain datasets and worse on the others. No *one* algorithm reigns supreme for all datasets. With classifiers, there are many factors to take into consideration when deciding which algorithm is superior to the other: time to generate the model, complexity, accuracy, versatility, and several more. The trick is determining what tradeoffs can be allowed with the specific application. There is a lot of research involved in each of these algorithms that will be discussed briefly in this study. Supervised learning algorithms have been developed over the last sixty years. This study attempts to further improve and investigate the work done by these algorithms to better interpret the emotional mapping to music that people experience everyday.

**Purpose of this Study**

This study was undertaken to better realize the impact music has on people. Without music or sound, there would not be the epic crescendos of stirring movies or the quiet oscillating tension of horror films. A conductor holds captive the very emotions of the audience throughout a movie, play, or opera. It is very fascinating an idea to contemplate -- while it is extremely difficult to fully understand what people were feeling emotionally when they first heard the Beatles at a concert or when the intro to Star Wars was first played, this study attempts to identify specific attributed that trigger specific emotional responses. The thesis tries to capture this complex mapping between the two.

**The Study**

This study uses 400 annotated music samples provided by Aljanaki, Wiering, and Veltkamp converting the attributes of these samples to an ARFF format that can then be ingested into Weka, a data mining and analysis tool[[2]](#footnote-2). All of the analysis and comparisons are done utilizing this tool. Weka allows the user to create classifiers that it will then use to generate a model from training data, and apply that model to the data used for testing the accuracy of the classifier.

**Data Limitations.** As specified in the scope above, the data is limited to 400 annotated music samples that were annotated while playing an online game and then after the game users were asked to rate the music induced emotion which the music generated per the GEMS scale (Genova Emotional Music Scale). The GEMS scale represents a select handful of the emotions that a person can feel when listening to music. The nine GEMS scale emotions are: Amazement, Solemnity, Tenderness, Nostalgia, Calmness, Power, Joyful activation, Tension, and Sadness. In addition, the study allowed for the samples to be annotated with multiple emotions per sample, if the sample generated multiple emotions. In out study, and to standardize the dataset, we totaled all the responses for a given sample and took the response that was most frequent as the presumed emotional response. Although this reduced the complexity and fuzziness of the data set which allowed us to classify one emotion to one music sample, there is an inherent limitation with our implemented methodology because samples can often be closely tied to multiple emotions and our study only considers, the single most relevant emotion, based on the data provided. For example, a sample may have made 20 people feel Tension and 18 people feel Sadness and it would be categorized the same as 37 people feeling Tension and 1 feeling Sadness. This lends itself to analytical biasing of the data, yet; unfortunately, this is a consequence of the tools that are being used as Weka only allows single class classification. Another limitation of the data is that among the thousands of attributes of music this study looks primary at MFCC, Mel-frequency cepstral coefficients, attributes of song. Specifically, the attributes discussed in the following sections are the only ones that were considered for this study, as follows:

1. MFCC - Mel-Frequency cepstral coefficients 1-12 are acoustic features that attempt to model the way in which a person hears and perceives things. It is calculated by using the Mel scale of the power spectral density of each frame to segment the energy into filter bands. The equation for Mel Scale conversion from frequency is:

where the f is the frequency of the signal. The different Mel scale values act as buckets for the frequencies and an upper and lower range for the frequencies needs to be chosen. So the Mel Scale might set twenty-six bands or ranges with the frequency band of 30Hz to 5000Hz. The log of the sum of the energies in each band is then done. Finally, the discrete cosine transform of the logs get us the MFCC coefficients. There are a total of 26 Mel-Frequency cepstral coefficients, but for most applications only 2-13 are used due to their relevance with music information retrieval applications.

1. Spectral Skewness - This attribute measures the degree to which a distribution is not symmetric. It is easily seen in a graph of the data, and it can be calculated from raw values.
2. Voice Probability - The probability of voicing is the percentage of voiced to unvoiced energy within each frame. All the samples are voiced songs for consistency.
3. Fundamental Frequency - This frequency in music is the pitch of a note that is determined to be the lowest present in the frame.
4. Zero-crossing rate of the time signal - This attribute is the rate that the signal is changing over time.
5. RMS Signal Frame Energy - The root-mean-square energy is an attribute related to dynamics and describes the overall volume of the audio piece.

The preceding set of music attributes combined with the min, max, average, and other statistical manipulations of the data combine to give each music sample 385 attributes to use for classification. This is a wide range of attributes and sufficiently describes the music samples. To put this in perspective, the RMS of a tense or scary music Sample, Incidious by Purple Planet Music, is 0.32958 and a classical music sample, River Flows in You by Yiruma, is 0.27221. If one was to make an approximation based on these two samples, one could say that the less volume in the music the more peaceful it is. Classification is not that simple, but this shows how that could work.

**Other Limitations of this Thesis.** This study will not take into account past experience or the association that a song has for someone that may elicit an emotion contrary to the music’s features. In omitting this, we hope to remove unwanted error and bias because humans naturally make associations with music to a time period or an event in their life. Eorela et al. argue of the complexity of music classification due to the high contextual dependence on the situation[[3]](#footnote-3). However, the underlying properties we postulate will accurately describe the emotional response for a majority of individuals.

The Naïve Bayes model assumes that the attributes within the dataset are conditionally independent, which for the most part is not true. However in a study done by Harry Zhang[[4]](#footnote-4), he argues that the Naïve Bayes algorithm still performs competitively in classification. Zhang postulates that in the real world most attributes are dependent, but because the dependencies largely distribute evenly in classes on each side of the distribution, it allows the Naïve Bayes algorithm to still perform fine because the dependencies cancel each other out. This allows us to utilize Naïve Bayes Model to effectively classify dependent attributes. Second, the one-way Anova test assumes the attribute data is normally distributed about the mean, which is true for some data and not for others, so this represents another limitation in the implementation of the modified algorithm presented in this study. In addition, the modified algorithm is only different from the base Naive Bayes algorithm on numeric datasets as oppose to nominal datasets. For example, lets say one was trying to classify the temperature of an unknown piece of metal. A nominal dataset for other metals might have the colors: blue, red, white, and orange, while a numeric dataset would have 100°C, 340°C, 510°C, and 205°C. The nominal dataset could be converted to a numeric by specifying a number value to the nominal values like blue: 1, red: 2, white: 3, orange: 4, but without the conversion the algorithm is not able to interpret the information. This limitation comes from the one-way Anova equations, which only work on numeric data. This ultimately means that for non-numeric datasets the modified algorithm will perform exactly as the original Naive Bayes algorithm as the base Naïve Bayes algorithm is able to classify nominal attributes.

The testing of the algorithm uses the same dataset for training and testing, which is convenient for our application. It does this by splitting the entire dataset into a percent for training and a percent for testing, but in the real world this may not be obtainable. In implementing the learning algorithms, one has to make sure that the data that is attempting to be classified is well represented by the training data else the model will perform very poorly. In our case, the data classification collection algorithm is somewhat fuzzy as it allows the same music to be classified into different emotions, as mentioned earlier.

**Necessary Mathematics and Algorithms**

In this section, we present a mathematical introduction to the classifiers and methods used for the majority of supervised learning algorithms and related topics. The Naïve Bayes classifier is a supervised learning algorithm that is used as the primary classification tool within Weka to determine an instance’s class. The one-way Anova statistic is used in conjunction with the Naïve Bayes algorithm, to remove attributes that are not significant thus producing a more sterile classification.

**Ensemble Methods.** Ensemble methods are methods used by many supervised learning algorithms to produce better performance by combining the results from multiple weaker or less robust classifiers. There are two common ensemble methods used today: Boosting and Bagging. Boosting, most commonly used by decision tree algorithms, takes classifiers and incrementally combines them with new classifiers. The new classifiers are targeted toward classifying the instances that the previous classifiers did not correctly classify. It can be more accurate than bagging, but it also has the issue of over-fitting, a term used to describe when a model describes random error or noise instead of the relationship giving a false positive. Bagging, conventionally used by Forest algorithms as an improvement on decision trees, separates out a subset of data into equal groups. The groups are run through a series of decision trees and the bag group votes based on which tree does best each time and the majority of the votes for all bags wins.

**Probabilistic Graph Models.** These models are graphs that are compiled of a set of nodes where each node is a random attribute. The graphs rely upon conditional independence as they derived their meaning from Bayes’ rule of probabilistic interference. Probabilistic graph models can be either directed or

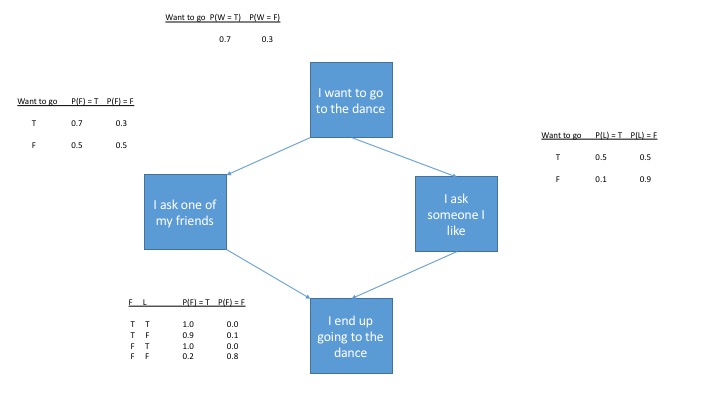


Figure - Directed Graph Model

undirected. Directed graph models have directional arrows between nodes as shown in the reinterpreted figure above from Kevin Murphy’s work[[5]](#footnote-5). In a simplified summary of the graph, if someone wants to go to the dance it will affect if they ask one of their friends and/or if they ask someone they like. Following the tree down, having asked a friend and/or asked someone they like affects whether they go to the dance. Directed graph models are typically used for Bayes Net algorithms. Additionally, they provide cause and effect relationships and are best suited to classification scenarios. An undirected graph model will not have the directional arrows like the directed model and there is no concept of Attribute A affecting or causing Attribute B. Undirected graph models are typically used in Markov networks.

**Naïve Bayes Model.** The Naïve Bayes model is a supervised learning algorithm based on Bayes’ Theorem that was first developed by Reverend Thomas Bayes in the 1700s as a way to predict the likelihood of an event occurring. Pierre-Simon Laplace later reformed it on in his published works. The Naïve Bayes model utilizes a simplification of the Bayes’ theorem by assuming that variables or attributes of an event are conditionally independent of each other thus reducing the number of input parameters needed when producing the classification model from 2(2n – 1) to 2n[[6]](#footnote-6). This makes the derivation of the Naïve Bayes Algorithm in its simplest form to be:

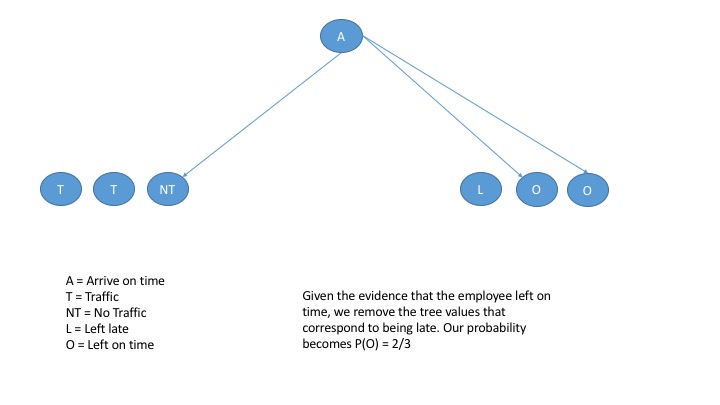
Where X is the Boolean values of the n attributes in the given set and P is the probability of Y given X. The equation reads the Probability that some condition Y happens based on attributes X1 through Xn is the product of the probability of Y given each individual attributes. For example, if there is an 2/3 probability that an employee will be on time if they left on time and there is a 1/3 percent chance that someone is on time given that there is no traffic. We are then given the conditions that the employee arrived on time, we are left to predict the probability that the employee left on time is calculated below:

Figure - Naive Bayes Probability Tree

This basic Naïve Bayes equation assumes that both X and Y are Boolean values, but the goal of a classifier is to predict or output the probability of a given instance being a certain class and for X and Y to be any real value. So, the above equation is the basis for the next derivation of the equation as shown:

In the above equation, X is the set of attribute values and Y is any real value. The above equation reads that the probability of some classification is equal to the probability of the class occurring multiplied by the product of the probability of that classification given each attribute Xi. This is all divided by probability of each class multiplied by the product of the probability of that classification given each attribute Xi.

An example is shown below of this equation:

P(Rare) = 0.1

P(Jeweler A | Rare) = 0.3

P(Africa | Rare) = 0.2

P(Expensive | Rare) = 0.5

P(Uncommon) = 0.2

P(Jeweler A | Uncommon) = 0.3

P(Africa | Uncommon) = 0.3

P(Expensive | Uncommon) = 0.3

P(Common) = 0.7

P(Jeweler A | Common) = 0.4

P(Africa | Common) = 0.5

P(Expensive | Common) = 0.2

= 0.095 or 9.5% for rare, 0.171 or 17.1% for uncommon, 0.733 or 73.3% for common

This equation will output the probability for each of the classes in the dataset. If the application specifies that the most probable classification is needed, then the equation reduces to the following:

Equation (3) is the final derivation of the Naïve Bayes classifier and both equation (2) and (3) is used in the classifiers implemented in Weka. For the final derivation, the returned value for our example would be the classification of the classifier as the probability is the highest.

**One-Way Anova Statistic.** The one-way Anova test which stands for one-way analysis of variance attempts to determine the statistical significance of a set of data given a minimum of two groups. We chose one-way as oppose to two-way Anova because we want to determine the significance on a per classification basis and because it assumes that there is a single treatment with any number of levels. It is typically used on data involving people’s response, but it can be applied to this instance because Naïve Bayes algorithm assumes that the attributes contributing to an event are conditionally independent or in other words, one attribute does not influence another. The one-way Anova statistic assumes normally distributed data around the mean, which has its limitations. The Analysis of variance methodology works with mean squares and more specifically, “mean square within-groups”, MSwithin, and “mean square between-groups”, MSbetween. The one-way Anova F-Statistic is calculated with

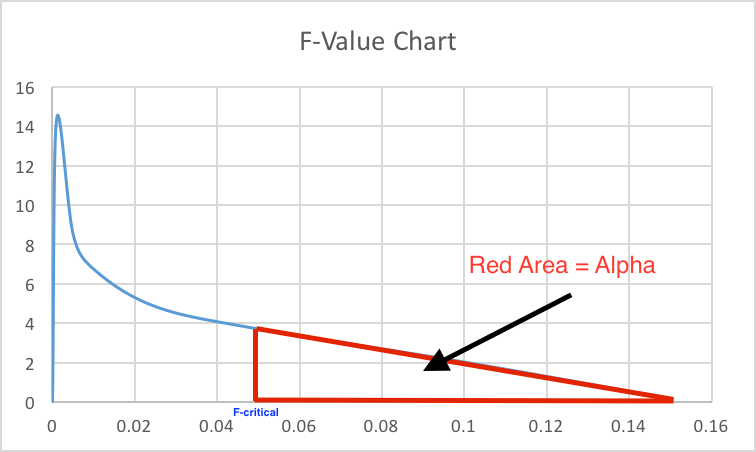
The F-value then gives us a p-value, probability value, via a lookup table, which can be compared against alpha or the level of confidence for the analysis. For this application an alpha of 0.05 was chosen, which represents a confidence of 95% that the result is accurate. So a p-value less than 0.05 represents significant data and should be used for classification. The shaded region in the graph in red in Figure 3 shows the F-critical region and denotes that the treatment was significant or in this application that the attribute is significant to the classification of the instance[[7]](#footnote-8). 

Figure - F-Critical Region

**CHAPTER 2**

**REVIEW OF THE LITERATURE AND RELATED WORK**

The area of music information retrieval is growing rapidly, but as Zentner points out “there is at present no systematic, empirically derived taxonomy of music induced emotions[[8]](#footnote-9).” So while it is growing, it is still finding its’ legs in certain aspects.

**Classifiers -- Review**

**Random Forest.** The random forest algorithm grows multiple classification trees where each tree is used to vote for a classification of the input object. The voting is then pooled and the classification with the most votes cumulatively is the chosen class. The tree is grown following a set of rules from Breiman and Cutler’s study[[9]](#footnote-10), as follows:

1. Each tree grows to the largest degree possible. This is accomplished without pruning, which is different from most tree classifiers.

2. The training set is taken from the original data input at random, but it replaces the cases each time.

3. A subset of the total number of attributes is chosen at random for each tree. The subset count is held constant through the training process and the best split of each attribute is determined and set for each node.

The advantages of the random forest algorithm are that it is currently the most accurate algorithm developed, it scales well, and estimates what attributes are most important to the classification. The disadvantage of this algorithm is that it is hard to interpret due to the level of complexity of the algorithm.

**Random Decision Tree.** The random decision tree algorithm grows multiple decision trees randomly. The algorithm grows these trees by randomly selecting an attribute and leaving it at each node as it traverses down. The tree stops growing in depth if the node no longer has an attribute to leave or at some predetermined depth chosen by the specific implementation of the algorithm. The input data is then run down the decision trees and class distributions are determined for each node on each tree. The trees are then checked for inconsequential nodes that exhibit similar classification distributions as its parent node. These nodes are pruned and the parent node becomes a leaf. After the pruning process, the leaves for each tree are used to classify an instance and the class distribution for each tree is then averaged. The class with the highest distribution percentage is the chosen class for the algorithm.

The advantages of this algorithm are it is easy to use, it scales well, and it is in the ballpark of most accurate algorithms.

**Linear SVM.** The Linear SVM algorithm is the linear kernel variant of the support vector machines. A support vector machine attempts to determine a line

that best separates data as shown in the below figure[[10]](#footnote-11). It picks the attributes that provide the biggest margin while being the most accurate.

Figure - Linear SVM

The advantages of this technique is that is it memory efficient, the model is not very large. Also, this algorithm can handle a large number of attributes. Consequently, it is not very accurate and the technique without modification will not provide classification distribution information.

**Bayes Net.** This algorithm is a type of directed probabilistic graph model and can be used to trace back the cause of a certain event. For example, lets say that Sally went to the beach. This could have been traced back to either a suggestion from a close friend or a need to get tan. The probabilistic distribution for these two events, the friend suggestion and needing a tan, determine what the Bayes Net algorithm predicts is the cause of the final event, ending up at the beach. This algorithm type is regularly used in the social media sphere to determine social fads or trends cause.

**Logistic Model Tree.** The Logistic Model Tree is a combination of two machine learning techniques. The first is logistical regression by which it gets its name. The second is the decision tree methodology as described in detail above. This algorithm combines the two by first creating a decision tree. Once the tree has been fleshed out, the leaves of the tree contain logistic regression functions that have been iteratively defined as one moves up the tree.[[11]](#footnote-12)

**Best-First Decision Tree.** This type of decision tree algorithm uses “best” node first when expanding the tree as oppose to depth first. The best node is described as “the node whose split leads to maximum reduction in impurity among all nodes available for splitting”. Impurity is calculated by using the Gini index (a measure of statistical breath of the data). This technique tries to remedy over fitting the data, a common pitfall of decision trees.

**Supervised Learning Algorithms Research**

Some of the foremost broad work on the topic of supervised learning algorithms have been done by Rich Capuana and Alexandru Niculescu-Mizil in their empirical study of such algorithms[[12]](#footnote-13). Their study breaks down all the common learning algorithms and compares their performance using a slew of metrics to best encompass all the pros and cons of the different algorithms. While they note that as the “No free lunch Theorem suggests, there is no universally best learning algorithm”, they do state that given their metrics calibrated boosted tree, random forests, and bagged tree perform the best on most datasets. The caveat with their test is that each learning algorithm was trained on 5000 samples. The availability of this much data for certain applications is plausible, but unrealistic in others.

Zoubin Ghahramani and Michael I. Jordan implement a supervised learning technique using a density estimation framework relying heavily on Gaussian’s mixture model as the basis for the classification[[13]](#footnote-14). This allows them to fill in for missing inputs or attributes. The use of a mixture of Gaussians is currently a frequently used technique due to its ability to still classify accurately despite the lack of data.

There are also many recent effects regarding Bayesian Networks. Goldenberg and Moore did a similar study to ours but they implemented a Bayesian Net to analyze social networks[[14]](#footnote-15). They concluded that there are significant gains to be had using a Bayes Net for large datasets within the social network realm. They found that using a SVM learning algorithm they were able to achieve an accuracy of 83 and 90% on two different datasets compared to a 50% default accuracy because the classification is binary (0 = In the class, 1 = not in the class).

Provost and Fawcett looked into the inadequacies with supervised learning algorithms as it pertains to nontrivial rare cases[[15]](#footnote-16). They argue that class accuracy is irrelevant and usually provides a false positive when it comes to rare class instances. For example, if 99% of the time a certain set of features in a song elicit a Tender emotion, but 1% of the time it is Anger and the emotion of Anger is not trivial in these instances, then supervised learning algorithms will do a poor job of determining the class. This is due to the fact that supervised learning algorithms are programmed to determine the most likely case and the number of past instances of the class does affect the probability of it occurring in the future.

**Emotional Mapping Research**

In the sphere of emotional mapping, Eorela et al. have done some considerable work classifying instances with the 5 basic emotions: Anger, Fear, Tender, Happy, and Sad[[16]](#footnote-17). They attempt to model the mappings to a three dimensional variant made up of valence, energy arousal, and tension arousal. Using this model, they were able to achieve an accuracy of 85% by using a 75-25% split of train to test of the annotated data of 800 samples. This was done using a linear regression model called PLS or Partial Least Squares Regression. PLS attempts to choses a relationship between a matrix of attributes and a matrix of classifications. In this case it attempts to determine the strong bonds between the emotions and certain music attributes.

Justin and Vastfjall did a study on the emotional responses to music and they created this fictitious example quoted below to exemplify the complexity within a single song, verbatim from [16][[17]](#footnote-18), as follows:

“Klaus arrived just in time for the concert on Friday evening . . . He sat down and the music began. A sudden, dissonant chord induced a strong feeling of arousal (i.e., brain stem reflex), causing his heart to beat faster. Then, when the main theme was introduced, he suddenly felt rather happy for no apparent reason (i.e., evaluative conditioning). In the following section, the music turned more quiet . . . The sad tone of a voice-like cello that played a slow, legato, falling melody with a trembling vibrato moved him to experience the same sad emotion as the music expressed (i.e., emotional contagion). He suddenly recognized the melody; it brought back a nostalgic memory from an event in the past where the same melody had occurred (i.e., episodic memory). When the melody was augmented by a predictable harmonic sequence, he started to fantasize about the music, conjuring up visual images – like a beautiful landscape – that were shaped by the music’s flowing character (i.e., visual imagery). Next, the musical structure began to build up towards what he expected to be a resolution of the tension of the previous notes when suddenly the harmonics changed unexpectedly to another key, causing his breathing to come to a brief halt (i.e., musical expectancy). He thought, ‘This piece of music is really a cleverly constructed piece! It actually made me reach my goal to forget my trouble at work.’ Reaching this goal made him happy (i.e., cognitive appraisal)”.

Justin and Vastfjall’s study shows that despite the statistics of the algorithms, there are limitations to the supervised learning algorithms because they produce a single classification for a piece of music when in fact 8 diffferent emotions can be happening within the same song.

Skowronek et al. attempted to develop an automatic music mood classification to help people sort their collection by music genre[[18]](#footnote-19). They developed an algorithm to classify the mood without the classes being mutually exclusive. The algorithm returns a binary classification of 1 = in the class or 2 = not in class. The features that were used for the evaluation were “signal describing features", “relevant tempo based features”, “chroma features”, and “percussive sound events”. Using the algorithm, they were able to get an accuracy of around 85% over the span of 12 emotion classifications, which is quite good for that breath of class categories.

**CHAPTER 3**

**METHODOLOGY**

**Utilized Software**

As mentioned earlier, in an effort to spend the most time gathering valuable analysis findings, we utilized some well-known and regarded tools in the field of study. The set of tools are listed subsequently. These tools helps to make the analysis part easier and simplify this study’s life cycle.

**Weka.** Weka is a data mining software available from the University of Waikato and can be downloaded from the Universities website or from source forge. Weka is a GUI interface for classifying, clustering and associating data according to the algorithm that is chosen. It reads .arff format audio files and allows the user to analyze machine learning algorithms. The learning curve for Weka is roughly a day for basic functions and significantly longer for creating your own classifiers and input files. However, the tool greatly enhances the performance of the user once they know their way around it. As shown in Figure 5, Weka has four GUI paths upon opening the tool:

Explorer - The primary GUI option that allows for classifying, filtering, clustering, etc. in an easy to use interface.

Experimenter - A GUI environment for conducting comparison tests between learning algorithms.

KnowledgeFlow - A drag and drop interface with the same functionality as the Explorer option.

 Simple CLI - Simple command line interface with access to all the classifiers, filters, and data output. It can be integrated with applications and systems to provide all the Weka functionality. Weka Explorer is the primary

Figure 5 - Weka GUI Choices

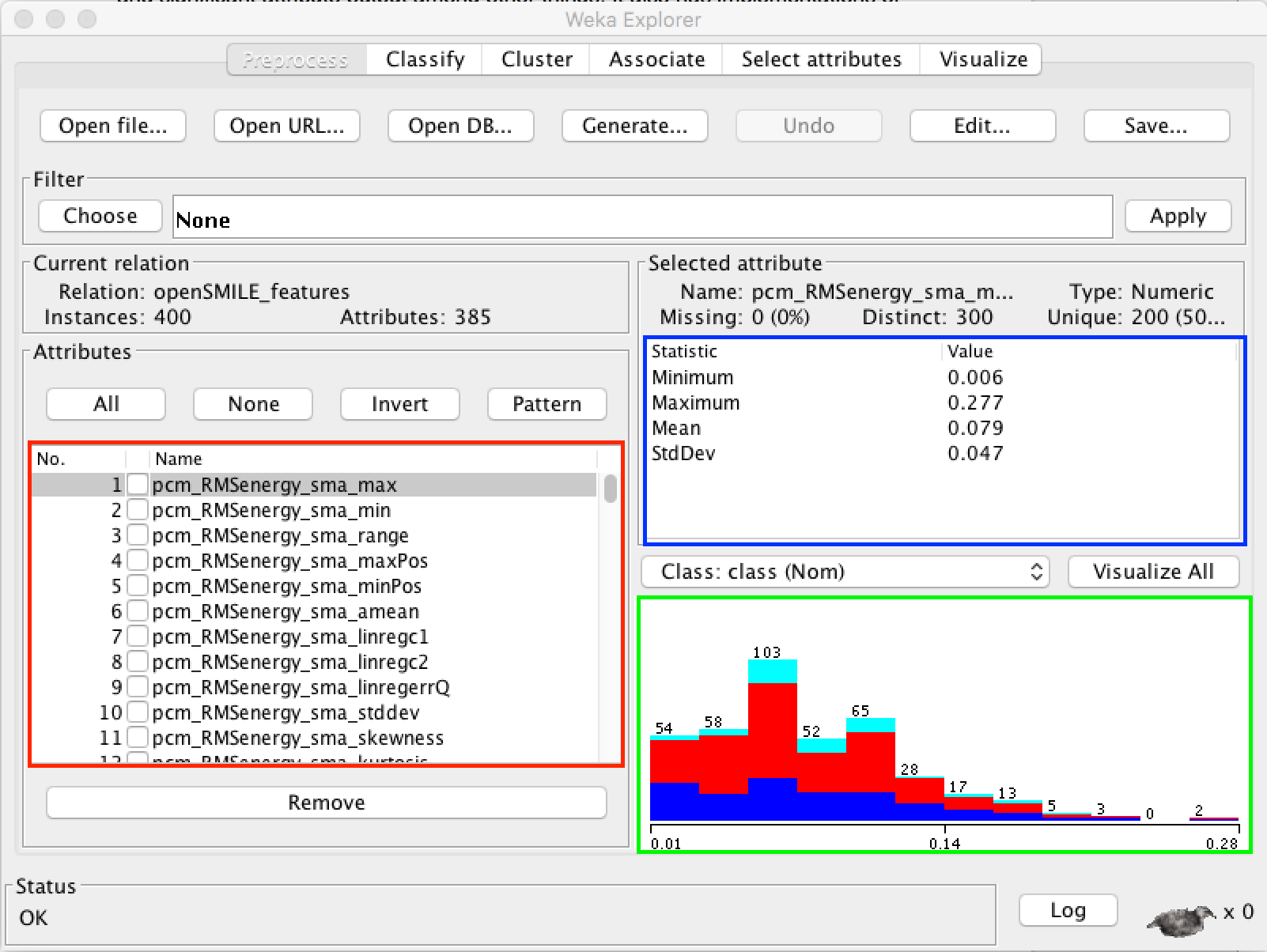
GUI we will be using. It is laid out as shown in Figure 6. The red bordered section shows the attributes ingested from your .arff file. The blue bordered section shows the statistics on the currently selected attribute. Finally, the green-

Figure 6 - Weka Explorer Preprocess

bordered section shows the visual breakdown of the attribute by class. Weka provides the ability to create training and test sets automatically for you along with cross validation folds. This gives your study the versatility to simulate many different scenarios that data might be received. Some of the noteworthy features that Weka provides are accuracy and error data, confusion matrices, and significant attribute output among other things. It also has implementations of the entire well known supervised learning algorithms all ready to go and to be tested against. The source codes for these implementations are included with the Weka source code. Overall, Weka was chosen as the software of choice because it is widely used by professionals and almost all technical papers in the area of machine learning use it.

**openSMILE.** In addition to Weka, we used openSMILE, which stands for Open Source Media Interpretation by Large Feature-Space Extraction. This tool allows us to convert the base .wav audio files to the desired attribute filled Weka format of .arff. This tool converts the files using signal analysis, autocorrelation functions, and subharmonic summations. This tool is a command line tool only, but it can be adapted to work with a programs GUI. We chose this tool over others like Marsyas because this tool is easy to use and very flexible for our purposes.

**Audacity.** This tool was an open-source solution for converting .mp3 to .wav files so that openSMILE could then convert those files to the designated Weka format. The process is arduous without the premium version of Audacity since files have to be converted 1 file at a time. However, it is one of the only open-source software solutions that has this functionality and does not rename the files. This was crucial for pairing up the annotated data to the audio data.

**Atom.** This is an IDE for building Node.js scripts that can be run from the command line. It is a lightweight-IDE that is modular to allow for upgrades in the areas needed and the ability to leave out other upgrades.

**Testing**

In this section, we would like to quickly discuss how we tested our algorithm and the different techniques available to further test the results.

**Testing Types.** The following are the most common methods of testing the performance of a supervised learning algorithm.

Holdout Method -This is one type of validation where a training and testing set are created by splitting the data by some percentage. One weakness of this method is that there is a high dependency on which data points get into the training set and which end up in the test set.

Cross-Validation with Folds - This testing type is a modification to the holdout method by splitting the data into *n* number of groups. These groups use the holdout method with each of the *n* groups taking a turn being the testing group and *n* -1 turns being the training group. The accuracy and other information is averaged across the runs. This method has less variance and will typically report more accurate results, but the time it takes to run this test is *n times longer* than the normal holdout method.

Testing Set -This method is similar to the holdout method only a separate testing set is provided and the entire dataset is used as training data. This method has disadvantages to consider. First if the testing set and training set come from different sources and the training set is not representative of the test set then the performance of the algorithms will be poor. Second, it results in less data to train the models because some data is left for just testing.

Training Set - This method uses the same data that was used to train the algorithms to test the algorithms. It is by far the simplest and the most prone to produce incorrectly good results.

**Test Setup.** We tested and analyzed our contributions via Weka, it utilizes a percent of the data for training to build the classifier and the rest for testing. We tested the algorithm with a 5 fold cross-validation run and a 75% train to test ratio. These provide a standardized method of determining level of accuracy and with seeded data, a repeatable set of data, allow for repeatable results if needed. The percent accuracy is not typically relevant because it is intrinsically linked to the dataset; therefore, a set of data about stocks or weather might be more difficult to classify than car tire life. In addition, within a category of data like weather, datasets still can be widely varying. So our view is that the analysis of the algorithms comes by comparing the algorithms rather than trying to achieve a certain accuracy.

**Test Algorithms.** We tested our contributions against variety of different algorithms to best determine the validity of the model. The algorithms that we tested against our modified algorithm were Random Forest, Best-First Tree, Bayes Net, and the original Naive Bayes algorithm. These algorithms are highly regarded and have been thoroughly vetted in the technical community. The Best-First Tree or Bayes Net are complex classifiers that acts as an upper boundary on the accuracy. The Random Forest classifier performs well in many applications across different fields, which also utilizes a very complex bagging algorithm and provides a high benchmark in the field of MIR.

**Process details -- Study Procedure**

In this section, we will quickly go over the processes needed to take a audio file in .mp3 format to a classified data statistic. First, the data is gathered in .mp3 format from Aljanaki et al.’s work with the 400 samples along with the annotated .csv file that gives the classifications. The .mp3 files are then fed to Audacity to be converted to a .wav file. Next, a bash script is run utilizing openSMILE to convert the .wav files to a single .arrf formatted file with all the data from the 400 samples in it. After this, a Node.js script is run that reads in both the .arff file and our annotated .csv file. The script then reads each line of the .csv file and counts the tally of each class for a given song. The class with the highest tally is the class that it appends to the .arff file for that song. Upon completion of the script, a global find and replace needs to be done to remove the name attribute from each song. This is done because the name category is a nominal field and thus not relevant in our study. Now that our .arff file is fully formatted and all the relevant data is included, the file is ready to be read in to Weka. As was demonstrated in the description of Weka section above, the Explorer GUI is Weka will be used and the file is opened in the preprocess tab. Once opened the classify tab allows the user to pick a classifier that has been implemented already in Weka or the user can decompress the .jar file in the source code and create their own classifier. After a classifier is selected the user will need to choose which type of test to run and in this case we chose a 5-fold cross-validation. Finally, clicking on the run button initializes the classifier. The classifier will run and at its completion the results are output to the window.

**Modified Naive Bayes Classifier**

The classifier was developed using the extracted version of the source code classifiers. In the extracted folder, there are tons of classes for core functionality, java libraries, filters, GUIs, classifiers, etc. We copied the Naive Bayes class as implemented to a new class called Predictive Approximation. Len Trigg and Eibe Frank developed the original Naive Bayes implementation based on the ideas of George John and Pat Langley as mentioned earlier. The modifications to the classifier were to address significance of the attribute to the classification. If the attribute was not significant then it was thrown out to better highlight the attributes that are significant to the classification. The first attempt we made to address this was by taking all the instance attribute values for a given class and determining the middle X percentile of the range as shown in Figure 7.

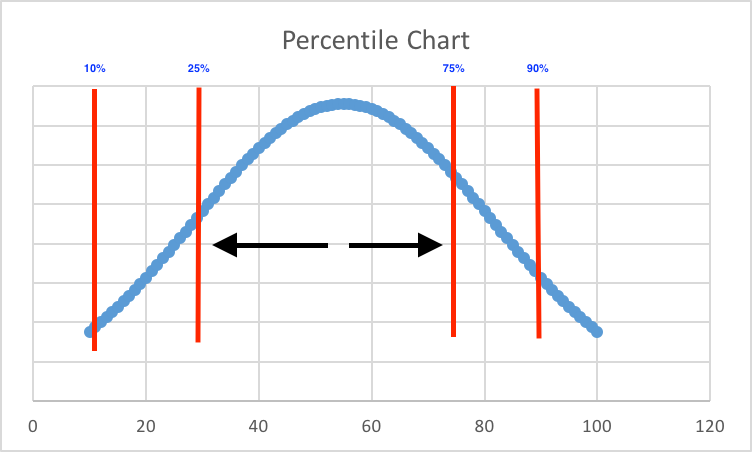


Figure 7 - Percentile Chart (Re-interpreted from [19])

This range was then used alongside a weight system based on the level of significance of the attribute to classify the test data. Upon implementing this, it was not found to be a benefit to this algorithm. After that attempt, we tried using the one-way Anova equation on each set of attributes to determine if significant or not. By doing this, the Naive Bayes calculations are not influenced in any way by the statistically insignificant factors as opposed to the first technique, which tries to add weight to each attribute via the distribution.

**CHAPTER 4**

**RESULTS, DISCUSSION AND FURTHER RESEARCH**

**Findings**

The results of the modifications to the Naïve Bayes algorithm marginally improve the results of the original Naïve Bayes algorithm. However, they are still significantly inferior to algorithms such as Random Forest or Bayes Net algorithms. The results of the different tests are shown in the Tables 1-3 below.

|  |  |  |
| --- | --- | --- |
| **4 Emotion Categories** | | |
|  | **5-Fold Cross Validation** | **Percentage Split 75-25** |
| **Random Forest** | 32% | 31% |
| **Naïve Bayes** | 28.25% | 24% |
| **Predictive Approximation** | 30.75% | 29% |
| **Bayes Net** | 32.75% | 29% |
| **Best-First Tree** | 30.75% | 29% |

Table 1- Accuracy by Classifier with four classes

|  |  |  |
| --- | --- | --- |
| **6 Emotion Categories** | | |
|  | **5-Fold Cross Validation** | **Percentage Split 75-25** |
| **Random Forest** | 22.75% | 30% |
| **Naïve Bayes** | 19.50% | 29% |
| **Predictive Approximation** | 22.25% | 32% |
| **Bayes Net** | 30.75% | 35% |
| **Best-First Tree** | 30.50% | 29% |

Table 2 – Accuracy by Classifier with six classes

|  |  |  |
| --- | --- | --- |
| **9 Emotion Categories** | | |
|  | **5-Fold Cross Validation** | **Percentage Split 75-25** |
| **Random Forest** | 23% | 17% |
| **Naïve Bayes** | 21% | 16% |
| **Predictive Approximation** | 18.75% | 14% |
| **Bayes Net** | 23.5% | 19% |
| **Best-First Tree** | 21.75% | 19% |

Table 3 – Accuracy by Classifier with nine classes

The tables are organized by the number of emotions included in the algorithm analysis. For the tables with less than 9 emotion categories, the emotions were combined by similar attributes of the emotions. For the 6 emotion categories the emotions combines were Joyful-Activation {Amazement, Joyful-Activation}, Nostalgia {Nostalgia}, Sadness {Sadness, Tenderness}, Tension {Tension}, Power {Power}, Calmness {Solemnity, Calmness}. The emotions got consolidated further to 4 emotion categories. The emotions combined for this were Joyful-Activation {Power, Amazement, and Joyful-Activation}, Nostalgia {Sadness, Tenderness, and Nostalgia}, Calmness {Solemnity, Calmness}, and Tension {Tension}. The best accuracy for each category is highlighted in yellow. Using the Weka experimenter tab tool, we also performed side-by-side analysis of the different algorithms. The results of the analysis are displaying in the Figure 8 for nine emotion categories with a 5-fold cross-validation.

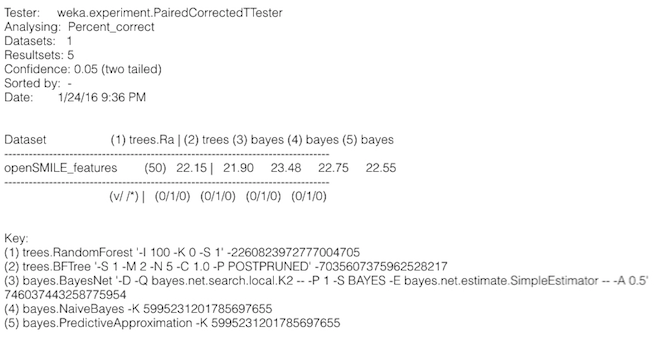


Figure 8 – Output of Weka Experimenter Analysis with nine classes

Similarly, Figure 9 and 10 are the results for the analysis for six and four emotion categories, respectively.

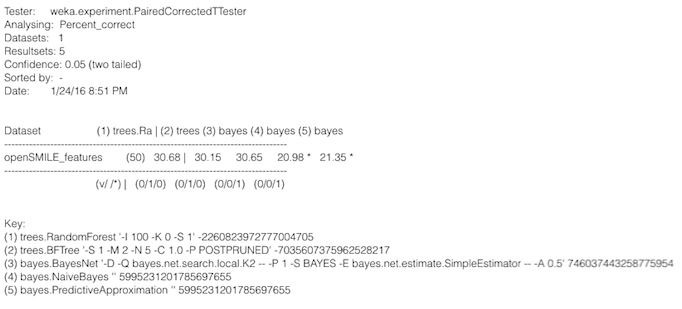


Figure 9 – Output of Weka Experimenter Analysis with six classes

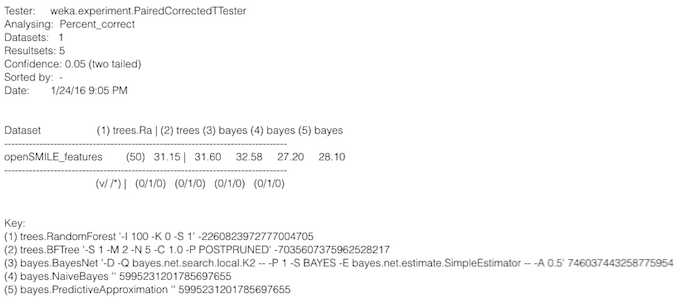


Figure 10 – Output of Weka Experimenter Analysis for four classes

These analyses show that though the Predictive Approximation algorithm does statistically better than the unmodified Naive Bayes algorithm, it still lags behind the Random Forest, Bayes Net, and Best-First Tree algorithms. However, it is still within +/- 2-3% of these top tiered algorithms.

**Discussion**

Based on the data, it can be concluded that though the improvements to the Naive Bayes algorithm are significant with a two-tailed confidence of .05, which allows for effects to be in two directions (positive and negative) with relation to the mean. They still do not achieve the performance of Bayes Net, Random Forest, or Best-First Tree over the span of all emotion category groupings. The Naive Bayes Algorithm and Predictive Approximation algorithms achieved an accuracy across the three subsets of 23.64% and 24%. This is lower than the accuracy of the other three models of 27.99%, 28.90%, and 27.88% for Random Forest, Bayes Net, and Best-First Tree respectively. This disparity is ultimately due to the fact that Naive Bayes algorithm along with the modified version focus on the data itself and try to optimize the initial model whereas Bayes Net, Best-First Tree, and Random Forest all focus on the testing of the model and refining its success rate. It is the difference between trying hard to refine one model to the best that can be done or choosing between hundreds of different models and seeing which returns the best results. It is apparent that in this application that the latter is more successful, but the best models is application dependent and has to be determined by experiment.

All the algorithms produced very poor performance on this music dataset compared to Caruana and Niculescu-Mizil’s empirical study of the algorithms and Skowronek et al’s study which reported accuracies in the 80 and 90th percentile[[19]](#footnote-20). This variance is due to a couple of different factors. First, as stated in the testing section no two datasets are alike and so the accuracies will similarly be different. As seen in Table 4, the algorithms accuracy for a different dataset, such as soybean classification, is exceptional for a variety of algorithms which do not produce the best results using the music classification. The data for this came from Weka’s samples and run on the different algorithms and shows the level of accuracy that can be achieved because, music data is so dependent on the individual taste, and classification, is hard to capture with these algorithms, in our opinion.

|  |  |  |
| --- | --- | --- |
| **Soybean Classification** | | |
|  | **5-Fold Cross Validation** | **Percentage Split 75-25** |
| **Random Forest** | 94.44% | 90.64% |
| **Naïve Bayes** | 92.68% | 88.89% |
| **Predictive Approximation** | 92.68% | 88.89% |
| **Bayes Net** | 93.11% | 90.64% |
| **Best-First Tree** | 91.80% | 94.15% |

Table 4 – Soybean Classification

Second, unlike some of the other studies where the algorithm just needed to determine if a music sample included that class or not which is a binary decision, our study had to pick between 9, 6, or 4 emotion categories by design. The breakdown of the default probability for those other studies with a binary decision is 50% as shown in Eorola et al’s study whereas for our study was 25%, 16.66%, and 11.11% assuming a random guess is chosen for the 4, 6, and 9 categories respectively. The difficulty in choosing just one class is that if attributes overlap significantly, then the algorithm is left to guessing, which the other algorithms better adapt to. In that way, it is possible that the classification of picking one of the possible several classifications is the basis of poor performance in the case of music-classification. We were limited to mapping one emotion for every song due to mapping of our method to Weka tools which required one emotion for the algorithms. Being able to handle “fuzzy” classification could possibly bring the accuracy down closer to the better classification and is a future direction which we would like to propose. This is also consistent with the opinion of one of the committee members who had predicted poor results based on debate in music industry as to whether a music piece can be attached to one particular emotion by an algorithm due to variable dependence and individual differences. Third, in our study the music samples had words to the songs. Words add an entirely new dimension of complexity to the classification. They have the ability to instantly predisposition someone to a certain emotion regardless of the attributes. Fourth, the songs used in the dataset are not polarizing. By this we mean that none of the songs are in the extremes of a given emotion mapping. For example, lets say we know the temperature is supposed to be between 70-90 degrees. If we are trying to predict whether the weather will be nice to play outside, it will be easier to predict that we will be able to play outside for that temperature range as opposed to between 30-50 degrees. This is polarizing data and easily lends itself to classification. On the other hand, our dataset of songs hangs in between a lot of emotions. Fifth, our emotion categories are more narrow than a typically “Happy, Sad, Tender, Angry, Fear” model. In our emotional categories, calmness and tenderness are very closely described as they can be very similar and so that effects the classification results as well. This makes it hard for the surveyors to differentiate what emotion they are feeling, which means the error (confusion of which emotion) is inherent in the data set at our disposal. Sixth, as discussed by Justin and Vastfjal in their study of music complexities, a given song can cause people to feel a variety of different emotions[[20]](#footnote-21) and also can elicit different emotions for the same piece of music. So this analysis is worth doing, asking a person to reclassify the same song. So in this case, the fact that each music sample is one-minute-long does it no service because the surveyor might not feel just one emotion, and the emotions may not be repeatable. Lastly (seventh), we feel that the classification data could be actually working. One further analysis which could be done is to tell the user that the algorithm is classifying this music piece as emotion A, and see how the user feels with that emotion classification even though they have said that the emotion they felt was B. In other words, with variability of data in our case, emotions generated by music, it is important to ask, how likely the user is to classify the data as emotion A, including their first choice as emotion B.

**Conclusion**

Music is an escape to some, a thrill for others, and ultimately it affects everyone differently. The challenge of distinguishing what about music makes some laugh, cry or cringe is still mostly a mystery. In the words of Victor Hugo, “Music expresses that which cannot be said and on which it is impossible to be silent[[21]](#footnote-22).” At the same time, when someone hears a crescendo and the event density growing, you can’t help feel your heart swell. These properties are concrete and not abstract. They can be measured and replicated. While algorithms have some ways to go to quantitatively measure this, the more effort that is put in this field the more it will improve. Just look at the mobile industry and the gains it has made in the last ten years due to the time that has been devoted to it. We believe that this field is a worthy cause and an endeavor well worth the time.

**BIBLIOGRAPHY/REFERENCES**

[1] Treasure, Julian. "The 4 Ways Sound Affects Us." TEDGlobal2009, July 2009. Web. 09 Apr. 2015. from https://www.ted.com/talks/julian\_treasure\_the\_4\_ways\_sound\_affects\_us#t-11867

[2] A. Aljanaki, F. Wiering, R. C. Veltkamp. Studying emotion induced by music through a crowdsourcing game. Information Processing & Management, 2015.

[3] Tuomas Eerola, Olivier Lartillot, Petri Toiviainen, "Prediction of Multidimensional Emotional Ratings in Music From Audio Using Multivariate Regression Models", International Conference on Music Information Retrieval, Kobe, 2009.

[4] Zhang, H. (2005). Exploring Conditions For The Optimality Of Naïve Bayes. International Journal of Pattern Recognition and Artificial Intelligence Int. J. Patt. Recogn. Artif. Intell., 19(02), 183-198. Retrieved January, 2016.

[5] Murphy, K. (1998). A Brief Introduction to Graphical Models and Bayesian Networks. Retrieved January, 2016, from http://www.cs.ubc.ca/~murphyk/Bayes/bnintro.html

[6] Mitchell, T. M. (1997). Machine Learning. New York: McGraw-Hill.

[7] Voice, A., Wilkins, A., Parambi, R., & Oraiqat, I. (2007). Factor analysis and ANOVA. Retrieved December, 2015, from https://controls.engin.umich.edu/wiki/index.php/Factor\_analysis\_and\_ANOVA

[8] M. Zentner, D. Grandjean, and K. Scherer. Emotions evoked by the sound of music: Characterization, classification, and measurement. Emotion, 8(4):494–521, 2008.

[9] Breiman, L., & Cutler, A. (n.d.). Random Forests. Retrieved January, 2016, from https://www.stat.berkeley.edu/~breiman/RandomForests/cc\_home.htm

[10] 1.4. Support Vector Machines. (2010). Retrieved January, 2016, from <http://scikit-learn.org/stable/modules/svm.html>

[11] Friedman, J., T. Hastie, and R. Tibshirani. (2000). Additive Logistic Regression: a Statistical View of Boosting. The Annals of Statistic 38(2), 337–374.

[12] Caruana, R., & Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. Proceedings of the 23rd International Conference on Machine Learning - ICML '06. Retrieved December, 2015.

[13] Ghahramani, Z., & Jordan, M. I. (1994). Supervised learning from incomplete data via an EM approach. Retrieved December, 2015.

[14] Goldenberg, A., & Moore, A. W. (2005). Bayes net graphs to understand co-authorship networks? Proceedings of the 3rd International Workshop on Link Discovery - LinkKDD '05. Retrieved January, 2016.

[15] Provost, F. J., & Fawcett, T. (1997). Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions. Knowl- edge Discovery and Data Mining (pp. 43–48).

[16] Juslin, P. N., & Västfjäll, D. (2008). Emotional responses to music: The need to consider underlying mechanisms. Behavioral and Brain Sciences Behav. Brain Sci., 31(06), 751. Retrieved January, 2016.

[17] Skowronek, J., McKinney, M., & Van de Par, S. (n.d.). A Demonstrator for Automatic Music Mood Estimation. Retrieved December, 2015.

[18] Hugo, V. (1887). William Shakespeare. Chicago: McClurg

[19] The Normal Distribution. (n.d.). Retrieved January, 2016, from http://people.stern.nyu.edu/churvich/MBA/Handouts/8-Normal.pdf

**Modified Naïve Bayes Classifier**

In Psuedo Code this Classifier does the following:

1. Starts with a collection of instances of training data
2. Enumerate through the attributes within the collection
   1. Repeat for each instance:
      1. i. Sort the values from high to low for an attribute
      2. ii. Compute the sum of the different of values
      3. iii. Increment the distinct variable count if values of Xn and
      4. Xn-1 is not the same.
      5. iiii. Store the attribute values in a new list for each class
   2. Calculate Precision
   3. Iterate thru the lists created by class
      1. i. Repeat for each class:
         1. Perform the one-way Anova Test on set where k is the count of values in the class for that attribute and N is the total for the current target class and each enumerated class.

* + - 1. These values are used to calculate Fcrit from a F distribution table.
      2. Then the SStotal, SSwithin, and SSbetween are calculated by the following where X is the current value in the list, is the overall mean of all classes for that attribute, and is the mean of the current class:
      3. Finally calculate the MSbetween, MSwithin, and Fcrit with the following where Fcrit needs to be greater than Fcrit from step 2 for the values to be significant and the test to pass true:

1. After model is complete via steps 1 and 2, classify test instance by iterating thru the attributes in the instance sums up the probability based on the model values with the attributes with no significant data having no impact on the determination.

package weka.classifiers.bayes;

import weka.classifiers.Classifier;

import weka.core.Attribute;

import weka.core.Capabilities;

import weka.core.Instance;

import weka.core.Instances;

import weka.core.Option;

import weka.core.OptionHandler;

import weka.core.RevisionUtils;

import weka.core.TechnicalInformation;

import weka.core.TechnicalInformationHandler;

import weka.core.Utils;

import weka.core.WeightedInstancesHandler;

import weka.core.Capabilities.Capability;

import weka.core.TechnicalInformation.Field;

import weka.core.TechnicalInformation.Type;

import weka.estimators.DiscreteEstimator;

import weka.estimators.Estimator;

import weka.estimators.KernelEstimator;

import weka.estimators.NormalEstimator;

import java.util.Enumeration;

import java.util.Vector;

import java.util.\*;

import java.lang.Object;

import org.apache.commons.math3.stat.inference.OneWayAnova;

/\*\*

<!-- globalinfo-start -->

\* Class for a Naive Bayes classifier using estimator classes. Numeric estimator precision values are chosen based on analysis of the training data. For this reason, the classifier is not an UpdateableClassifier (which in typical usage are initialized with zero training instances) -- if you need the UpdateableClassifier functionality, use the NaiveBayesUpdateable classifier. The NaiveBayesUpdateable classifier will use a default precision of 0.1 for numeric attributes when buildClassifier is called with zero training instances.<br/>

\* <br/>

\* For more information on Naive Bayes classifiers, see<br/>

\* <br/>

\* George H. John, Pat Langley: Estimating Continuous Distributions in Bayesian Classifiers. In: Eleventh Conference on Uncertainty in Artificial Intelligence, San Mateo, 338-345, 1995.

\* <p/>

<!-- globalinfo-end -->

\*

<!-- technical-bibtex-start -->

\* BibTeX:

\* <pre>

\* &#64;inproceedings{John1995,

\* address = {San Mateo},

\* author = {George H. John and Pat Langley},

\* booktitle = {Eleventh Conference on Uncertainty in Artificial Intelligence},

\* pages = {338-345},

\* publisher = {Morgan Kaufmann},

\* title = {Estimating Continuous Distributions in Bayesian Classifiers},

\* year = {1995}

\* }

\* </pre>

\* <p/>

<!-- technical-bibtex-end -->

\*

<!-- options-start -->

\* Valid options are: <p/>

\*

\* <pre> -K

\* Use kernel density estimator rather than normal

\* distribution for numeric attributes</pre>

\*

\* <pre> -D

\* Use supervised discretization to process numeric attributes

\* </pre>

\*

\* <pre> -O

\* Display model in old format (good when there are many classes)

\* </pre>

\*

<!-- options-end -->

\*

\* @author Corey McIntosh (cmcintos@uccs.edu)

\* @version $Revision: 1 $

\*/

public class PredictiveApproximation extends Classifier

implements OptionHandler, WeightedInstancesHandler,

TechnicalInformationHandler {

/\*\* for serialization \*/

static final long serialVersionUID = 5995231201785697655L;

/\*\* The attribute estimators. \*/

protected Estimator [][] m\_Distributions;

protected String [][] m\_anovaTest;

protected static final double DEFAULT\_ALPHA = 0.05;

/\*\* The class estimator. \*/

protected Estimator m\_ClassDistribution;

/\*\*

\* Whether to use kernel density estimator rather than normal distribution

\* for numeric attributes

\*/

protected boolean m\_UseKernelEstimator = false;

/\*\*

\* Whether to use discretization than normal distribution

\* for numeric attributes

\*/

protected boolean m\_UseDiscretization = false;

/\*\* The number of classes (or 1 for numeric class) \*/

protected int m\_NumClasses;

/\*\*

\* The dataset header for the purposes of printing out a semi-intelligible

\* model

\*/

protected Instances m\_Instances;

/\*\*\* The precision parameter used for numeric attributes \*/

protected static final double DEFAULT\_NUM\_PRECISION = 0.01;

/\*\*

\* The discretization filter.

\*/

protected weka.filters.supervised.attribute.Discretize m\_Disc = null;

protected boolean m\_displayModelInOldFormat = false;

/\*\*

\* Returns a string describing this classifier

\* @return a description of the classifier suitable for

\* displaying in the explorer/experimenter gui

\*/

public String globalInfo() {

return "Class for a Modified Naive Bayes classifier using estimator classes "

+" and significance determinance. Numeric "

+" estimator precision values are chosen based on analysis of the "

+" training data. For this reason, the classifier is not an"

+" UpdateableClassifier (which in typical usage are initialized with zero"

+" training instances) -- if you need the UpdateableClassifier functionality,"

+" use the NaiveBayesUpdateable classifier. The NaiveBayesUpdateable"

+" classifier will use a default precision of 0.1 for numeric attributes"

+" when buildClassifier is called with zero training instances.\n\n"

+"For more information on Naive Bayes classifiers, see\n\n"

+ getTechnicalInformation().toString();

}

/\*\*

\* Returns an instance of a TechnicalInformation object, containing

\* detailed information about the technical background of this class,

\* e.g., paper reference or book this class is based on.

\*

\* @return the technical information about this class

\*/

public TechnicalInformation getTechnicalInformation() {

TechnicalInformation result;

result = new TechnicalInformation(Type.INPROCEEDINGS);

result.setValue(Field.AUTHOR, "George H. John and Pat Langley");

result.setValue(Field.TITLE, "Estimating Continuous Distributions in Bayesian Classifiers");

result.setValue(Field.BOOKTITLE, "Eleventh Conference on Uncertainty in Artificial Intelligence");

result.setValue(Field.YEAR, "1995");

result.setValue(Field.PAGES, "338-345");

result.setValue(Field.PUBLISHER, "Morgan Kaufmann");

result.setValue(Field.ADDRESS, "San Mateo");

return result;

}

/\*\*

\* Returns default capabilities of the classifier.

\*

\* @return the capabilities of this classifier

\*/

public Capabilities getCapabilities() {

Capabilities result = super.getCapabilities();

result.disableAll();

// attributes

result.enable(Capability.NOMINAL\_ATTRIBUTES);

result.enable(Capability.NUMERIC\_ATTRIBUTES);

result.enable(Capability.MISSING\_VALUES);

// class

result.enable(Capability.NOMINAL\_CLASS);

result.enable(Capability.MISSING\_CLASS\_VALUES);

// instances

result.setMinimumNumberInstances(0);

return result;

}

/\*\*

\* Generates the classifier.

\*

\* @param instances set of instances serving as training data

\* @exception Exception if the classifier has not been generated

\* successfully

\*/

public void buildClassifier(Instances instances) throws Exception {

// can classifier handle the data?

getCapabilities().testWithFail(instances);

// remove instances with missing class

instances = new Instances(instances);

instances.deleteWithMissingClass();

m\_NumClasses = instances.numClasses();

// initalize Anova object

OneWayAnova anova = new OneWayAnova();

// Copy the instances

m\_Instances = new Instances(instances);

// Discretize instances if required

if (m\_UseDiscretization)

{

m\_Disc = new weka.filters.supervised.attribute.Discretize();

m\_Disc.setInputFormat(m\_Instances);

m\_Instances = weka.filters.Filter.useFilter(m\_Instances, m\_Disc);

}

else

{

m\_Disc = null;

}

// Reserve space for the distributions

m\_Distributions = new Estimator[m\_Instances.numAttributes() - 1]

[m\_Instances.numClasses()];

m\_ClassDistribution = new DiscreteEstimator(m\_Instances.numClasses(),

true);

m\_anovaTest = new String[m\_Instances.numAttributes() - 1]

[m\_Instances.numClasses()];

int attIndex = 0;

Enumeration enu = m\_Instances.enumerateAttributes();

while (enu.hasMoreElements()) {

Attribute attribute = (Attribute) enu.nextElement();

boolean anovaTest = false;

// If the attribute is numeric, determine the estimator

// numeric precision from differences between adjacent values

double numPrecision = DEFAULT\_NUM\_PRECISION;

if (attribute.type() == Attribute.NUMERIC) {

m\_Instances.sort(attribute);

if ((m\_Instances.numInstances() > 0)

&& !m\_Instances.instance(0).isMissing(attribute))

{

double lastVal = m\_Instances.instance(0).value(attribute);

double currentVal, deltaSum = 0;

Collection<double[]> data = new ArrayList<double[]>();

Collection<double[]> temp = new ArrayList<double[]>();

HashMap<Double, ArrayList<Double>> listMap =

new HashMap<Double, ArrayList<Double>>();

double currentClass = m\_Instances.instance(0).classValue();

int distinct = 0;

listMap.put(currentClass, new ArrayList());

listMap.get(currentClass).add(lastVal);

for (int i = 1; i < m\_Instances.numInstances(); i++)

{

Instance currentInst = m\_Instances.instance(i);

if (currentInst.isMissing(attribute))

{

break;

}

currentClass = currentInst.classValue();

currentVal = currentInst.value(attribute);

if (currentVal != lastVal)

{

deltaSum += currentVal - lastVal;

lastVal = currentVal;

distinct++;

}

if(!listMap.containsKey(currentClass))

{

listMap.put(currentClass, new ArrayList());

}

listMap.get(currentClass).add(currentVal);

}

double[] currentTarget = new double[listMap.get(currentClass).size()];

for(Map.Entry<Double, ArrayList<Double>> entry : listMap.entrySet())

{

List<Double> valueList = entry.getValue();

double[] target = new double[valueList.size()];

for (int i = 0; i < target.length; i++) {

target[i] = valueList.get(i);

}

if(entry.getKey() == currentClass)

{

currentTarget = target;

}

else

{

temp.add(target);

}

}

if (distinct > 0)

{

numPrecision = deltaSum / distinct;

}

for(double[] range : temp)

{

data.add(currentTarget);

data.add(range);

anovaTest = anovaTest || anova.anovaTest(data, DEFAULT\_ALPHA);

data.clear();

}

}

}

for (int j = 0; j < m\_Instances.numClasses(); j++) {

switch (attribute.type()) {

case Attribute.NUMERIC:

m\_anovaTest[attIndex][j] = String.valueOf(anovaTest);

if (m\_UseKernelEstimator) {

m\_Distributions[attIndex][j] =

new KernelEstimator(numPrecision);

} else {

m\_Distributions[attIndex][j] =

new NormalEstimator(numPrecision);

}

break;

case Attribute.NOMINAL:

m\_Distributions[attIndex][j] =

new DiscreteEstimator(attribute.numValues(), true);

break;

default:

throw new Exception("Attribute type unknown to NaiveBayes");

}

}

attIndex++;

}

// Compute counts

Enumeration enumInsts = m\_Instances.enumerateInstances();

while (enumInsts.hasMoreElements()) {

Instance instance =

(Instance) enumInsts.nextElement();

updateClassifier(instance);

}

// Save space

m\_Instances = new Instances(m\_Instances, 0);

}

/\*\*

\* Updates the classifier with the given instance.

\*

\* @param instance the new training instance to include in the model

\* @exception Exception if the instance could not be incorporated in

\* the model.

\*/

public void updateClassifier(Instance instance) throws Exception {

if (!instance.classIsMissing()) {

Enumeration enumAtts = m\_Instances.enumerateAttributes();

int attIndex = 0;

while (enumAtts.hasMoreElements()) {

Attribute attribute = (Attribute) enumAtts.nextElement();

if (!instance.isMissing(attribute)) {

m\_Distributions[attIndex][(int)instance.classValue()].

addValue(instance.value(attribute), instance.weight());

}

attIndex++;

}

m\_ClassDistribution.addValue(instance.classValue(),

instance.weight());

}

}

/\*\*

\* Calculates the class membership probabilities for the given test

\* instance.

\*

\* @param instance the instance to be classified

\* @return predicted class probability distribution

\* @exception Exception if there is a problem generating the prediction

\*/

public double [] distributionForInstance(Instance instance)

throws Exception {

if (m\_UseDiscretization) {

m\_Disc.input(instance);

instance = m\_Disc.output();

}

double [] probs = new double[m\_NumClasses];

for (int j = 0; j < m\_NumClasses; j++) {

probs[j] = m\_ClassDistribution.getProbability(j);

}

Enumeration enumAtts = instance.enumerateAttributes();

int attIndex = 0;

while (enumAtts.hasMoreElements()) {

Attribute attribute = (Attribute) enumAtts.nextElement();

if (!instance.isMissing(attribute)) {

double temp, max = 0;

for (int j = 0; j < m\_NumClasses; j++) {

temp = Math.max(1e-75, Math.pow(m\_Distributions[attIndex][j].

getProbability(instance.value(attribute)),

m\_Instances.attribute(attIndex).weight()));

if(attribute.type() == Attribute.NOMINAL || Boolean.parseBoolean(m\_anovaTest[attIndex][j]))

{

probs[j] \*= temp;

if (probs[j] > max)

{

max = probs[j];

}

}

else

{

probs[j] \*= 1e-75;

if (probs[j] > max)

{

max = probs[j];

}

}

if (Double.isNaN(probs[j])) {

throw new Exception("NaN returned from estimator for attribute "

+ attribute.name() + ":\n"

+ m\_Distributions[attIndex][j].toString());

}

}

if ((max > 0) && (max < 1e-75)) { // Danger of probability underflow

for (int j = 0; j < m\_NumClasses; j++) {

probs[j] \*= 1e75;

}

}

}

attIndex++;

}

// Display probabilities

Utils.normalize(probs);

return probs;

}

/\*\*

\* Returns an enumeration describing the available options.

\*

\* @return an enumeration of all the available options.

\*/

public Enumeration listOptions() {

Vector newVector = new Vector(3);

newVector.addElement(

new Option("\tUse kernel density estimator rather than normal\n"

+"\tdistribution for numeric attributes",

"K", 0,"-K"));

newVector.addElement(

new Option("\tUse supervised discretization to process numeric attributes\n",

"D", 0,"-D"));

newVector.addElement(

new Option("\tDisplay model in old format (good when there are "

+ "many classes)\n",

"O", 0, "-O"));

return newVector.elements();

}

/\*\*

\* Parses a given list of options. <p/>

\*

<!-- options-start -->

\* Valid options are: <p/>

\*

\* <pre> -K

\* Use kernel density estimator rather than normal

\* distribution for numeric attributes</pre>

\*

\* <pre> -D

\* Use supervised discretization to process numeric attributes

\* </pre>

\*

\* <pre> -O

\* Display model in old format (good when there are many classes)

\* </pre>

\*

<!-- options-end -->

\*

\* @param options the list of options as an array of strings

\* @exception Exception if an option is not supported

\*/

public void setOptions(String[] options) throws Exception {

boolean k = Utils.getFlag('K', options);

boolean d = Utils.getFlag('D', options);

if (k && d) {

throw new IllegalArgumentException("Can't use both kernel density " +

"estimation and discretization!");

}

setUseSupervisedDiscretization(d);

setUseKernelEstimator(k);

setDisplayModelInOldFormat(Utils.getFlag('O', options));

Utils.checkForRemainingOptions(options);

}

/\*\*

\* Gets the current settings of the classifier.

\*

\* @return an array of strings suitable for passing to setOptions

\*/

public String [] getOptions() {

String [] options = new String [3];

int current = 0;

if (m\_UseKernelEstimator) {

options[current++] = "-K";

}

if (m\_UseDiscretization) {

options[current++] = "-D";

}

if (m\_displayModelInOldFormat) {

options[current++] = "-O";

}

while (current < options.length) {

options[current++] = "";

}

return options;

}

/\*\*

\* Returns a description of the classifier.

\*

\* @return a description of the classifier as a string.

\*/

public String toString() {

if (m\_displayModelInOldFormat) {

return toStringOriginal();

}

StringBuffer temp = new StringBuffer();

temp.append("Predictive Approximation Classifier");

if (m\_Instances == null) {

temp.append(": No model built yet.");

} else {

int maxWidth = 0;

int maxAttWidth = 0;

boolean containsKernel = false;

// set up max widths

// class values

for (int i = 0; i < m\_Instances.numClasses(); i++) {

if (m\_Instances.classAttribute().value(i).length() > maxWidth) {

maxWidth = m\_Instances.classAttribute().value(i).length();

}

}

// attributes

for (int i = 0; i < m\_Instances.numAttributes(); i++) {

if (i != m\_Instances.classIndex()) {

Attribute a = m\_Instances.attribute(i);

if (a.name().length() > maxAttWidth) {

maxAttWidth = m\_Instances.attribute(i).name().length();

}

if (a.isNominal()) {

// check values

for (int j = 0; j < a.numValues(); j++) {

String val = a.value(j) + " ";

if (val.length() > maxAttWidth) {

maxAttWidth = val.length();

}

}

}

}

}

for (int i = 0; i < m\_Distributions.length; i++) {

for (int j = 0; j < m\_Instances.numClasses(); j++) {

if (m\_Distributions[i][0] instanceof NormalEstimator) {

// check mean/precision dev against maxWidth

NormalEstimator n = (NormalEstimator)m\_Distributions[i][j];

double mean = Math.log(Math.abs(n.getMean())) / Math.log(10.0);

double precision = Math.log(Math.abs(n.getPrecision())) / Math.log(10.0);

double width = (mean > precision)

? mean

: precision;

if (width < 0) {

width = 1;

}

// decimal + # decimal places + 1

width += 6.0;

if ((int)width > maxWidth) {

maxWidth = (int)width;

}

} else if (m\_Distributions[i][0] instanceof KernelEstimator) {

containsKernel = true;

KernelEstimator ke = (KernelEstimator)m\_Distributions[i][j];

int numK = ke.getNumKernels();

String temps = "K" + numK + ": mean (weight)";

if (maxAttWidth < temps.length()) {

maxAttWidth = temps.length();

}

// check means + weights against maxWidth

if (ke.getNumKernels() > 0) {

double[] means = ke.getMeans();

double[] weights = ke.getWeights();

for (int k = 0; k < ke.getNumKernels(); k++) {

String m = Utils.doubleToString(means[k], maxWidth, 4).trim();

m += " (" + Utils.doubleToString(weights[k], maxWidth, 1).trim() + ")";

if (maxWidth < m.length()) {

maxWidth = m.length();

}

}

}

} else if (m\_Distributions[i][0] instanceof DiscreteEstimator) {

DiscreteEstimator d = (DiscreteEstimator)m\_Distributions[i][j];

for (int k = 0; k < d.getNumSymbols(); k++) {

String size = "" + d.getCount(k);

if (size.length() > maxWidth) {

maxWidth = size.length();

}

}

int sum = ("" + d.getSumOfCounts()).length();

if (sum > maxWidth) {

maxWidth = sum;

}

}

}

}

// Check width of class labels

for (int i = 0; i < m\_Instances.numClasses(); i++) {

String cSize = m\_Instances.classAttribute().value(i);

if (cSize.length() > maxWidth) {

maxWidth = cSize.length();

}

}

// Check width of class priors

for (int i = 0; i < m\_Instances.numClasses(); i++) {

String priorP =

Utils.doubleToString(((DiscreteEstimator)m\_ClassDistribution).getProbability(i),

maxWidth, 2).trim();

priorP = "(" + priorP + ")";

if (priorP.length() > maxWidth) {

maxWidth = priorP.length();

}

}

if (maxAttWidth < "Attribute".length()) {

maxAttWidth = "Attribute".length();

}

if (maxAttWidth < " weight sum".length()) {

maxAttWidth = " weight sum".length();

}

if (containsKernel) {

if (maxAttWidth < " [precision]".length()) {

maxAttWidth = " [precision]".length();

}

}

maxAttWidth += 2;

temp.append("\n\n");

temp.append(pad("Class", " ",

(maxAttWidth + maxWidth + 1) - "Class".length(),

true));

temp.append("\n");

temp.append(pad("Attribute", " ", maxAttWidth - "Attribute".length(), false));

// class labels

for (int i = 0; i < m\_Instances.numClasses(); i++) {

String classL = m\_Instances.classAttribute().value(i);

temp.append(pad(classL, " ", maxWidth + 1 - classL.length(), true));

}

temp.append("\n");

// class priors

temp.append(pad("", " ", maxAttWidth, true));

for (int i = 0; i < m\_Instances.numClasses(); i++) {

String priorP =

Utils.doubleToString(((DiscreteEstimator)m\_ClassDistribution).getProbability(i),

maxWidth, 2).trim();

priorP = "(" + priorP + ")";

temp.append(pad(priorP, " ", maxWidth + 1 - priorP.length(), true));

}

temp.append("\n");

temp.append(pad("", "=", maxAttWidth +

(maxWidth \* m\_Instances.numClasses())

+ m\_Instances.numClasses() + 1, true));

temp.append("\n");

// loop over the attributes

int counter = 0;

for (int i = 0; i < m\_Instances.numAttributes(); i++) {

if (i == m\_Instances.classIndex()) {

continue;

}

String attName = m\_Instances.attribute(i).name();

temp.append(attName + "\n");

if (m\_Distributions[counter][0] instanceof NormalEstimator) {

String meanL = " mean";

temp.append(pad(meanL, " ", maxAttWidth + 1 - meanL.length(), false));

for (int j = 0; j < m\_Instances.numClasses(); j++) {

// means

NormalEstimator n = (NormalEstimator)m\_Distributions[counter][j];

String mean =

Utils.doubleToString(n.getMean(), maxWidth, 4).trim();

temp.append(pad(mean, " ", maxWidth + 1 - mean.length(), true));

}

temp.append("\n");

// now do std deviations

String stdDevL = " std. dev.";

temp.append(pad(stdDevL, " ", maxAttWidth + 1 - stdDevL.length(), false));

for (int j = 0; j < m\_Instances.numClasses(); j++) {

NormalEstimator n = (NormalEstimator)m\_Distributions[counter][j];

String stdDev =

Utils.doubleToString(n.getStdDev(), maxWidth, 4).trim();

temp.append(pad(stdDev, " ", maxWidth + 1 - stdDev.length(), true));

}

temp.append("\n");

// now the weight sums

String weightL = " weight sum";

temp.append(pad(weightL, " ", maxAttWidth + 1 - weightL.length(), false));

for (int j = 0; j < m\_Instances.numClasses(); j++) {

NormalEstimator n = (NormalEstimator)m\_Distributions[counter][j];

String weight =

Utils.doubleToString(n.getSumOfWeights(), maxWidth, 4).trim();

temp.append(pad(weight, " ", maxWidth + 1 - weight.length(), true));

}

temp.append("\n");

// now the precisions

String precisionL = " precision";

temp.append(pad(precisionL, " ", maxAttWidth + 1 - precisionL.length(), false));

for (int j = 0; j < m\_Instances.numClasses(); j++) {

NormalEstimator n = (NormalEstimator)m\_Distributions[counter][j];

String precision =

Utils.doubleToString(n.getPrecision(), maxWidth, 4).trim();

temp.append(pad(precision, " ", maxWidth + 1 - precision.length(), true));

}

temp.append("\n");

// now the anovaFValue

String anovaL = " anovaTest";

temp.append(pad(anovaL, " ", maxAttWidth + 1 - anovaL.length(), false));

for (int j = 0; j < m\_Instances.numClasses(); j++) {

String anovaTest = (String)m\_anovaTest[counter][j];

temp.append(pad(anovaTest, " ", maxWidth + 1 - anovaTest.length(), true));

}

temp.append("\n\n");

} else if (m\_Distributions[counter][0] instanceof DiscreteEstimator) {

Attribute a = m\_Instances.attribute(i);

for (int j = 0; j < a.numValues(); j++) {

String val = " " + a.value(j);

temp.append(pad(val, " ", maxAttWidth + 1 - val.length(), false));

for (int k = 0; k < m\_Instances.numClasses(); k++) {

DiscreteEstimator d = (DiscreteEstimator)m\_Distributions[counter][k];

String count = "" + d.getCount(j);

temp.append(pad(count, " ", maxWidth + 1 - count.length(), true));

}

temp.append("\n");

}

// do the totals

String total = " [total]";

temp.append(pad(total, " ", maxAttWidth + 1 - total.length(), false));

for (int k = 0; k < m\_Instances.numClasses(); k++) {

DiscreteEstimator d = (DiscreteEstimator)m\_Distributions[counter][k];

String count = "" + d.getSumOfCounts();

temp.append(pad(count, " ", maxWidth + 1 - count.length(), true));

}

temp.append("\n\n");

} else if (m\_Distributions[counter][0] instanceof KernelEstimator) {

String kL = " [# kernels]";

temp.append(pad(kL, " ", maxAttWidth + 1 - kL.length(), false));

for (int k = 0; k < m\_Instances.numClasses(); k++) {

KernelEstimator ke = (KernelEstimator)m\_Distributions[counter][k];

String nk = "" + ke.getNumKernels();

temp.append(pad(nk, " ", maxWidth + 1 - nk.length(), true));

}

temp.append("\n");

// do num kernels, std. devs and precisions

String stdDevL = " [std. dev]";

temp.append(pad(stdDevL, " ", maxAttWidth + 1 - stdDevL.length(), false));

for (int k = 0; k < m\_Instances.numClasses(); k++) {

KernelEstimator ke = (KernelEstimator)m\_Distributions[counter][k];

String stdD = Utils.doubleToString(ke.getStdDev(), maxWidth, 4).trim();

temp.append(pad(stdD, " ", maxWidth + 1 - stdD.length(), true));

}

temp.append("\n");

String precL = " [precision]";

temp.append(pad(precL, " ", maxAttWidth + 1 - precL.length(), false));

for (int k = 0; k < m\_Instances.numClasses(); k++) {

KernelEstimator ke = (KernelEstimator)m\_Distributions[counter][k];

String prec = Utils.doubleToString(ke.getPrecision(), maxWidth, 4).trim();

temp.append(pad(prec, " ", maxWidth + 1 - prec.length(), true));

}

temp.append("\n");

String anovaL = " [anovaTest]";

temp.append(pad(anovaL, " ", maxAttWidth + 1 - anovaL.length(), false));

for (int k = 0; k < m\_Instances.numClasses(); k++) {

String anovaTest = (String)m\_anovaTest[counter][k];

temp.append(pad(anovaTest, " ", maxWidth + 1 - anovaTest.length(), true));

}

temp.append("\n");

// first determine max number of kernels accross the classes

int maxK = 0;

for (int k = 0; k < m\_Instances.numClasses(); k++) {

KernelEstimator ke = (KernelEstimator)m\_Distributions[counter][k];

if (ke.getNumKernels() > maxK) {

maxK = ke.getNumKernels();

}

}

for (int j = 0; j < maxK; j++) {

// means first

String meanL = " K" + (j+1) + ": mean (weight)";

temp.append(pad(meanL, " ", maxAttWidth + 1 - meanL.length(), false));

for (int k = 0; k < m\_Instances.numClasses(); k++) {

KernelEstimator ke = (KernelEstimator)m\_Distributions[counter][k];

double[] means = ke.getMeans();

double[] weights = ke.getWeights();

String m = "--";

if (ke.getNumKernels() == 0) {

m = "" + 0;

} else if (j < ke.getNumKernels()) {

m = Utils.doubleToString(means[j], maxWidth, 4).trim();

m += " (" + Utils.doubleToString(weights[j], maxWidth, 1).trim() + ")";

}

temp.append(pad(m, " ", maxWidth + 1 - m.length(), true));

}

temp.append("\n");

}

temp.append("\n");

}

counter++;

}

}

return temp.toString();

}

/\*\*

\* Returns a description of the classifier in the old format.

\*

\* @return a description of the classifier as a string.

\*/

protected String toStringOriginal() {

StringBuffer text = new StringBuffer();

text.append("Predictive Approximation Classifier");

if (m\_Instances == null) {

text.append(": No model built yet.");

} else {

try {

for (int i = 0; i < m\_Distributions[0].length; i++) {

text.append("\n\nClass " + m\_Instances.classAttribute().value(i) +

": Prior probability = " + Utils.

doubleToString(m\_ClassDistribution.getProbability(i),

4, 2) + "\n\n");

Enumeration enumAtts = m\_Instances.enumerateAttributes();

int attIndex = 0;

while (enumAtts.hasMoreElements()) {

Attribute attribute = (Attribute) enumAtts.nextElement();

if (attribute.weight() > 0) {

text.append(attribute.name() + ": "

+ m\_Distributions[attIndex][i]);

}

attIndex++;

}

}

} catch (Exception ex) {

text.append(ex.getMessage());

}

}

return text.toString();

}

private String pad(String source, String padChar,

int length, boolean leftPad) {

StringBuffer temp = new StringBuffer();

if (leftPad) {

for (int i = 0; i< length; i++) {

temp.append(padChar);

}

temp.append(source);

} else {

temp.append(source);

for (int i = 0; i< length; i++) {

temp.append(padChar);

}

}

return temp.toString();

}

/\*\*

\* Returns the tip text for this property

\* @return tip text for this property suitable for

\* displaying in the explorer/experimenter gui

\*/

public String useKernelEstimatorTipText() {

return "Use a kernel estimator for numeric attributes rather than a "

+"normal distribution.";

}

/\*\*

\* Gets if kernel estimator is being used.

\*

\* @return Value of m\_UseKernelEstimatory.

\*/

public boolean getUseKernelEstimator() {

return m\_UseKernelEstimator;

}

/\*\*

\* Sets if kernel estimator is to be used.

\*

\* @param v Value to assign to m\_UseKernelEstimatory.

\*/

public void setUseKernelEstimator(boolean v) {

m\_UseKernelEstimator = v;

if (v) {

setUseSupervisedDiscretization(false);

}

}

/\*\*

\* Returns the tip text for this property

\* @return tip text for this property suitable for

\* displaying in the explorer/experimenter gui

\*/

public String useSupervisedDiscretizationTipText() {

return "Use supervised discretization to convert numeric attributes to nominal "

+"ones.";

}

/\*\*

\* Get whether supervised discretization is to be used.

\*

\* @return true if supervised discretization is to be used.

\*/

public boolean getUseSupervisedDiscretization() {

return m\_UseDiscretization;

}

/\*\*

\* Set whether supervised discretization is to be used.

\*

\* @param newblah true if supervised discretization is to be used.

\*/

public void setUseSupervisedDiscretization(boolean newblah) {

m\_UseDiscretization = newblah;

if (newblah) {

setUseKernelEstimator(false);

}

}

/\*\*

\* Returns the tip text for this property

\* @return tip text for this property suitable for

\* displaying in the explorer/experimenter gui

\*/

public String displayModelInOldFormatTipText() {

return "Use old format for model output. The old format is "

+ "better when there are many class values. The new format "

+ "is better when there are fewer classes and many attributes.";

}

/\*\*

\* Set whether to display model output in the old, original

\* format.

\*

\* @param d true if model ouput is to be shown in the old format

\*/

public void setDisplayModelInOldFormat(boolean d) {

m\_displayModelInOldFormat = d;

}

/\*\*

\* Get whether to display model output in the old, original

\* format.

\*

\* @return true if model ouput is to be shown in the old format

\*/

public boolean getDisplayModelInOldFormat() {

return m\_displayModelInOldFormat;

}

/\*\*

\* Returns the revision string.

\*

\* @return the revision

\*/

public String getRevision() {

return RevisionUtils.extract("$Revision: 1 $");

}

/\*\*

\* Main method for testing this class.

\*

\* @param argv the options

\*/

public static void main(String [] argv) {

runClassifier(new PredictiveApproximation(), argv);

}

}

**Node.js Conversion Script**

**Index.js**

This script reads in a .csv file of survey data and the .arff file of music data and tallies the values for each surveyor for each song. The emotion that is most common is chosen for the song and then propagated to each emotion attribute in the .arff file. The .arff file is then re-written out.

var csv = require('csv');

var arff = require('arff');

var readFile = require('fs').readFile;

var writeFile = require('fs').writeFile;

var closeFile = require('fs').closeFile;

var MusicData = require('./MusicData');

var listOfThings = [];

function createFiles (csv) {

var fileName = "/Users/coreyryanmcintosh/Desktop/openSMILE-2.1.0/MusicArff/MusicIS09.arff";

readFile(fileName, 'utf8', function(error, content)

{

if (error) {

return console.error(error);

}

var file = arff.parse(content);

var attr = file.attributes;

for (var index in attr)

{

if(attr[index].name == 'emotion')

{

attr[index].type = 'enum';

attr[index].values = ['Amazement', 'Solemnity', 'Tenderness',

'Nostalgia', 'Calmness', 'Power',

'Joyful-Activation', 'Tension', 'Sadness'];

}

}

for (var key in csv)

{

var maxValue = Math.max(csv[key].amazement, csv[key].solemnity,

csv[key].tenderness, csv[key].nostalgia, csv[key].calmness,

csv[key].power, csv[key].joyful, csv[key].tension, csv[key].sadness);

var data = file.data;

switch(maxValue)

{

case (csv[key].amazement):

data[key].emotion = 'Amazement';

break;

case (csv[key].solemnity):

data[key].emotion = 'Solemnity';

break;

case (csv[key].tenderness):

data[key].emotion = 'Tenderness';

break;

case (csv[key].nostalgia):

data[key].emotion = 'Nostalgia';

break;

case (csv[key].calmness):

data[key].emotion = 'Calmness';

break;

case (csv[key].power):

data[key].emotion = 'Power';

break;

case (csv[key].joyful):

data[key].emotion = 'Joyful-Activation';

break;

case (csv[key].tension):

data[key].emotion = 'Tension';

break;

case (csv[key].sadness):

data[key].emotion = 'Sadness';

break;

}

}

writeFile(fileName, arff.format(file), function(err){

if(err)

{

return console.error(err);

}

});

});

}

readFile('/Users/coreyryanmcintosh/Desktop/data.csv', 'utf8', function(error, content)

{

if (error) {

return console.error(error);

}

csv.parse(content, function(err, content){

content.shift();

var datas = content.map(function(row) {

return new MusicData(row);

});

var tempId = 1;

var newMusicData = new MusicData();

for (var key in datas)

{

if(datas[key].id == tempId)

{

newMusicData.id = datas[key].id;

newMusicData.genre = datas[key].genre;

newMusicData.amazement = datas[key].amazement + newMusicData.amazement;

newMusicData.solemnity = datas[key].solemnity + newMusicData.solemnity;

newMusicData.tenderness

= datas[key].tenderness + newMusicData.tenderness;

newMusicData.nostalgia = datas[key].nostalgia + newMusicData.nostalgia;

newMusicData.calmness = datas[key].calmness + newMusicData.calmness;

newMusicData.power = datas[key].power + newMusicData.power;

newMusicData.joyful = datas[key].joyful + newMusicData.joyful;

newMusicData.tension = datas[key].tension + newMusicData.tension;

newMusicData.sadness = datas[key].sadness + newMusicData.sadness;

}

else {

listOfThings.push(newMusicData);

tempId = datas[key].id;

newMusicData = new MusicData(datas[key]);

}

}

createFiles(listOfThings);

});

});

**MusicData.js**

This is a class that holds the MusicData object. It contains all the GEMs scale emotions and is used to store the survey data for each song.

function MusicData(row) {

row = row === undefined ? [] : row;

this.id = parseInt(row[0] || 0);

this.genre = row[1];

this.amazement = parseInt(row[2] || 0);

this.solemnity = parseInt(row[3] || 0);

this.tenderness = parseInt(row[4] || 0);

this.nostalgia = parseInt(row[5] || 0);

this.calmness = parseInt(row[6] || 0);

this.power = parseInt(row[7] || 0);

this.joyful = parseInt(row[8] || 0);

this.tension = parseInt(row[9] || 0);

this.sadness = parseInt(row[10] || 0);

}

module.exports = MusicData;

**Temp.js**

This script was written to allow for custom attributes of the songs that were not already in the .arff file to be put in the file. It reads in the .arff file and the custom attribute data in a .csv file and places each attribute in the corresponding song.

var csv = require('csv');

var arff = require('arff');

var readFile = require('fs').readFile;

var writeFile = require('fs').writeFile;

var closeFile = require('fs').closeFile;

var XLSX = require('xlsx');

var fileName = "/Users/coreyryanmcintosh/Desktop/openSMILE-2.1.0/MusicArff/MusicIS09.arff";

var fileName2 = "/Users/coreyryanmcintosh/Desktop/newData.xlsx";

readFile(fileName, 'utf8', function(error, content)

{

if (error) {

return console.error(error);

}

var file = arff.parse(content);

var workbook = XLSX.readFile(fileName2);

var worksheet = workbook.Sheets[workbook.SheetNames[0]];

var count = 2;

var loc = "B";

var data = file.data;

for (var key in data)

{

count++;

var location = loc + count;

data[key].event\_density = worksheet[location].v;

if(count == 102)

{

count = 2;

loc = String.fromCharCode(loc.charCodeAt(0) + 1);

}

}

writeFile(fileName, arff.format(file), function(err){

if(err)

{

return console.error(err);

}

});

});

1. (Treasure, 2009) [↑](#footnote-ref-1)
2. (Aljanaki et al., 2015) [↑](#footnote-ref-2)
3. (Eerola et al., 2009) [↑](#footnote-ref-3)
4. (Zhang, 2005) [↑](#footnote-ref-4)
5. (Murphy, 1998) [↑](#footnote-ref-5)
6. (Mitchell, 1998) [↑](#footnote-ref-6)
7. (Voice et al., 2007) [↑](#footnote-ref-8)
8. (Zentner et al., 2008) [↑](#footnote-ref-9)
9. (Breiman & Cutler) [↑](#footnote-ref-10)
10. (Support Vector Machines, 2010) [↑](#footnote-ref-11)
11. (Friedman et al., 2000) [↑](#footnote-ref-12)
12. (Caruana & Niculescu-Mizil, 2006) [↑](#footnote-ref-13)
13. (Ghahramani & Jordan, 1994) [↑](#footnote-ref-14)
14. (Goldenberg & Moore, 2005) [↑](#footnote-ref-15)
15. (Provost & Fawcett, 1997) [↑](#footnote-ref-16)
16. (Eerola et al., 2009) [↑](#footnote-ref-17)
17. (Juslin & Vastfjall, 2008) [↑](#footnote-ref-18)
18. (Skowronek et al.) [↑](#footnote-ref-19)
19. (Caruana & Niculescu-Mizil, 2006), (Skowronek et al.) [↑](#footnote-ref-20)
20. (Justin & Vastfjal, 2008) [↑](#footnote-ref-21)
21. (Hugo, 1887) [↑](#footnote-ref-22)