**Final Project Report**

**CIS 472**

**Dexin Wang**

**Han Wang**

**Zekun Tang**

**03/08/2016**

**Table of Contents**

**Abstract………………………………………………………………………………….**

**1.Introduction………………………………………………………………….……….**

**2. Data Integration……………………………………………..……………………….**

**2.1 Raw Data……………………………….……………………………………**

**2.2 Tables………………………………………………………………………..**

**2.3 Data Preprocessing……………………………………………………….**

**2.3.1 Table Combination…………………………………………..….**

**2.3.2 Data Clean………………………………….…………………..…**

**2.3.3 Data Transformation……………..…………………………..….**

**2.3.4 Data Selection……………..………….………………………….**

**3. Algorithms Applied……………………..…………………………………….…….**

|  |
| --- |
| **P3** |
| **P3** |
| **P4** |
| **P4** |
| **P5** |
| **P7** |
| **P7** |
| **P8** |
| **P8** |
| **P9** |
| **P9** |
| **P9** |
| **P12** |
| **P16** |
| **P19** |
| **P23** |
| **P26** |
| **P30** |
| **P34** |
| **P39** |

**3.1 General steps of our testing…………………….……………………….**

**3.2 J4.8 Decision Tree……………..………………………….……………….**

**3.3 Naive Bayes……………..………………………..……………..………….**

**3.4 Logistic Regression..……………………..……..……………….……….**

**3.5 Support Vector Machine ..………………………..……..……………….**

**3.6 Random Forest..…………………………………..……………………….**

**3.7 Perceptron…………………………………..……..……………………….**

**4.** [**Practical**](C:/Users/michelle.Han/AppData/Local/Yodao/DeskDict/frame/20160202181444/javascript:void(0);)[**Application**](C:/Users/michelle.Han/AppData/Local/Yodao/DeskDict/frame/20160202181444/javascript:void(0);)**………………………………………..……………..……….**

**5. Reference…………………………………………..………..………..…..………….**

**Abstract**

In this paper, we are going to apply three different strategies and algorithms to analyze the data from [kaggle.com](http://kaggle.com) about EMI Music users and predict their rating to a specific artist’s specific track. We have the data of users’ ratings on artists and tracks, their answers to some music questions, and their demographic and personal background information. Basically, the main purpose is to predict how much a certain user like a specific song.

**Keywords:**

*Data mining, Data mining methods, Data mining applications, Music data mining, Music rating prediction*

**1. Introduction**

On one hand, music artists create lots of new songs every day. On the other hand, users are exposed to various platforms to listen to songs. However, facing massive amount of music information, there are too many choices for users. They have no idea about how to choose a good artist or which songs fit their styles. It is a problem for users, because nobody wants to waste time and money to listen a song that they do not like eventually. It is also a problem for music platforms. How much time users will spend on music platforms depends on how easy it is for users to find the songs they like. The longer time users spend on music, the higher chance for music platforms to achieve its profit goals and commercial success. If music platforms can promote or suggest some tracks that users may like, they could attract new users and retain existing users to their music platform. Everyone benefits this way. Thus, it is necessary for music platforms to predict each user’s likings, preferences, and styles based on users’ online behaviors and existing records, such as their comments and reviews to previous listened songs, their answers to some general music questions, etc. Based on existing data, music platforms can mine the data, analyze their users behavior, predict each user’s rating to certain songs, and then recommend the best ones to each user.

**2. Data Integration**

**2.1 Raw Data**

We download data from [www.kaggle.com/c/MusicHackathon/data](https://www.kaggle.com/c/MusicHackathon/data). We selected four tables, Training, Testing, Users, Words tables, to analyze.

2.2 Tables

| Tables | |
| --- | --- |
| Training | A mapping table which maps users’ background, users’ words with Artist and Users. It also contains rating data to be trained. |
| User | It contains detailed information about users’ background. |
| Words | It contains detailed information about users' responses to some specific artists. |
| Testing | A mapping table which maps users' background, user’s communication records with Artist and User. The final goal is to predict rating on this table. |

| Training/Testing Data Set | |
| --- | --- |
| Attributes | **Functions** |
| Artist | An anonymized identifier for the EMI artist |
| Track | An anonymized identifier for the artist’s track |
| User | An anonymized identifier for the market research |
| Rating | Numerical entries that range between 0 to 100 to show how users like the music. Bigger number means higher rating |
| Time | An anonymized research date to help data miner understand which artists and tracks researched were in the same wave |

| Words Data Set | |
| --- | --- |
| Attributes | **Functions** |
| Artist | An anonymized identifier for the EMI artist |
| User | An anonymized identifier for the market research respondent, who will have just heard one or more samples from the artist |
| HEARD\_OF | An entry which answers the question: Have you heard of this artist and/or heard music by this artist? |
| OWN\_ARTIST\_MUSIC | answers the question: Do you have this artist in your music collection? |
| LIKE\_ARTIST | A numerical entry which answers the question: To what extent do you like or dislike listening this artist? |
| List of Words | There are 82 different words, ranging from “Soulful” to “Cheesy” and “Aggressive.” After listening to tracks from a particular artist, each respondent will select the words they think best describe the artist from a given set. The values in each column are therefore 1, if the respondent thinks that word describes the artist; 0 if the respondent does not think the word describes the artist; and empty if the word was not part of the current interview set. |

| Userkey/Users Data Set | |
| --- | --- |
| Attributes | **Functions** |
| User | The anonymized user identifier |
| Gender | Male/female |
| Age | The respondent’s age, in years |
| Working Status | Whether they are working full-time/retired/etc. |
| Region | The region of the United Kingdom where they live |
| Music | The respondent’s view on the importance of music in his/her life. |
| LIST\_OWN | An estimate for the number of daily hours spent listening to music they own or have chosen |
| LIST\_BACK | An estimate for the number of daily hours the respondent spends listening to background music/music they have not chosen |
| Music Habit Questions | Each of these asks the respondent to rate, on a scale of 0-100 |

**2.3 Data Preprocessing**

Data preprocessing could be a very important step to do in machine learning, because the higher quality data means higher accuracy. Therefore, we designed four programs to improve data quality.

**2.3.1 Table combinaison**

The goal of the final project is to predict the rating on the test table with some models. However, the models are created by different algorithms and more detailed information such as users’ background and users` responses. Therefore, we have to combine “Training”, “Words”, and “Users” with “artist” and “user” attributes as a new “Training” table. Because the models will apply to “testing” table to predict rating, we also need to combine “Testing”, “Words”, “Users” and to add “rating” column with “?”values as a new testing table.

**2.3.2 Data Clean**

In the new “Training” table, some attributes have empty values. For different attributes, we have to consider whether these empty values in attributes have to be replaced by the most frequent values. For example, each user will have age base on common sense. Therefore, the Age attribute in Users table should not be empty. On the other hand, some users may not hear some music from the specific artists, so they could not own these artists’ CDs. Hence, OWN\_ARTIST\_MUSIC in Words table is allowed to be empty.

**2.3.3 Data Transformation**

That rating is the most important attribute we have to predict. However, rating is the numerical type that ranges from 0 to 100. When we train the algorithms to predict numerical type, it is challenging to get higher accuracy. Hence, we change the numerical values from 1-5, which numbers mean intervals with a range of 20.

**2.3.4 Data selection**

After the table combination, each new Training table and new Testing table contains 115 attributes. It is too large to apply algorithms. Therefore, we select some influential attributes. One of our program is to calculates the distribution of each values in the attributes. For example, some attributes contain empty value which occupy more than 90 percentages (some attributes in Music Habit Questions). Therefore, we could consider these attributes as un-influential attributes.

**3. Algorithms Applied**

**3.1 General steps of our testing**

| 70% of training data for training, 30% of training data for testing. | |
| --- | --- |
| Step 1: | We tested the raw integrated data set with all features by a certain algorithm |
| Step 2: | We filtered features by different attributes evaluators and search methods based on what algorithms we used in first step to handle our data set. |
| Step 3: | Run the same algorithm again with selected features to test accuracy |
| Step 4: | Base on the selected features, we adjusted the parameters of algorithm to test multiple times. |
| Step 5: | Comparing the results of the same algorithm under different conditions. |

We split the training data to two parts. 70% for training, and 30% for testing. We do not use the testing data, because there is no label class in test data set. We are not able to estimate the accuracy of certain algorithm’s accuracy in this data set. Also, we do not apply cross validation to test our data set. We have tried to apply 10 folds to run our data set. However, under some algorithms, the Weka crashes easily because our data set is too big. Different algorithms have various costs to run. Our computers can not complete this mission under some conditions. We cannot compromise on a smaller data set, because small data could not provide enough information to support our model. This may impact our final model. Thus, in order to measure all algorithms’ performance in the same way, we just split our training data into two parts.

Basically, we tried to minimize cost and optimize efficiency step by step. In our data table, there are five different class labels, which means the random probability to pick up a correct label is 20%. We compared our accuracy with 20% to see how accurate we can improve to select a correct one.

In the first step, we just test a certain algorithm with our raw integrated data set roughly to get the baseline of both data sets, and the time needed to build model, and general accuracy.

In the second step, based on some rules and methods (We will talk specifically about what we use later in this paper), we selected necessary and related attributes to lower the cost of running algorithms and increase efficiency.

In the third step, based the selected attributes, we adjusted some parameters of our algorithm to compare the results like precision, FP, and recall. We record the highest accuracy and lowest model building time we got.

In the fourth step, we compare the results of the same algorithm under different conditions to find out the optimal solution of the highest precision and lowest model building time.

**3.2 J4.8 Decision Tree**

* **The benefit of applying J4.8 decision tree**

The first algorithm we use is J4.8 decision tree. Firstly, there are both nominal and numerical attributes mixed in our data set, because some of the attributes’ value cannot be changed to numerical. Decision tree is good to handle both nominal and numerical attributes in a data set. Secondly, it is survey data. There are some missing values. They may have negative impact to our prediction. However, decision tree is able to handle missing values to balance their effects in our result prediction. Thirdly, in our integrated data table, there are as many as 116 features. That means there are 116 dimensions in our data, which cannot be linearly separable. Decision tree would be a good option to figure out nonlinear data.

* **Testing raw integrated data on Weka (Test with all features).**

| **Build Model Time (s):** | **Correctly Classified Instances Percentage (%)** | **Detail Accuracy:** |
| --- | --- | --- |
| **16.97** | **49.0134** | **TP: 0.49**  **FP: 0.17**  **Precision: 0.484**  **Recall:0.49**  **F-Measure: 0.484** |

We run our raw integrated data with J4.8 decision tree roughly, and got the result in the above table. We set them as our baseline to compare with our later data selected table and algorithm adjusted one. Comparing to the 20% correct random probability, we can get a 49% correctness. The J4.8 decision tree can help to improve 145% correct possibility ((0.49 - 0.2) / 0.2 = 1.45).

* **Testing selected data on Weka.**

| **Build Model Time (s):** | **Correctly Classified Instances Percentage (%)** | **Detail Accuracy:** |
| --- | --- | --- |
| **5.94** | **48.5202** | **TP: 0.485**  **FP: 0.17**  **Precision: 0.478**  **Recall:0.485**  **F-Measure: 0.478** |

We selected 40 max information gain attributes from all 116 attributes. During the process of building a decision tree model, we need to calculate each attributes’ information gain, and always select the max one to add to the decision tree. We selected the biggest 40 of them, because they relate to decision tree model most. And they have the biggest impact on decision tree model. Also, 40 max information gain attributes take over one third of all features. It is a relatively high proportion. The selected attributes can present the entire attributes’ relationship with the label class. Moreover, reducing data dimensions contributes a lot to decrease the model build time. Less attributes leads to less work for the system to calculate their entropies, information gains and compare each of them to add to decision tree. In the table above, we noticed that the model build time decrease 65% ((16.97 - 5.94) / 16.97 = 0.6499). It obviously drops down. However, the accuracy decreases a little as well. Because we only take one third of attributes from the raw integrated table, less attributes and less information are available to create decision tree model. Inevitably, the model’s accuracy decreases. But the performance of selected data’s model is still very good. We cut 66.7% raw integrated data. The accuracy of the model only decrease by 0.49%, and the model build time is shortened by 65%. In general, the whole performance of the decision tree model is improved.

* **Testing adjusted algorithm on Weka.**

| **Build Model Time (s):** | **Correctly Classified Instances Percentage (%)** | **Detail Accuracy:** |
| --- | --- | --- |
| **7.21** | **50.2624** | **TP: 0.503**  **FP: 0.16**  **Precision: 0.502**  **Recall:0.503**  **F-Measure: 0.502** |

Based on selected attributes, we set the Unpruned parameter of decision tree from False to True. Both of the previous decision trees are pruned, but the new one is unpruned. The structure of unpruned decision tree is relatively complex. It has more branches than pruned decision tree, which means unpruned tree is complete tree. More branches need more time to build. Thus, even with the same amount of selected attributes, the unpruned decision tree needs more time to build the entire model. It requires 21.38% ((7.21 - 5.94) / 5.94 = 0.2138) more time to build an unpruned tree. Also, more complex tree with more branches means it can increase the accuracy. More branches in a decision tree, more detailed information from data table is recorded. Thus, it improves the accuracy. Comparing the second one with the new one, the accuracy increases 1.74%. In general, building an unpruned decision tree would sacrifice the building time but increase the prediction accuracy.

**3.3 Naive Bayes**

* **The benefit of Naive Bayes**

Firstly, Naive Bayes classifier is simple to train, by just doing several counts to calculate possibility for each attribute, and they time together. Find the max, it is the answer. Moreover, Naive Bayes is fast to train. The decoupling of the class conditional feature distributions means that each distribution can be independently estimated as a one dimensional distribution. This in turn helps to alleviate problems stemming from the curse of dimensionality (Zhang). Also, our data set is not linearly separable. Naive Bayes is able to handle multiple dimensions data set. Thus, we would like to choose Naive Bayes to test our data set.

* **Testing raw integrated data on Weka (Test with all features).**

| **Build Model Time (s):** | **Correctly Classified Instances Percentage (%)** | **Detail Accuracy:** |
| --- | --- | --- |
| **0.21** | **47.6667** | **TP: 0.475**  **FP: 0.166**  **Precision: 0.475**  **Recall:0.475**  **F-Measure: 0.433** |

We run Naive Bayes algorithm with our raw integrated data set. The result is in the above table. Firstly, the model building time is pretty fast. It only takes 0.21 second. Moreover, comparing the classified accuracy with random pick up accuracy seems good. After all, Naive Bayes algorithm increases the accuracy from 20% of pick up randomly to 47.667%. It increases 138.3% ((0.47667 - 0.2) / 0.02 = 1.383).

* **Testing selected data on Weka.**

| **Build Model Time (s):** | **Correctly Classified Instances Percentage (%)** | **Detail Accuracy:** |
| --- | --- | --- |
| **0.04** | **49.4%** | **TP: 0.494**  **FP: 0.163**  **Precision: 0.488**  **Recall:0.494**  **F-Measure: 0.489** |

We select attributes by Classifier Subset Evaluator to select data and get 28 most relative attributes as a subset. The Classifier Subset Evaluator can evaluate attribute subsets on training data or a separate hold out testing set. Uses a classifier to estimate the 'merit' of a set of attributes. And then, we run Naive Bayes again with 28 selected attributes. We get the result in the above table. We find that both model build time and classified accuracy have increased. Firstly, after attributes selected, although only 24% attributes left, they relate to class label very much. Thus, much less data needs to be processed, the model build time decreases a lot. It decreases 81% ((0.21 - 0.04) / 0.21 = 0.8095). Although 0.21 second is very fast when we run raw integrated data with Naive Bayes, the Naive Bayes deal with the selected data to predict almost instantly. Short model build time is very important in practical application; it can respond to request rapidly. Also, there are only 28 relative attributes left. We avoid some noisy data and irrelevant attributes’ impact. Thus, there are less attributes and information, we still can increase classified accuracy. In general, selected relevant attributes helps to decrease Naive Bayes both model build time and improve classified accuracy performance.

* **Testing adjusted algorithm on Weka.**

| **Build Model Time (s):** | **Correctly Classified Instances Percentage (%)** | **Detail Accuracy:** |
| --- | --- | --- |
| **0.04** | **49.4%** | **TP: 0.493**  **FP: 0.163**  **Precision: 0.485**  **Recall:0.493**  **F-Measure: 0.488** |

Based on the same selected data, we set the Debug parameter of Naive Bayes from False to True. The Debug classifier is supposed to output additional information to the console, and makes some changes in model build time and increase accuracy. However, there is no difference in model build time and classified accuracy. Probably, the data set is not big enough to make changes under this parameter.

**3.4 Logistic Regression**

* **The benefit of Logistic Regression**

The Logistic Regression algorithm is a supervised learning algorithm that can be used for regression for any type of N-dimensional signal. Firstly, our raw integrated data table mixes with different type of data, such as characters, integers, etc.. Logistic Regression can handle our raw integrated data well. Also, there are 116 data dimensions in our raw integrated data. Logistic Regression is good choice to deal with such many dimensions’ data.

* **Testing raw integrated data on Weka (Test with all features).**

| **Build Model Time (s):** | **Correctly Classified Instances Percentage (%)** | **Detail Accuracy:** |
| --- | --- | --- |
| **32.02** | **47.933%** | **TP: 0.479**  **FP: 0.164**  **Precision: 0.476**  **Recall:0.479**  **F-Measure: 0.477** |

We run Logistic Regression algorithm with our raw integrated data set in 100 iterations. The result is in the above table. Firstly, the model build time is acceptable but a little slow. It takes 32.02 seconds. Moreover, comparing the classified accuracy with random pick up accuracy, it seems good. After all, Logistic Regression increases the accuracy from 20% of pick up randomly to 47.933%. It increases 139.6% ((0.47933 - 0.2) / 0.2 = 1.396).

* **Testing Selected data on Weka**

| **Build Model Time (s):** | **Correctly Classified Instances Percentage (%)** | **Detail Accuracy:** |
| --- | --- | --- |
| **9.81** | **48.433%** | **TP: 0.484**  **FP: 0.17**  **Precision: 0.479**  **Recall:0.484**  **F-Measure: 0.479** |

We applied Chi Squared Attribute Evaluator to the top 40 attributes by their ranks among all of attributes. We evaluate the worth of an attribute by computing the value of the chi-squared statistic with respect to the label class, and choose the smallest 40 chi-squared attributes from 116. The smaller chi-squared values mean the closer relationship between them and class label. In order to have a fair comparison with Decision Tree algorithm, we choose 40 attributes.

Based on the 40 attributes, we run the Logistics Regression again with 100 iterations. We get the result in the above table. We notice that the model build time is shortened down from the previous rough test. It drops from 32.02 seconds to 9.81 seconds. It is a big change; it is about 69.36% ((32.02 - 9.81) / 32.02 = 0.6936) less than previous test. Also, we have 0.5% accuracy increased. Although it is a small change, we have only one third of attributes comparing with previous test and have much less model build time. The selected data model cost less, but it performed better than rough Logistics Regression model.

* **Testing adjusted algorithm on Weka**

| **Build Model Time (s):** | **Correctly Classified Instances Percentage (%)** | **Detail Accuracy:** |
| --- | --- | --- |
| **16.68** | **48.266%** | **TP: 0.483**  **FP: 0.17**  **Precision: 0.476**  **Recall:0.483**  **F-Measure: 0.477** |

Basing on the 40 selected attributes, we adjusted the Logistics Regression’s parameter Debug from False to True, and set iteration from 100 to 150. We get the result in the above table.

Firstly, the adjusted Logistics Regression’s model build time increased 70% ((16.68 - 9.81) / 9.81 = .7003). We turn on the Debug. It is a helper class for debugging output, logging, clocking, etc. It may take some time to debug the output, which increases model build time. Also, the adjusted algorithm run 150 iterations to classify input. More work to do to classify input correctly. It takes more time.

Moreover, the classified accuracy has a small drop. It may be overfitting. Although the attributes in our data set has already decreased to 40, the data set is still very big. There may be some noisy data which would impact the classified accuracy. Furthermore, we increase the iterations. The model could have memorized some wrong features of some attributes. With the increasing number of iterations, the model memorizes the wrong feature, and consequently makes more wrong decisions. Debug and more iterations are not good choices.

**3.5 Support Vector Machine**

* **The benefit of SVM**

Firstly, it has a regularization parameter, which makes the user think about avoiding over-fitting. It is robust to noisy data. It is beneficial to us to get higher classified accuracy. Secondly, with an appropriate kernel, SVM can work well even if data isn't linearly separable in the base feature space. Our raw integrated data table has 116 dimensions. We can’t separate our data linearly. SVM would be a good option to build model for prediction.

* **Testing raw integrated data**

| **Build Model Time (s):** | **Correctly Classified Instances Percentage (%)** | **Detail Accuracy:** |
| --- | --- | --- |
| **137.53** | **33.9** | **TP: 0.339**  **FP: 0.244**  **Precision: 0.319**  **Recall:0.339**  **F-Measure: 0.308** |

After running the SVM with our data in Weka, we get the result in the above table. Firstly, the model build time is relatively long. it takes more than 2 minutes. Moreover, we get an accuracy rate that is higher than our random pick up possibility. But its performance is fair.

* **Testing raw integrated data**

| **Build Model Time (s):** | **Correctly Classified Instances Percentage (%)** | **Detail Accuracy:** |
| --- | --- | --- |
| **18.04** | **46.6667** | **TP: 0.467**  **FP: 0.188**  **Precision: 0.467**  **Recall:0.467**  **F-Measure: 0.454** |

We applies Cfs Subset Evaluator to select 23 best attributes from 116 attributes. Cfs Subset Evaluator evaluates the worth of a subset of attributes by considering the individual predictive ability of each feature along with the degree of redundancy between them. And subsets of features that are highly correlated with the class label while having low inter-correlation are preferred.

After we run SVM with these 23 attributes, we get the result in the above table. Firstly, we notice that model build time has dropped down rapidly, it is about 86.88% ((137.53 - 18.04) / 137.53 = 0.8688). We only choose 23 attributes, which is about one fifth of the raw integrated data set. Weka has much less work to handle data. It can get result much more rapidly. Furthermore, the classified accuracy has improved a lot compared with previous rough test. It was an obvious improvement for accuracy, from 33.9% to 46.6667%. Classified accuracy has increased by 37.66 % ((0.46667 - 0.339) / 0.339 = 0.3766). The selected attributes have highly correlation with class label. We excluded a lot of noisy data which may disturb prediction results.

* **Testing adjusted algorithm on Weka**

| **Build Model Time (s):** | **Correctly Classified Instances Percentage (%)** | **Detail Accuracy:** |
| --- | --- | --- |
| **19.87** | **46.767** | **TP: 0.468**  **FP: 0.186**  **Precision: 0.463**  **Recall:0.468**  **F-Measure: 0.456** |

We normalize data, and run SVM again. We get result in the above table. We notice that model build time has improved a lot, which means the efficiency has dropped down. The model build time increase from 18.04 seconds to 19.87 seconds. It increased 10.14% ((19.87 - 18.04) / 18.04 = 0.1014). During the process, SVM needs to normalize data. It took time to handle data and meet the new request. However, at the cost of increased model build time, the classified accuracy has increased by 0.1%. SVM is more friendly to normalized data.

**3.6 Random Forest**

* **The benefit of Random Forest**

Random Forest is one of the most accurate learning algorithms available. For many data sets, it produces a highly accurate classifier. Also,

It runs efficiently on large databases. It is suitable to our raw integrated data set, because we have 116 attributes. Moreover, Random Forest can handle thousands of input variables without variable deletion. There are thousands instances in our data table. Finally, it gives estimates of what variables are important in the classification. During the process, we need to calculate its information gain to find out how each attribute correlates to class label.

| **Build Model Time (s):** | **Correctly Classified Instances Percentage (%)** | **Detail Accuracy:** |
| --- | --- | --- |
| **10.97** | **48.7** | **TP: 0.487**  **FP: 0.17**  **Precision: 0.488**  **Recall:0.487**  **F-Measure: 0.485** |

* **Testing raw integrated data**

We run our raw integrated data with Random Forest roughly, and got the result in the above table. We set them as our baseline to compare with our later data selected table and algorithm adjusted one. Comparing with the 20% correct random probability, we can get a 48.7% correctness. The Random Forest can help to improve 145% correct possibility ((0.49 - 0.2) / 0.2 = 1.45).

* **Testing selected data**

| **Build Model Time (s):** | **Correctly Classified Instances Percentage (%)** | **Detail Accuracy:** |
| --- | --- | --- |
| **7.8** | **50.3** | **TP: 0.503**  **FP: 0.164**  **Precision: 0.504**  **Recall:0.503**  **F-Measure: 0.501** |

We selected 40 max information gain attributes from all 116. During the process of building a Random Forest model, we need to calculate each attributes’ information gain, and always select the max one to add to the decision tree. We select the biggest 40 of them, because they relate to Random Forest model mostly. And they have biggest impact to decision tree model. It is fair to compare performance of other models.

Moreover, reducing data dimensions contributes a lot to the decrease of the model build time. Less attributes, the system need less work to calculate their entropies, information gains and compare each of them to add to decision tree. In the table above, we notice that the model build time decrease 28.89% ((10.97 - 7.8) / 10.97 = 0.2889). It obviously drops down. Also, the accuracy has increased. It changes from 48.7% to 50.3%, it increased 1.6 percent. Because through selecting attributes, we eliminate some noisy data. It improves the data quality. Thus, we can get a higher classified accuracy.

* **Testing adjusted algorithm**

| **Build Model Time (s):** | **Correctly Classified Instances Percentage (%)** | **Detail Accuracy:** |
| --- | --- | --- |
| **13.89** | **51.6667%** | **TP: 0.517**  **FP: 0.159**  **Precision: 0.517**  **Recall:0.517**  **F-Measure: 0.515** |

Based on selected attributes, we set the Depth of Random Forest from 0 to 10, and set the number of trees from 100 to 200. The more depth of Random Forest and the more number of tree, the more completed the Random Forest. It takes more time to build more detailed Random Forest. That is why the model build time has increased a lot, from 7.8 seconds to 13.89 seconds. It increases 78.07% ((13.89 - 7.8) / 7.8 = 0.7807). Moreover, the completed Random Forest records more detailed information. It is beneficial to higher classified accuracy.

**3.7 Perceptron**

* **The benefit of Perceptron**

In this part, we are going to apply single layer perceptron and multiple hidden layers perceptron on the training dataset. As we know, the single layer perceptron is an algorithm for [supervised](https://en.wikipedia.org/wiki/Supervised_classification%22%20%5Co%20%22Supervised%20classification) learning of binary classifiers. It means that it is the function that can decide whether an input belongs to one class or another. But in our case, it will not work very well as other algorithms since our dataset has six classes. Binary classifier will perform badly.

However, we will also apply multiple hidden layers perceptron to our training dataset. Compared with other algorithm include single layer perceptron, the multiple layers’ perceptron is a feedforward artificial neural network model. It maps sets of input data onto a set of appropriate outputs. From our point of view, this algorithm should perform better than the others on our dataset. Here are several reasons, first of all, there are more than one hundred attributes in our dataset and at the time. Multiple hidden layers perceptron is able to deal with a huge number of features. Second, to run multiple hidden layers perceptron will save time since it runs efficiently on large datasets. Third, the training data we will use the raw integrated data which means that will be the number of noise. As we know, multiple hidden layers perceptron will help estimate which variables are important in class-action and be good at working with noise. Generally, multiple hidden layers perceptron will be a good algorithm for our project.

* **Testing raw integrated data**

| **Build Model Time (s):** | **Correctly Classified Instances Percentage (%)** | **Detail Accuracy:** |
| --- | --- | --- |
| **49.23** | **42.9** | **TP: 0.429**  **FP: 0.204**  **Precision: 0.332**  **Recall:0.429**  **F-Measure: 0.303** |

We run our raw integrated data with single layer perceptron roughly, and we get the result as the above table. We set them as our baseline to compare with our later data which are 3 hidden layers and 5 hidden layers perceptron. Compared with the 20% correct random probability, we can get around 42% correctness. The single layer perceptron helps to improve 110% correct possibility ((0.42 - 0.2) / 0.2 = 1.10).

* **Testing selected data**

| **Build Model Time (s):** | **Correctly Classified Instances Percentage (%)** | **Detail Accuracy:** |
| --- | --- | --- |
| **138.05** | **48.3** | **TP: 0.483**  **FP: 0.165**  **Precision: 0.497**  **Recall:0.483**  **F-Measure: 0.46** |

After we applied single layer perceptron, we find that even though we get our prediction better than randomly guess. But we think if we improve the perceptron algorithm, we will get even better predicted accuracy. This time, we want to apply multiple hidden layers perceptron to the raw integrated dataset. To change the algorithm parameter, number of hidden layer to 3 and build our model. Then we got our result as above table. With increasing hidden layer for perceptron algorithm, we find that the build model time changes dramatically. It is 49.23 second when we run single layer perceptron and changes to 138.05 second instead. And it increases more than twice as previous one. However, “Correctly Classified Instances Percentage” changes to 48.3% from 42.9%. As our expectation, multiple hidden layers perceptron performs a lot better than single layer one. Better performance of multiply hidden layers’ perceptron does not only show up on “Correctly Classified Instances Percentage”, but also from TP: 0.483, FP: 0.165, Precision: 0.497, Recall:0.483 and F-Measure: 0.460. Beside FP, all the other ones have increased. So we know that multiple hidden layers perceptron works better than single layer perceptron on certain dataset.

* **Testing adjusted algorithm**

| **Build Model Time (s):** | **Correctly Classified Instances Percentage (%)** | **Detail Accuracy:** |
| --- | --- | --- |
| **212.3** | **48.533** | **TP: 0.485**  **FP: 0.167**  **Precision: 0.502**  **Recall:0.485**  **F-Measure: 0.461** |

At this time, we set parameter of perceptron algorithm, number of hidden layer, to be 5. With increasing number of hidden layer, the training result improves a little from 3 hidden layer and performs a lot better than single layer perceptron as we expected. Not surprisingly, build model time increases to 212.30 with numbers of hidden layer increase. More computations have been done during the hidden layers, that is the reason why build model time increases. However, “correctly Classified Instances Percentage” has been improved from 48.3% to 48.5%. It means that more instances have been corrected classified. And we can find that from randomly guessing possibility, the accuracy increases 142.50% as (0.48.5 - 0.2) / 0.2 = 1.4250). It is a huge progress in class prediction. To compare the 3 hidden layers and 5 hidden layers’ perceptron’s result, all the values of TP, FP, Precision, Recall and F-Measure improved in different degrees. In general, from all analysis we did above, we know that, on one hand multiple hidden layers perceptron works better than single layer perceptron on certain dataset; and the other hand, with reasonably increasing the number of hidden layers of perceptron algorithm, we will train a relative good model for our project.

4**.** [**Practical**](C:/Users/michelle.Han/AppData/Local/Yodao/DeskDict/frame/20160202181444/javascript:void(0);)[**Application**](C:/Users/michelle.Han/AppData/Local/Yodao/DeskDict/frame/20160202181444/javascript:void(0);)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Trying Model** | J48 Decision Tree | Random  Forest | Naive Bayes | SVM | Perceptron | Logistic Regression |
| **Accuracy** | 50.2624% | 51.667% | 49.3% | 47.667% | 48.533% | 47.633% |

With data preprocessing and algorithm adjustment on six algorithms during training stage, we build up 6 models from them (J48 Decision tree, Random Forest, Naive Bayes, LibSVM, Perceptron, Logistic Regression). For now, we need to apply these training models to the test dataset to predict the result which is rating from each user. From the table1 we can find that even though every training model’s accuracy is different from others, but all the accuracy is very closed. The accuracy is between 47% to 52%. It means that on one hand with the highest accuracy, 51.6667%, we cannot simply believe the random forest’s prediction; on the other hand, logistic regression’s accuracy (47.633%) is not as good as random forest and other models, but it still reasonably close to others. As the result of this, when we apply the test dataset to any single model of them, we cannot receive the most reliable prediction for each instance in dataset.

To find a solution of getting the most reliable prediction, we want to implement a simple way to achieve it. Without considering the time consuming, we are intention is pursuing the most reliable prediction which is rating of each user in test dataset. First step, we need to have six training models from different algorithm. Second step, we applied the test dataset to six different models and save predictions separately. Then, we need to analyze and compare the prediction. We should compute the number of the same rating which is prediction of each instance. For example, J48 decision tree, random forest, LibSVM and Naive Bayes’ models predict the rating of such an instance from test dataset is ‘4-4’. For the same instance in test dataset, perceptron and logistic regression predict its rating is ‘5-5’. So we can get that for such an instance, the number of predict 4 and ‘5-5’ is 2 and we need to record them. Last step is that with an example like above, we need to make a final decision for this instance’s prediction. Since the number of rating ‘4-4’ is bigger than rating of ‘5-5’, we should consider that for such the instance, we predict its rating is ‘4-4’. Here is the way to get a more reliable prediction than simply apply test dataset to a single training model.

When we applied our models to the test dataset, the predicted classes, rating, maybe very various. Not like the case as above, the predicted classes, rating, from six models can be that for J48 decision tree and random forest’s predicted rating is ‘4-4’, LibSVM and Naive Bayes’ models predict the rating is ‘5-5’ and perceptron and logistic regression predict its rating is ‘1-1’. So we can record that there are 2 ratings are ‘1-1’ as well as ‘4-4’ and ‘5-5’. At this case, we cannot simply decide predicted rating for such a instance since all three classified ratings has the same amount. We need to do is that look up each model’s accuracy and sum the specific two accuracies of having the same prediction. For this case if we sum the accuracy of perceptron and logistic regression and the result is bigger than J48 decision tree and random forest’s as well as LibSVM and Naive Bayes’. We will pick the predicted rating of perceptron and logistic regression which is ‘1-1’. Here is the way to handle this certain case.

Conclusion

Our project is a practical project from Kaggle.com which means that it is not like a course project. Course project usually focus on specific knowledge we learned within a time period and then trained us in order to have good understanding of this knowledge. But for project like what we do, we need to face and overcome numbers of difficulties we never handled in class project before like data integration, algorithm selection and result analysis. So our team members, Dexin Wang, Han Wang and Zekun Tang have to work closely together and pay extra attention on ‘accidents’ we came across. In our team’s division of task assignment, based on time-saving principle and each person’s interests, we assigned different tasks to each team member specifically. Dexin Wang responses for data integration. Han Wang and Zekun Tang focused on algorithm selection, data pre-processing and model comparison. For testing part which is predicting instances’ rating in test dataset, it is conducted by Han Wang.

Future work of our project will be focused on finding more appropriate algorithms for handling certain dataset in project. Since our team members are all beginners in machine learning field, main task for us is to get familiar with more algorithms and differences among those algorithms. After doing that we should be able to select a more appropriate algorithm when we deal with project. Another future work should be to redo the data pre-process step. Its goal is to select the best amount of most appropriate feature attributes. With better data pre-processing, we can build a better model for the later testing.

Reference

Ben-Hur, A., & Weston, J. (2010). A user’s guide to support vector machines. *Data mining techniques for the life sciences*, 223-239.

Naive Bayes. *Scikit Learn*. Retrieved Electronically on March 17th, 2016, from <http://scikit-learn.org/stable/modules/naive_bayes.html> .

Logistic Regression. *Gesture Recognition Toolkit (GTR)*. Retrieved Electronically on March 17th, 2016, from <http://www.nickgillian.com/wiki/pmwiki.php/GRT/LogisticRegression>.

Zhang, H. (2004). The optimality of naive Bayes. *AA*, *1*(2), 3.