**CS 6220 DATA MINING TECHNIQUES – COURSE PROJECT**

**Authors:**

**Nidhi Gupta**

**Laxmipriya Iyer**

**ABSTRACT:**

Given a dataset from UCI machine learning lab containing 10 years of clinical care at 130 US hospitals and integrated delivery networks. The data contains 50 attributes such as race, time in hospital, number of diagnosis etc. The task is to predict, as accurately as possible, whether a patient will be readmitted within 30 days after being discharged from hospital. This was achieved by the following process:

1. Transformed attribute/feature values containing string values by allocating a unique numerical value to each string value found in attribute
2. Preprocessed data with imputation of missing values
3. Used Filter method (Pearson correlation) to select a feature subset by analyzing the correlation between feature and class label
4. Handling the imbalance in training dataset
5. Developed a classification algorithm using different classifiers to classify the test instances

The overall test set accuracy using Ensemble was found to be **88.95%** and training set accuracy was found to be **87.36%.**

**PRELUDE:**

The project involved developing and implementing a classification algorithm to determine whether a patient will be readmitted within 30 days after being discharged from hospital. The project is implemented in Java and used WEKA Library. The project handles any given input format by internally converting it to ‘.arff’ format, a format required by Weka library. A brief description of the process followed is below:

Data Preprocessing and transformation

1. Transform Attributes containing string values into numerical values by assigning a unique numerical value to a string value for each feature. This conversion facilitates feature selection using Filter method (Pearson Correlation). This step is performed on both – training as well as test data.
2. Imputing missing values: We impute missing values by using mean (for continuous feature values) and mode (for discrete feature values) of records belonging to same class in dataset. This step is performed on both – training as well as test data.
3. Handling imbalance in the dataset: Learning algorithms get biased if the data is imbalanced, producing accurate results for majority class. This can be handled using data balancing techniques like under sampling, oversampling, Synthetic minority oversampling etc. We used Oversampling technique to handle the imbalance

Generate Test data and random sampling of training data

1. Allocate 20% of input data for testing and remaining 80% data for training.

From the allocated training data, use **Bagging** to create five random **balanced** (equal proportion of majority and minority class examples) samples of training data, each of size approximately 30K.

Building Classifiers

1. We used different classification algorithms (such as KNN, J48, Support Vector Machine, Naïve Bayes and Random Forest) to generate classifiers.
2. Ensemble Learning: Each test data instance is classified 0 (corresponds to No or greater than 30 days) or 1(corresponds to less than 30 days) using the generated classifiers. Final prediction is made by taking a majority vote model among the predictions of these classifiers.

Prediction and Accuracy

1. Predicted values are tested against truth values as given. Collective accuracy and intermediate accuracies within each model are detailed below.
2. **DATA PREPROCESSING AND TRANSFORMATION**

Types of data encountered: Categorical and Continuous

Data preprocessing step includes:

* Removal of some features by visual inspection and experimentation:

Features like encounter-id, patient number, and weight were removed from both

training and test spilt. Encounter-id and patient number were removed as they are

unique in the entire dataset and exhibit no correlation with the class label. Though

weight feature in theory should be considered a relevant feature but since approx.

96% of weight data is missing, our experiments did not show any positive

improvement in accuracy.

* Transforming String values into Numerical values:

Certain features like gender, race, medical specialty etc. contains string values and hence need to be converted into unique numerical values. This conversion was necessary to compute their Pearson correlation with class label.

* Handling of Missing Values:

There are three methods to handle instances with missing values:

* + Ignore/Delete the record: This method suggests deleting any instance which has missing values.
  + Impute the missing value using mean/median/mode: This is the most frequently used method for handling missing values. To compute missing values, this method uses the mean of records belonging to the same class (to reduce the bias) for continuous attributes, and the mode for categorical attributes.
  + Impute using Prediction model: In this method, we try to predict the missing values of an attribute using some classification algorithm and the non-missing values of attributes. In our dataset, we observe that we have 92478 (1.81% of entire training dataset) instances with missing values in the entire dataset. This represents a considerable percentage of the dataset to be discarded. Hence, deleting these entries affects the outcome and will lead to inaccurate results. Further analysis of the missing values reveals that different attributes (race, payer code and medical specialty) are missing across different instances in the dataset. Using prediction models to compute these missing values would require building multiple classifiers and running the prediction process across all these classifiers. Moreover, there are instances with more than one missing attribute value, further increasing the complexity of predicting individual attribute values. Hence, imputing missing values using prediction models would be an inefficient use of resources given that missing values represent less than 10% of the dataset. Consequently, imputing missing values using mean/mode was selected as the method of choice.
* Handling Imbalanced Dataset

As we learned in CS6220, standard learning algorithms are biased towards the majority class as it tries to reduce the global error rate, not taking into account the data distribution. Hence, we need to handle the imbalance in dataset.

In out sample dataset, we observe that we have only 11,357 (11.15%) records with class value ‘<30’ and remaining 90409 (88.83%) instances with value >30 or ‘NO’. This results a bias towards the class >30 or ‘NO’

To deal with this imbalance, we used Oversampling technique to increase the examples of minority class . We implemented random oversampling by selecting the minority class (‘<30’) multiple times. We randomly constructed five bags of size(30k), each bag constituting approx. 50% of majority and 50% minority class instances.

* Feature Subset selection

To use classifiers like KNN, it’s important to filter noisy/irrelevant features from training data. We used Filter method (Pearson correlation) to determine the correlation of the feature with the class label. We also experimented with Wrapper method (and used KNN algorithm), but it was computationally expensive on this dataset.

1. **GENERATE TEST AND BAGGING TRAINING DATA BEFORE MODEL GENERATION**

Split the pre-processed input data into test and training data by randomly allocating 20%

of the given input data for testing and remaining 80% for training. Though the training

reduces by using some chunk of it for testing purposes, it did not impact the accuracy

adversely as we used BAGGING to generate five random samples/bags of training data

with replacement. While generating bags/samples for training, we used Oversampling of

minority class which randomly replicated minority samples multiple times in each bag, thereby

ensuring an equal proportion of majority and minority class.

1. **BUILDING CLASSIFIERS**

Random Forest:

An ensemble classifier using many decision tree models. Random forest like any Ensemble technique works on reducing the variance by building multiple decision trees using random subsets of data and random subsets of attributes. In our implementation of Random forest, we are building 100 random trees with the max depth set at 8.

After conducting various experiments with above parameters, the best accuracy was found when these parameters are optimized at value of 100 for number of Random tree and a max depth set vat 8. Increasing the number of trees beyond 100, did not show any improvement in training accuracy and also increased the model building time.

Prediction and evaluation:

|  |  |
| --- | --- |
| Training Set | |
|  |  |
| Accuracy | 88.64% |
| Error | 11.36% |
| Mean Absolute Error | 0.1951 |

|  |
| --- |
| a = >30 or NO |
| b = <30 |

|  |  |  |  |
| --- | --- | --- | --- |
| Confusion matrix | | | |
|  |  | Predicted | |
|  |  | a | b |
| Actual | a | 26519 | 0 |
| b | 3400 | 0 |

|  |  |
| --- | --- |
| Test Set | |
|  |  |
| Accuracy | 88.80% |
| Error | 11.19% |
| Mean Absolute Error | 0.1941 |

|  |  |  |  |
| --- | --- | --- | --- |
| Confusion matrix | | | |
|  |  | Predicted | |
|  |  | a | b |
| Actual | a | 18250 | 0 |
| b | 2300 | 0 |

J48 (Decision Tree):

J48 is implemented using C4.5 decision tree algorithm. A decision tree is a flowchart like structure where each node represents the test on attribute value and each branch represents the outcome of the test and the tree leaves represent the class labels. At each node, C4.5 selects the attribute with the maximum information gain.

Prediction and evaluation

|  |  |
| --- | --- |
| Training Set | |
|  |  |
| Accuracy | 87.84% |
| Error | 12.16% |
| Mean Absolute Error | 0.1728 |

|  |
| --- |
| a = >30 or NO |
| b = <30 |

|  |  |  |  |
| --- | --- | --- | --- |
| Confusion matrix | | | |
|  |  | Predicted | |
|  |  | a | b |
| Actual | a | 25826 | 903 |
| b | 2771 | 709 |

|  |  |
| --- | --- |
| Test Set | |
|  |  |
| Accuracy | 88.13% |
| Error | 11.87% |
| Mean Absolute Error | 0.1928 |

|  |  |  |  |
| --- | --- | --- | --- |
| Confusion matrix | | | |
|  |  | Predicted | |
|  |  | a | b |
| Actual | a | 17982 | 268 |
| b | 2172 | 128 |

Naïve Bayes:

Naïve Bayes is a probabilistic classifier and is based on Bayes theorem .It assumes that each feature contribute independently of other feature, given the class variable. The algorithm classifies every instance based on the equation:

\hat{y} = \underset{k \in \{1, \dots, K\}}{\operatorname{argmax}} \ p(C_k) \displaystyle\prod_{i=1}^n p(x_i \vert C_k).

Prediction and evaluation

|  |  |
| --- | --- |
| Training Set | |
|  |  |
| Accuracy | 85.63% |
| Error | 14.37% |
| Mean Absolute Error | 0.1935 |

|  |
| --- |
| a = >30 or NO |
| b = <30 |

|  |  |  |  |
| --- | --- | --- | --- |
| Confusion matrix | | | |
|  |  | Predicted | |
|  |  | a | b |
| Actual | a | 25425 | 1365 |
| b | 2975 | 428 |

|  |  |
| --- | --- |
| Test Set | |
|  |  |
| Accuracy | 84.98% |
| Error | 15.02% |
| Mean Absolute Error | 0.1914 |

|  |  |  |  |
| --- | --- | --- | --- |
| Confusion matrix | | | |
|  |  | Predicted | |
|  |  | a | b |
| Actual | a | 17113 | 1137 |
| b | 1950 | 350 |

KNN (K-Nearest Neighbor)

KNN is an instance-based online learning algorithm (lazy learning algorithm). It stores all the training instances and classifies the test instances based on the k-nearest neighbors from the training data.

We experimented with different values of K, using 10 fold cross validation, starting with K=1. At K=1, the training accuracy observed is 85.1043% and at k=2, the accuracy increased to 86.3219%. At K=3, the accuracy dropped to 85.4178% and kept dropping thereafter.

We get maximum test accuracy of 86.3219% at k=2.

Prediction and evaluation

|  |  |
| --- | --- |
| Training Set | |
|  |  |
| Accuracy | 86.3219 |
| Error | 13.6781 |
| Root Mean Squared error | 0.3619 |

|  |
| --- |
| a = >30 or NO |
| b = <30 |

|  |  |  |  |
| --- | --- | --- | --- |
| Confusion matrix | | | |
|  |  | Predicted | |
|  |  | a | b |
| Actual | a | 25909 | 881 |
| b | 3159 | 244 |

|  |  |
| --- | --- |
| Test Set | |
|  |  |
| Accuracy | 87.9416 |
| Error | 12.0584 |
| Root Mean Squared error | 0.3791 |

|  |  |  |  |
| --- | --- | --- | --- |
| Confusion matrix | | | |
|  |  | Predicted | |
|  |  | a | b |
| Actual | a | 18025 | 225 |
| b | 2253 | 47 |

SVM (Support Vector Machine)

SVM is a maximum margin classifier and is a method of used for classification of both linear and non-linear data.

It uses non-linear transformation (known as the kernel trick) to transform the original data into a higher dimension where it searches for a linear optimal separating hyperplane (decision boundary). This ability of SVM to model complex non-linear decision boundaries make it high in accuracy as well as much less prone to overfitting.

The kernel function we implemented in our model is PolyKernel and the cost parameter C is adjusted. By doing various experiments and in each experiment taking a different value of C, a value of 0.1 is selected which balances the tradeoff between misclassification of training examples against the simplicity of decision plane

Prediction and evaluation

|  |  |
| --- | --- |
| Training Set | |
|  |  |
| Accuracy | 88.39% |
| Error | 11.61% |
| Mean Absolute Error | 0.1161 |

|  |
| --- |
| a = >30 or NO |
| b = <30 |

|  |  |  |  |
| --- | --- | --- | --- |
| Confusion matrix | | | |
|  |  | Predicted | |
|  |  | a | b |
| Actual | a | 26502 | 0 |
| b | 3482 | 0 |

|  |  |
| --- | --- |
| Test Set | |
|  |  |
| Accuracy | 87.9416 |
| Error | 12.0584 |
| Mean Absolute Error | 0.3791 |

|  |  |  |  |
| --- | --- | --- | --- |
| Confusion matrix | | | |
|  |  | Predicted | |
|  |  | a | b |
| Actual | a | 18250 | 0 |
| b | 2300 | 0 |

Bagging with J48, SVM, Naïve Bayes, Decision Tree and Random Forest

Bagging creates ensembles by “Bootstrap Aggregation”, by repeatedly randomly re-sampling the data. Bagging decrease the error by decreasing the variance due to unstable learners like decision trees On implementing J48, SVM, Naïve Bayes, Decision Tree and Random Forest we got test set accuracy as 88.95% (around 35 test instances are classified as re-admitted within 30 days and 20515 are classified as either readmitted after 30 days or not readmitted at all. We also observed that the bag size has a direct impact on accuracy. In our experiments we attained the best accuracy by keeping a bag size of = 35% of the train data where each bag represents an equal proportion of majority and minority class instances.

1. **ENSEMBLE LEARNING**

Once we have developed an ensemble of base classifiers, we ran these classifiers for all test instances and used a majority vote model to predict the final label of each test instance

The accuracy attained using Ensemble technique is 88.95%. Of the total number of 20200 test instances ,35 are predicted as ‘Readmitted within 30 days’ and 20515 are predicted as either ‘Readmitted>30 days or not readmitted’.

1. **PREDICTION AND EVALUATION**

To calculate the accuracy, we compare the predicted class value with the actual class value.

The final accuracy details obtained by our classification algorithm is below:

**Statistics**

|  |  |
| --- | --- |
| Training Set: After taking mean of accuracies from individual models | 87.36% |
| Test Set: After taking mean of accuracies from individual models | 87.72% |
| Test Set with Bagging Ensemble Technique | 88.95% |