2009 UC San Diego Data Mining Contest

Participated as a Machine Learning Cpt S 570 Course Project

Group Name:

Learner

Participants:

Barnan Das

Yasamin Sahaf

School of Electrical Engineering and Computer Science

Washington State University, Pullman

Best Score (Lift): **3.285**

Number of Submissions: **188**

Project Description:

In this project a data set of e-commerce transactions is provided. The goal is to classify the anomalous transactions and legal transactions.

There is no information provided about the dataset. Data set has 20 attributes, most of them are real, 2 of them are string(email address and state). By just looking at dataset, we can observe some information about a few attributes but not all of them.

Since the data set is not huge, and Weka software has most of the well-known algorithms , we decided to use Weka for most part of our experiments.

Preprocessing on data set:

When we first tried to use the data obtained from the UCSD site, we faced problem to run it in Weka directly. It was because of the attributes "state1" and "custAttr2", which is basically email Ids., which are string. Weka does not deal well with sting attributes, (though it has String to Vector Filtering Method, but just changing it to vectors is not enough. We tried it and we realized that it needs a lot of preprocessing). As a result, for solving this problem we did feature selection and removed 3 of the attributes (state1, custAttr1 and custAttr2).

We ran some algorithms on this data set. (Results are in the attached performance spreadsheet).

In the second part of our experiments we tried removing only email and state that were strings. Since customer attribute 1 is likely to be costumer ID, and each customer has a unique ID and email address, we assumed that removing email won't affect the result, because we still have costumer ID.

We also realized that flag 2, 3 and 4 are only 0 or 1. So we decide to treat them as nominal attributes instead of numeric.

Algorithms that we used and why

Bayesian Classifiers(bayes in Weka): The reason we wanted to use this classifier for the very first time was because of its simplicity. As it deals with combining prior knowledge with observed data, we expected it to perform really well. But, we found that this class of classifiers do not perform very well when they deal with a lot of discrete data. We think, that is the reason why they are more successful in the area of Information Retrieval

Instance Based Classifiers(lazy in Weka): We considered this class of classifiers because we had a lot of training data and also the number of attributes was 18(after filtering). But it didn't seem to perform very well. Moreover, they were very slow and some took even more than 10 hours to produce the predictions.

Artificial Neural Networks(functions in Weka): As the data set is highly discrete and real valued, we considered using this. The results were indeed good giving lift more that 3. But, the changes we wanted to do in the parameters didn't effect much, even change in Learning Rate did no good.

Kernel Based Methods(functions in Weka): As this class along with Support Vector Machines are considered to be the best performing classifiers among many learning problems, we were looking forward to its performance. But, unfortunately, it didn't perform very well. SMO took more than 2 days to produce the predictions. That is the reason why when playing with parameter changes, we planned to focus on better performing alorithms, like the ones of Ensemble Method class, rather than the ones which are expected to be better theoretically.

Ensemble Methods(meta in Weka): This was the best class of classifiers, which was expected though. Most of them gave lift higher than 3. As these algorithms conduct a weighted voting on many other algorithms, they usually perform better than many others. This was also proven when we developed our own ensemble algorithm, which was an ensemble of other ensembles used in weka, indeed. In order to reduce redundancy of the component classifiers of this ensemble, we also used some others from the decision tree class.

Artificial Immune System(Weka plug in used): These are computational systems inspired by the principles and processes of the vertebrate immune system. We had not used these algorithms any where before, but reading about these when we found that they perform really well for some learning problems. Because Immune systems are actually doing classification among body cells and non-body cells(like viruses). Artificial Immune systems are inspired by these features, so they were doing the same thing that we wanted to do in this project (distinguishing between fraud and legal transactions). That was why we expected AIS to perform good in this task. But found that, it performed even worse than the rule based classifiers. We downloaded the plug-in from this link:
http://sourceforge.net/projects/wekaclassalgos/files/

(AdaBoostM1 in Weka): In boosting it is running a Learning algorithm on training set, it will make some mistakes, the examples that it made mistake on will boost their importance. Each time it is running the algorithm, it will focus on the mistakes of the previous one.
AdaBoostM1 method, considers the geometric mean of posterior of instances inside a bag (arithmatic mean of log-posterior) and the expectation for a bag is taken inside the loss function.

AdaBoostM1 gives us acceptable result but its lift is still less than 3. We believe the problem with Boosting is that the Learning algorithm should know which examples are more important in order assign correct weights to them. In other words, it should have prior probability on instance space.

(Weka Misc): This Classifier has two algorithms in Weka: HyperPipes and VFI.
For each category a HyperPipe is constructed that contains all points of that category (essentially records the attribute bounds observed for each category). Test instances are classified according to the category that most contains the instance. But it did not work well on out data, the lift was even lower than 1.
We believe the reason was that this algorithm is extremely simple, it has the advantage of being extremely fast. In our case because of the small size of dataset, this advantage was not helpful for us. We need a more complex one.
VFI(Class implementing the voting feature interval classifier) For numeric attributes, upper and lower boundaries (intervals) are constructed around each class. Discrete attributes have point intervals. Class counts are recorded for each interval on each feature. It is doing the classification by voting. Weak point here is that it ignores Missing values. It does not handle numeric class (so it works in our case). The result was better than HyperPipes, more than 2.5.
It also has simple attribute weighting scheme. Higher weight is assigned to more confident intervals, where confidence is a function of entropy:
weight (att_i) = (entropy of class distrib att_i / max uncertainty)^-bias.

Decision Rules (Rules in Weka): In general rules classifiers didn't do very well on our data. We believe the reason was the fact that concept learning approaches does not work well with discrete attributes because they are generating rules. And in our data set we had discrete attributes.
Conjunctive Rule: We used this algorithm because it implements a single conjunctive rule learner that can predict for numeric and nominal class labels and we have nominal

class labels.
It is also using Reduced Error Pruning(REP) to prune the generated rule.
In pruning, weighted average of accuracy rate of the pruning data is used for classification while the weighted average of the mean-squared errors of the pruning data is used for regression.

Decision Table, It was the weakest among the rules. The reason is that it is using a simple decision table majority classifier. But **DTNB** did better. Because it is using a hybrid classifier: decision table/naive bayes. At each point in the search, the algorithm evaluates the merit of dividing the attributes into two disjoint subsets: one for the decision table, the other for naive Bayes. A forward selection search is used, where at each step, selected attributes are modeled by naive Bayes and the remainder by the decision table, and all attributes are modeled by the decision table initially.
Jrip was a little better than Decision Table but still worse than DTNB. It implements a propositional rule learner, Repeated Incremental Pruning to Produce Error Reduction (RIPPER).
In general Decision table, Conjunctive rule, Jrip, OneR had approximately the same results.
The best classifier among the rules was PART. (That is using partial tree structure that can be pruned using a pruning set.)

Decision Tree (Tree in weak): We chose Decision Trees because in our data set the target function is discrete (Fraud or Legal). And disjunctive hypothesis maybe required.
So, our data set had most of the characteristics for getting good result out of Trees. They actually did well in our experiments. Most of them gave us lifts more than 3.
We have tried different Decision trees like: ADTree, J48, NBTree, LAD, LMT, Decision Stump

ADTree: An alternating decision tree consists of decision nodes and prediction nodes. Decision nodes specify a predicate condition. Prediction nodes contain a single number. ADtree gave us better result than J48. Because in AD Tree an instance is classified by following all paths for which all decision nodes are true and summing any prediction nodes that are traversed. But in **J48,** an instance follows only one path through the tree. Moreover, we can boost the accuracy by increasing the number of iterations, this value should be tuned based on the data set that is used. So we tried different values for it, to find the best one that suit our data set. The best was 18.

Algorithms chosen for playing with the Parameters and why?

EnsembleSelection(meta in Weka): This was the one that gave 3.159 lift and hence was a natural choice for playing with the parameters. Although an “Optimise with Accuracy” for the “hillClimbMetric” parameter was expected to give higher performance, but it was found that “Optimise with all Metric” performed better. This can be the case for data sets and classifiers about whom one can't tell how it performed just by looking at the accuracy. Some times, precision, recall and f measure also determine their performance or more specifically “effectiveness”. But, the default RMSE(root mean square error), seem to perform the best. Though we tried to play with the algorithm that would optimize the ensemble, like Forward Selection and Backward Elimination, we found Forward Selection does the best among them. The number of hill climbing iterations didn't make any difference.

LogitBoost(meta in Weka): LogitBoost gave an impressive 3.178 lift with default parameters. Changes made with the “classifier” and “numIterations” failed to give any better result. Seems that the default “classifier” Decision Stump gives best results as LogitBoost should use a regression scheme as the base learner. But, setting “resampling” to true gave better result that default parameters all together. Thus, it can be concluded that resampling in case of ensembled method is a better option than reweighting. Therefore, the highest lift obtained by using the classifiers of weka on was 3.258.

Bagging(meta in Weka): The lift obtained by using the default parameters of bagging was 3.083. Though the changes made to the "classifier" made no difference, the change in "numIterations" did. It is also obvious that when we increase the number of iterations, the performance increases. But, increasing it too much makes Weka run out of memory. We had used atmost 4GB heap size for JVM and found that it cannot even work with 20 iterations. And increasing number of iterations does not always gives better result, for each data set there is one best setting that should be found by trying different values. Setting "numIterations" to 12 gave the best outcome of Bagging as 3.167 lift. It also gave some what good performance when "calcOutOfBag", i.e. whether the out-of-bag error is calculated or not, is set to true.

(AdaBoostM1 in Weka): This classifier is one the classifiers that gave us good result. The default lift is : 2.937.
We made use of ADTree to improve the accuracy of AdaBoostM1.
AdaBoostM1 uses decision stumps as weak hypotheses. Boosting decision stumps creates a set of T weighted decision stumps (where T is the number of boosting iterations), which then vote on the final classification according to their weights. As Individual decision stumps are weighted according to their ability to classify the data.
When we used AdaBoostM11 with the default parameters (Decision stump as the classifier and T=10), the lift was less than 3. Because when it uses a simple learner like Decision stump, it results in an unstructured set of T hypotheses. As a result, it is harder to find correlations between attributes. But by using ADTree, the set of hypotheses will get structured and can be visualized in a tree based on the relationship between a hypothesis and its parent. By using ADTree the lift got improved to more than 3.
Moreover, we can boost the accuracy by increasing the T (number of iterations), this

value should be tuned based on the data set that is used. So we tried different values for T, to find the best one that suit our data set. The result was T=18 that gave us lift of 3.143 which in comparison with the default lift (2.937), the improvement was noticeable.

(Bayes): Among Bayes classifiers, BayesNet was the best, so we tried different classifiers as its search algorithm. With Simulator-Anealing chosen as search algorithm it provides best result. In comparison with other algorithms it was acceptable(2.996)

Other Approaches

1- Ensemble Method:

Description of Ensemble method:

The ensemble method we have designed, basically does its work on a selected classifier outputs chosen on the basis of their best performance.

Algorithm:

1. Take probability predictions of n classifiers denoted by $\boldsymbol{classifier_j}$ containing 50000 of the predictions, denoted by $\boldsymbol{prediction_i}$.
2. Assign $\boldsymbol{weight_j}$ to these classifiers on the basis of lift obtained. That way, classifiers with higher lift get higher weight and with lower lift get lower weight.
3. Find the ensembled prediction $\boldsymbol{ensemble_i}$ by:

$$ensemble_i = \sum_{j=1}^{n} (prediction_{ij} * weight_j)$$

4. $\boldsymbol{ensemble_i}$ from 1 to 50000 is our prediction.

Calculation of weight:

If $\boldsymbol{lift_j > mean(lift_j)}$

$$weight_j = (lift_j / \sum_{j=1}^{n} lift_j) + k$$

else

$$weight_j = (lift_j / \sum_{j=1}^{n} lift_j) - k$$

In our case we found that k=0.03 gives the best result. But, this value of k is dataset dependant and the optimal value can only be found empirically.

Following is a list of the classifiers of Weka we have used for this ensemble:

Classifier	Parameters	Lift
Ensemble Selection (Ensemble Method)	Algorithm=Best Model, hillClimbMetric=Optimize with Accuracy	3.060
Ensemble Selection (Ensemble Method)	Algorithm=Forward Selection + Backward Elimination	3.119
Ensemble Selection (Ensemble Method)	Algorithm=Backward Elimination	3.127
Ensemble Selection (Ensemble Method)	Algorithm=Forward Selection, hillClimbingMetric=Optimize with all Metrics	3.131
LogitBoost (Ensemble Method)	Default	3.178
LogitBoost (Ensemble Method)	Shrinkage = 2.0	3.111
LogitBoost (Ensemble Method)	useResampling=True	3.258
LogitBoost (Ensemble Method)	WeightThreshold=200, useResampling=True	3.258
LogitBoost (Ensemble Method)	Seed=5, useResampling=True	3.206
Bagging (Ensemble Method)	Default	3.083
AdaBoostM1 (Ensemble Method)	NumIterations=18, classfier=ADTree, numOfBoostingIterations=18	3.123
AdaBoostM1 (Ensemble Method)	NumIterations=18, classfier=ADTree	3.143

AdaBoostM1 (Ensemble Method)	NumIterations=20	3.052
ADTree (Decision Tree)	NumIterations=20	3.163
LADTree (Decision Tree)	Default	3.167
LMT (Decision Tree)	Default	3.008

We decided to choose these set of algorithms because they were the best in the lot. But, we also considered some other algorithms, which didn't perform well, to avoid redundancy of classifiers.

We also tried play with double level of Ensemble, though we don't know whether it is right to do or not. In this process, the prediction files that we got by using different values of k were considered for doing the second level of ensemble. But this experiment didn't give any remarkably better performance.

2- Neural Data Mining for Credit Card Fraud Detection

Paper:

http://www.informatik.uni-frankfurt.de/asa/papers/ICTAI99.pdf

Reference number 4 of the paper:

http://www.informatik.uni-frankfurt.de/fbreports/fbreport07-99.pdf

In order to get better results we tried to implement an algorithm which is not a normal one, and is specifically for detecting credit card anomalies.
But, unfortunately we didn’t get good results from it.

Title: Neural Data Mining for Credit Card Fraud Detection

This paper has introduces a new approach for Credit Card Fraud detection. Normal algorithms like the ones we used in Weka focus on numeric data or symbolic data. So we can't consider both of them together. The other problem we had with string values made us ignoring them. But these attributes can have impact on predications. So we chose this paper to consider and focus on all features, nominal and symbolic This paper has focused on symbolic data first, and then works on nominal data using other normal algorithms like neural nets. And then combining the results.
Their main concept relies on the fact that we can make rules for predicting the fraud data. It would be like an IF then Else statement by using symbolic features.

They have presented an algorithm to produce these rules. They have started by making a dataset of fraud transactions. (In our case there were 2654) . Since we will only work on the symbolic features, we made a new dataset of the fraud data consisting of only symbolic features. We defined 7 features as symbolic. Although only two of them are strings, but we realized that some attributes only have values of 0 and 1. So we can consider them as symbolic features.
The attributes are:
State1, CustAttr2, Indicator1, Indocator2, Flag1, Flag2, Flag3, Flag4

These transactions make the leaves of the tree, then each transaction is compared with all others rules are generated for them. For example with instances: x1= (F,D,C,D,A) and x2= (F,D,G,D,A) we will generate: (F,D,*,D,A) (using '*' as don't care value).
So, we are generating a tree, and as we go up the tree one star will be added to the rules in that level. (level 1 has one star, level 2 has two stars and…)
In our case we had 7 features, so we could do the generalization till level 6.
When we generate a new rule, we can delete the rules that produced it. Because this new

rule has all the information in it. Or we can leave them as they are (What we did in our implementation).

Figure bellow is taken from the paper and shows a sample tree.

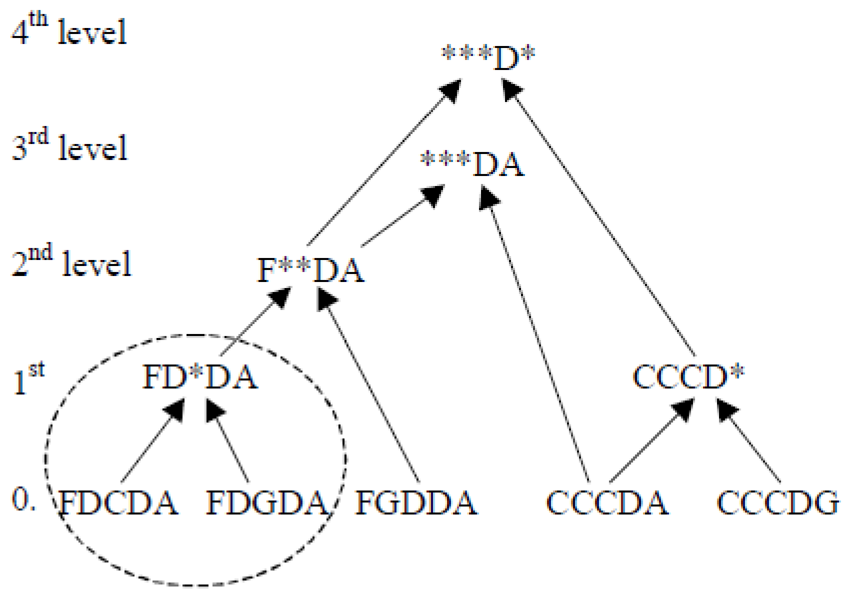


Fig. 3 *The generalization graph*

In our experiment our fraud data generated 172 rules in Level 1, 60 rules in Level 5 and 13 rules in Level 6.
All rules are different from each other in each level. In general, there are many rules in each level, they have defined the '**share'** of the rule as below:

$$\text{share} = \frac{\#\text{of fraud transactions covered by the rule}}{\#\text{of fraud transactions}}$$

It represent the percentage of the fraud data which is covered by the rule.
But, share does not reflect the fact that there are also legal transactions which may fit a

fraud rule. so they defined a new parameter as ‘confidence’:

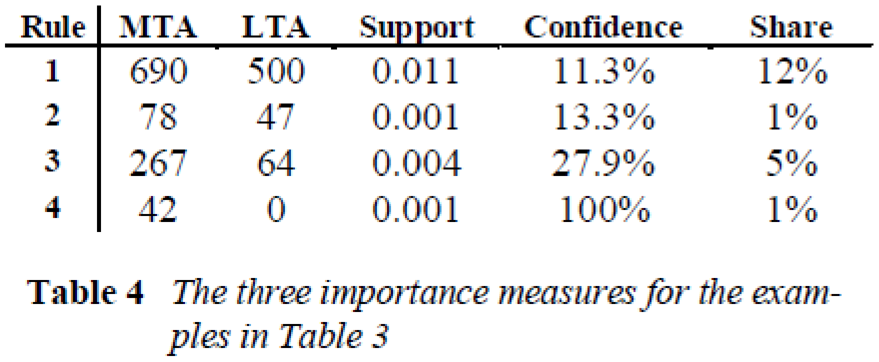
$$\text{confidence} = \frac{\#\text{of fraud transactions covered by the rule}}{\#\text{of transactions covered by the rule}}$$

Then confidence and share are calculated for each rule.
For calculating share and confidence because of high time complexity of the algorithm they have chosen a sample of legal data of size 30000. They had 5850 fraud transaction.
We chose a sample of 24339 legal transactions for our 2654 fraud transactions.
In the paper it is written that this rule based diagnostic system described can be implemented in many different manners. But it is not clearly stated which method they have used to classify new instances.
Then they compared share and confidence for each level and found the best result: level 4 and above.
In one of the references of the paper (reference number 4) it gives 2 implementation suggestions:
1- Diagnostic rules can be stored in conventional hardware based content addressable memory (CAM), Each time a transaction is fed to the CAM, one or several hits which eventually occur will indicate a fraud transaction.
2- Converting decision rules to a sequential decision procedure. The alarm is given when one of the rules are fulfilled.
In the first approach it is not clearly indicated how these hits occur, it is not clear whether each hit occurs by one of the rules or a combination of the rules result in an alarm. And if there is a combination, which combination should be chosen?!
So, we tried different ways that we could think of:
1- Using ensemble methods: assuming each rule as a hypothesis, for each new instance, classified it with all of the rules of that level, then take a vote among them.
We gave it all the fraud data as test set. But, this approach didn’t work, since most of the

rules were classifying instances as legal! So all the instances classified as legal instead of fraud. The reason is: each rule represents a part of the data, so each individual rule can't predict all fraud transactions. That is why we get low values for confidence of one rule.

In the table below we can also see that in the paper's experiments each rule alone has low confidence and share.

Note that for rule number 4, although the confidence is 100%, but it is not a good result, since share is 1%. So it means it only recognizes 1% of the fraud data.



Rule	MTA	LTA	Support	Confidence	Share
1	690	500	0.011	11.3%	12%
2	78	47	0.001	13.3%	1%
3	267	64	0.004	27.9%	5%
4	42	0	0.001	100%	1%

Table 4 *The three importance measures for the examples in Table 3*

2- In our second experiment we gave the merged data set(with legal and fraud data) and we listened to all of the rules. Means, for each instance even if one of the rules predicted fraud we would classify it as fraud.

Again this approach didn't work, although it gave us 100% share, but the confidence was very low.

There are some other approaches that we thought of, but didn't have time to implement them:

1- In the reference number 4 it is mentioned that in the mining algorithm they compare confidence and share of the rules and keep only the rules with sufficient confidence and share. This approach results in less rules in each level. And we are sure that all of them are good rules. But still we believe that adding this feature alone does not help and still we have to find an appropriate way of using rules of each level.

2- We can use different ensemble methods. Like Boosting approach, we can assign some weights to each rule(hypothesis) and the ones with higher confidence will get more weight(since their error was less during learning).

Conclusion

It was a great experience woking on this project. For future work we would like to do more preprocessing on the dataset, finding out new features and relations. We would also like to use some more algorithms like Genetic Algorithms.