CS 6220 DATA MINING TECHNIQUES – COURSE PROJECT ANALYSIS REPORT

Authors:

Ashita Dadlani

Pavitra Srinivasan

Raviteja Somisetty

Sandeep Lagisetty

ABSTRACT:

Given a dataset from UCI machine learning lab containing instances of an individual's education, demographic and family information, the prediction task was to determine whether an individual makes over 50K per year. This was achieved by the following process:

- Step I: Preprocessing data with proper imputation of missing values and handling the imbalance in training dataset
- Step II: Creating a classification algorithm using different classifiers and classifying test instances
- Step III: Producing various statistics derived from the classification algorithm

The overall test set accuracy was found to be 85.7625% and training set accuracy was found to be 90.2%.

ACKNOWLEDGEMENT:

We would like express our deepest gratitude to Prof. Yijun Zhao who gave us the opportunity to work on this project, and acted as a mentor throughout the course. Apart from enabling us to develop a new skill set, Prof. Zhao's encouragement allowed us to be creative and explore multiple ideas.

We would also like to thank our Teaching Assistant – Wei Luo for being both available and approachable throughout the course. His suggestions and inputs were immensely helpful in solving queries and clarifying concepts throughout the course.

INTRODUCTION:

The project involved generation and implementation of a classification algorithm to determine an individual's income status (income > 50K or income <=50K) based on education, demographic, and family information. The project is implemented in Java and uses WEKA Library. It handles any given input format by internally converting it to '.arff' format. A brief description of the process followed is below:

- Step I: Data Preprocessing
    - Replacing missing values: We impute missing values by using mean and mode of an attribute. This step is performed on both – training as well as test data.
    - Handling imbalanced dataset: Given the tendency of standard learning algorithms to be biased towards the majority class, imbalance in training data was handled using SMOTE (random oversampling of minority class).

- Step II: Algorithm Generation and Implementation
    - Generating Classifiers: We used multiple different classification algorithms (such as KNN, J48, Support Vector Machine, AdaBoost, Bagging, Naïve Bayes, and combinations of these) to generate classifiers.
    - Ensemble Learning: Each test data instance is classified (>50K or <=50K) using the generated classifiers. Final prediction is made by taking a majority vote among the predictions of these classifiers.

- Step III: Accuracy Evaluation
    - Accuracy is determined by comparing the actual and predicted class values.

Step II and Step III were repeated multiple times with different classifier combinations to improve overall accuracy of the algorithm.

Each of the above steps is further detailed below.

STEP I: DATA PREPROCESSING (Analysis)

Types of data encountered: Categorical and Continuous

Data preprocessing step includes:

- 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 over the entire dataset (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 2,399 (7.34% of entire training dataset) instances with missing values in training data and 1,211 (7.44% of entire test dataset) records with missing values in test 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 (workclass, occupation and native-country) are missing across different instances. Using prediction models to compute these missing values would require building multiple classifiers and

replicating the prediction process across all these classifiers. Additionally 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/median/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 they try 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 7,841 (24%) records with class value '>50K' and remaining 24,720 (76%) instances with value '<=50K'. This results a bias towards the class '<=50K'.
  To deal with this imbalance, we used Synthetic Minority Oversampling Technique (SMOTE). We implemented random oversampling by selecting the minority class ('>50K') multiple times. As a result we get a total of 43,538 instances, 18,818 of which have class '>50K' and 24,720 have class "<=50K" (same as before). This reduces the imbalance ratio to 43:57.

STEP II: ALGORITHM GENERATION AND IMPLEMENTATION (Analysis)

- **Generating Classifiers**

  - J48 (Decision Tree):
    Decision Tree builds classification in the form of tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes (with two or more branches) and leaf nodes (representing a classification or a decision). Decision Trees are widely used because they can produce high performance with minimal effort and can also be implemented alongside many efficient algorithms. J48 algorithm is based on univariate decision tree (splitting using one attribute at internal node). J48 is Non-Parametric, so we need not worry about outliers or whether the data is linearly separable i.e. they take care of cases where you have class A at the low end of some feature x, class B in the mid-range feature x and A again at the high end. The disadvantage of decision trees is that

they try to overfit the data, so to overcome we use ensemble methods (as detailed later).

Accuracy Evaluation:

Training Set:

- Accuracy: 90.6289%
- Error: 9.3711%
- Mean Absolute Error: 0.1412
- Confusion Matrix:

```
   a        b     <-- classified as
23323     1397  |    a = <=50K
 2683    16135  |    b = >50K
```

Test Set:

- Accuracy: 85.2159%
- Error: 14.7841%
- Mean Absolute Error: 0.1955
- Confusion Matrix:

```
   a        b     <-- classified as
11515    920   |    a = <=50K
 1487    2359  |    b = >50K
```

- Naïve Bayes:

  Naïve Bayes belongs to Generative model and is generally called "Punching Bag of Classifiers". It is a probabilistic classifier and is based on Bayes rule which relates current probability to prior probability and assumes strong independence between features. Naive Bayes does quite well when the training data doesn't contain all possibilities so it can be very good with low amounts of data. When there is huge amounts of data, Decision Trees work better than Naïve Bayes.
  This is the reason the accuracy of Naïve Bayes is less than J48 as we are working with a large dataset. Usually Naïve Bayes is a high bias/low variance classifier which has an advantage over low bias/ high variance classifiers (KNN) for a small data set. Naïve Bayes can be fooled by correlated attributes at it assumes strong independence of features. Consider a situation where P (A/C1) = P (A/C2) and P (B/C1) = P (B/C2) => marginal distribution of A and B in both classes are same. In this case Naïve Bayes can't classify data points as it relies on marginal distribution of attributes given class to do classification. If the distinguishing characteristic for classification is not the marginal distributions but correlation, then Naïve Bayes will not be a good choice. This is another potential reason for decision trees

outperforming Naïve Bayes. Naïve Bayes generally doesn't overfit the data, so we need not think about pruning. In a nutshell this algorithm classifies every instance based on the equation:

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

Accuracy Evaluation:

Training Set:

- Accuracy: 83.0079%
- Error: 16.9921%
- Mean Absolute Error: 0.1869
- Confusion Matrix:

a	b	<-- classified as
22791	1929	a = <=50K
5469	13349	b = >50K

Test Set:

- Accuracy: 83.6804%
- Error: 16.3196%
- Mean Absolute Error: 0.1719
- Confusion Matrix:

a	b	<-- classified as
11453	982	a = <=50K
1675	2171	b = >50K

- KNN (K-Nearest Neighbor)

KNN is an instance-based learning algorithm (lazy learning algorithm). It stores all the training instances and classifies the test instances based on the similarity measure between each test instance and k-nearest neighbors in training data. Being a lazy classifier, it is difficult to use this for prediction in real time but KNN has a more complex decision boundary than others, thus obtaining nice classification. If the decision boundary is linearly separable then KNN doesn't have much advantage over Naïve Bayes. KNN is used when the dimensionality is low and it doesn't work well for high dimensionality. The other problem in using KNN is choosing which features are relevant as they are related in deciding the distance metric. We can overcome this problem by choosing Decision trees which put initial effort on deciding important features.

We started the implementation of KNN by choosing k=1. We observed that on increasing the value of k, accuracy increased (k=1, accuracy = 79.17%; k=5, accuracy = 82.07%). At k=11, we get maximum test accuracy of 83.08%. On further increasing the value of k, accuracy decreases.
This happens because when k=1, we limit the comparisons to only one nearest neighbor. Consequently, the prediction is entirely driven by a single neighbor which can result in low accuracy. As we keep increasing the value of k, the prediction is based on majority vote and accuracy increases up to a certain optimal k. Beyond this optimal k, the comparison neighborhood (k nearest neighbors) becomes too large and prone to noise and outliers thereby decreasing prediction accuracy.

Accuracy Details:

Training Data:

- Accuracy: 88.35%
- Error: 11.65%
- Mean Squared Error: 0.164
- Confusion Matrix

```
    a      b      <-- classified as
22390  2330  |    a = <=50K
 2738 16080  |    b = >50K
```

Test Data:

- Accuracy: 83.0723 %
- Error: 16.9277
- Mean Absolute Error: 0.2142
- Confusion Matrix

```
   a      b     ← classified as
11062   1373   | a= <=50K
 1383   2463   | b= >50K
```

- <u>SVM(Support Vector Machine)</u>
  SVM uses non-linear mapping (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 them high in accuracy as well as much less prone to

overfitting. The main drawback of SVM is that the training time is very slow. This is the main reasons that SVM wasn't used with bagging and boosting (detailed later).

The kernel we have used is PolyKernel and the parameter to be adjusted is c. This parameter trades off misclassification of training examples against the simplicity of the decision surface. A low value of c makes the decision surface smooth (may cause underfitting), while a high value of c aims at classifying all the training instances correctly (leading to overfitting). Keeping these factors in mind, the time to build the model and classify test instances, the value of c was chosen to be 0.1. The accuracy of SVM is low when compared to KNN and Naïve Bayes which is an anomalous behavior. SVM can be difficult in determining the good value of the offset. We construct an ROC curve from the scores of the SVM algorithm outputs, where we sort the scores and use each as a positive/negative classification threshold. Based on the ROC curve, we set the new offset parameter of the score that best balances the precision/recall. PolyKernel with higher exponent and higher soft margin constant did take very long time to train, this can be true when all the data or most of the data end up as the support vectors. Though the 'no free lunch' theorem suggests that no algorithm can be universally used for all problems but most of the time ensemble methods like Bagging with J48 outperforms SVM.

Accuracy Details:

Training Set:

- Accuracy: 84.5767%
- Error: 15.4233%
- Mean Absolute Error: 0.1542
- Confusion Matrix:

```
    a      b   <-- classified as
20154   4566 |   a = <=50K
 2149  16669 |   b = >50K
```

Test Set:

- Accuracy: 80.1364%
- Error: 19.8636%
- Mean Absolute Error: 0.1986
- Confusion Matrix:

```
    a      b   <-- classified as
10176   2259 |   a = <=50K
  975   2871 |   b = >50K
```

- Bagging with J48

  Bagging creates ensembles by "Bootstrap Aggregation", that is, by randomly re-sampling the data. Bagging reduces variance through building multiple models from samples of the training data, and having each model in the ensemble vote with equal weight.

  On implementing J48, we got test set accuracy as 85.2159% (around 13847 correctly classified test records – the highest accuracy achieved amongst all classifiers which did not use ensemble methods). Hence, we implemented bagging using classifier J48.

  During the implementation, we noticed that the test set accuracy increased as we increased the bag size (starting from 1%), attained a maxima at bag size = 27% and 28% of the training data, and then started decreasing. The highest test accuracy achieved was 85.3756% (13890 records), representing an additional 43 correctly classified records.

  Bag size has an effect on accuracy. At small bag sizes (such as 1%), the classifier overfits the training data and has poor accuracy in test data. As bag size increases, overfitting is reduced and accuracy increases until an optimal size is reached. Further increase in bag size reduces the accuracy as the bag contains a large portion of the training data reducing the impact of bagging.

  Accuracy Evaluation:

  Training Set:
  - Accuracy: 89.8434%
  - Error: 10.1566%
  - Mean Absolute Error: 0.1544
  - Confusion Matrix:

```
    a        b     <-- classified as
22986     1734  |    a = <=50K
 2688    16130  |    b = >50K
```

  Test Set:
  - Accuracy: 85.3756%
  - Error: 14.6244%
  - Mean Absolute Error: 0.1985
  - Confusion Matrix:

```
    a        b     <-- classified as
11443     992  |    a = <=50K
 1389    2457  |    b = >50K
```

- AdaBoost with J48

  AdaBoost is an ensemble based meta-classifier and the algorithm begins by assigning same weight to all the instances in the training set. The key idea behind the classifier is to reassign more weight to misclassified instances. This step is repeated m times and hence, m different classifiers are generated. Finally, we take a weighted vote of all the m learners and output the vote. AdaBoost is known for having good generalization i.e. it doesn't usually overfit the data. The classifier used for boosting is J48. Decision trees are non-linear and Boosting with linear model doesn't work well. AdaBoost (with decision trees as the weak learners) is often referred to as the best out-of-the-box-classifier. When used with decision tree learning, information gathered at each stage of the AdaBoost algorithm about the relative 'hardness' of each training sample is fed into the tree growing algorithm such that later trees tend to focus on harder to classify examples. The weight threshold is taken to be 10.

  Accuracy Evaluation:

  Training Set:
  - Accuracy: 90.6289%
  - Error: 9.3711%
  - Mean Absolute Error: 0.1829
  - Confusion Matrix:

```
    a      b    <-- classified as
23323   1379  |   a = <=50K
 2683  16135  |   b = >50K
```

  Test Set:
  - Accuracy: 85.2159%
  - Error: 14.7841%
  - Mean Absolute Error: 0.2275
  - Confusion Matrix:

```
    a     b    <-- classified as
11515   920  |   a = <=50K
 1487  2359  |   b = >50K
```

- **Ensemble Learning**

  Once we build all the classifiers, we run these classifiers for all test instances and take a majority vote of the predicted class by all classifiers.

Highest accuracy attained amongst all the classifiers mentioned above was 85.3756%, that is, 13,890 records. By implementing majority vote amongst all the classifiers, the correctly classified test instances increased to 13,963 (85.7625%). Hence, the majority vote increases the number of correctly classified instances by 73.

Summary of the Classifier Accuracy analysis:

Algorithm used	Training Set accuracy (%)	Test Set Accuracy (%)	Mean Absolute Error for Training set	Root mean Squared Error for Training Set	Root relative squared Error for Training Set (%)	Mean Absolute Error for Test set	Root mean Squared Error for Test set	Root Relative Squared Error for Training Set (%)
J48	90.6289	85.2159	0.1412	0.2657	53.6341	0.1955	0.3288	70.2897
Naïve Bayes	83.0079	83.6804	0.1869	0.3635	73.3781	0.1719	0.3661	78.259
Bagging with J48 (bag size = 28%)	89.7813	85.3412	0.1551	0.2721	54.9369	0.199	0.3211	68.6301
AdaBoost with J48 (weight Threshold - 10)	90.6289	85.2159	0.1829	0.2909	58.7222	0.2275	0.3572	76.3539
SVM (C= 0.1)	84.5767	80.1364	0.1542	0.3927	79.2758	0.1986	0.4457	95.2726
KNN (K=11)	88.35%	83.0723	0.164	0.2842	57.36%	0.2142	0.3462	74.0143

STEP III: ACCURACY 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:

Training Set:

- Correctly Classified Instances: 39,282
- Misclassified Instances: 4,256
- Accuracy: 90.2246%
- Error: 9.7754%

Test Set:

- Correctly Classified Instances: 13,963
- Misclassified Instances: 2,318
- Accuracy: 85.7625%
- Error: 14.2375%

As we see, the training set accuracy is ~90%. This shows that the training data is not overfitted.

THANK YOU