Abstract

This project uses a series of learning approaches for a transaction anomaly detection problem. The input data is a 19-feature dataset of transaction records with high binary class skewness (1% vs 99%). We apply Logistic Regression, AdaBoost and Boosting Trees and we use Lift and prediction accuracy metrics to evaluate our results. All approaches were based on the material covered in the course notes [5] of this class.

1 Introduction

The problem tackled in this project is Transaction Anomaly Detection. In several web transactions, such as bank activities and exchanges, it is often the case that several anomalies may occur. These anomalies may be transaction failures, inconsistencies, break-downs, etc. All these cases are referred to as "anomalies". One would expect that anomalies in a set of transactions within a system are quite fewer than complete transactions, provided that the system is operating properly and does not contain system faults. This means that in a transactions dataset, anomalies are expected to be fewer than complete transactions, hence it is highly possible that the anomaly class is quite "skewed". What is more, in order for a transaction to happen to have an anomaly several factors have contributed to that, such as virus attacks, bad maintenance, simultaneous multiple wrong entries, while there are also "neutral" features, such as zip code and state of the web access region, staring and ending hours of transaction, etc. The latter can be as significant as the former in their contribution to an anomaly occurrence. All these factors are seen as characteristics (features) for each transaction and they are useful in the task of identifying if a transaction contains an anomaly or not.

1.1 Dataset

In our project we use a transactions dataset which consists of 19 discrete features, given by company FICO. The features are shown in the following figure. Along with the features the dataset contains also an additional feature which takes values 0 and 1, representing the class feature. The class is 0 when the corresponding transaction does not contain an anomaly, while it is 1 when it does. Overall the input contains 92498 transaction records, where a 70% of those is used as training set and the other 30% is used as test set. The former are used to get information and learn how class behaves as a result of the specific values of the rest features. We next use this "learning" in the test set, in order to predict the class of its records. In fact we know beforehand their class but we try to guess them using the model we constructed based on the training set, and then we compare the results to the original values. The more success values we get, the more accurate our model was.
In short, the goal of our project is to detect if the test set records contain an anomaly. The features of the dataset are shown below.

- Features

<table>
<thead>
<tr>
<th>amount</th>
<th>total</th>
<th>hour1</th>
<th>hour2</th>
</tr>
</thead>
<tbody>
<tr>
<td>field1</td>
<td>field2</td>
<td>field3</td>
<td>field4</td>
</tr>
<tr>
<td>flag1</td>
<td>flag2</td>
<td>flag3</td>
<td>flag4</td>
</tr>
<tr>
<td>indicator1</td>
<td>indicator2</td>
<td>state</td>
<td>zip</td>
</tr>
</tbody>
</table>

A special feature of this dataset is that its class is too skewed. Almost 98 – 99% of the data have class 0 and only 1% has class 1. This requires attention in the learning model evaluation step as discussed later.

2 Data Pre-processing

Some of the features described above are numeric and some are nominal. Nominal features do not preserve any numeric order, hence there is no meaning behind their number values. We turn them into binary features, consistently keeping the exact information they carry. For example, take feature State with domain: {AL, AK, AZ, AR, CA, CO,..., WY}. When we turn it into binary, we get a new feature (features set), S with domain: {0,1}, where S = {State = 'AL', ..., State = 'CA',..., State = 'WY'}. So we have one featured replaced by a binary one, or, equivalently, by 52 new features:

1 feature replaced by 52 features

\[ x(\text{State}) = \text{CA} \]

\[ x(\text{State} == \text{CA}) = 1 \]

\[ x(\text{State} == \text{AZ}) = 0 \]

\[ ... \]

\[ x(\text{State} == \text{WY}) = 0 \]

Analyzing all nominal values according to the above binary scheme, we end up with 1028 features instead of 19.

Another interesting factor is how informative features are. In order to get a closer look to the class behavior we plot the histogram of class 1 for each feature in Weka (Isolate & plot records of (class 1)) and we compare it to the total classes histogram (plot all records (class 1 and class 0) togehter). As we see in the first result of the following figure, "field1" seems to be more informative than "field2". This is because the distribution along the values of "field1" are not the same for class-1 only and for all classes together. This means that some features carry more information about anomaly than others.

To identify informative features we use a t-statistic metric. Figure 1 shows the most informative features along the set of numeric features only and figure 2 shows the most informative ones overall.

In an outline of the organization of this project, in Section 2 we first describe Lift, a common technique we use to evaluate the algorithms and to be able to compare their performance. Then we describe our main approaches to the above problem; In Section 3 we describe Linear/Logistic Regression used for classification and in Sections 4 and 5 we describe boosting approaches; Section 4 contains AdaBoost, while Section 5 contains Boosting Trees. In Section 6 we compare the above algorithms performace and extract general conclusions.
Figure 1: Informative Numeric Features

Figure 2: Informative Features (overall)
3 Model Evaluation

There are several methods for evaluating a learning model. A good evaluation metric is prediction accuracy. This metric is based on the average performance of the model on the test data and is defined as:

\[
\text{prediction accuracy} = \frac{\text{#correctly predicted test set records}}{\text{#total test set records}}
\]

This metric is not always useful for evaluating learning algorithms. One reason is that we are never certain about the nature of the test data, as they are unseen data. In some cases though we can guess their behavior up to some point based on the training data. Another case is when the learning algorithm depends on the distribution of the class values along the data records. Then the prediction accuracy may be high without representing the predictive power of the model.

A good example for the latter case is the input dataset of our problem. We have binary class (domain \{0, 1\}) representing the absence/presence of an anomaly respectively, and the size of the dataset is \( n = 92498 \). If the distribution was:

\[
\text{class} = \begin{cases} 
1, & \{1, ..., \frac{n}{2}\} \text{records} \\
0, & \{\frac{n}{2} + 1, ..., n\} \text{records}
\end{cases}
\]

then we may use the prediction accuracy as defined above, since for a prediction 90%, a big percentage of both class values are included in this quantity.

But in our case the distribution is:

\[
\text{class} = \begin{cases} 
1, & 1\% \text{records} \\
0, & 99\% \text{records}
\end{cases}
\]

i.e the data class is too skewed, hence an accuracy of 99% could predict all test records to have class 0 while leaving out all 1s. In practice it would not include class-1 in the class prediction.
To avoid this case we use another metric, called Lift. Lift is defined as

\[ \text{Lift} = \frac{a}{b} \]

and is calculated as follows.

Consider that when we run the model on the test data we get probabilistic results, i.e. values in \([0, 1]\) showing the probability that class is 0 or 1, \(Pr[\text{class}(\text{record}) = 1]\). In our case we consider that probability 1 means record has class 1 and probability 0 means it has class 0.

We run the model on the test set and sort all prediction results. In the sorted list the higher probability records are more possible to have class 1. Then we take the top 20% records of this list and calculate the 1s, i.e. records having class 1. We call this quantity \(a\), and in this point we have defined the nominator of the lift fraction:

\[ a = \text{rate of 1s on top-20%-probability records} \]

Note that the percentage 20% is defined arbitrarily, based on the percentage of probability values which are close to 1. Then we calculate 1s within the whole test set and we call this quantity \(b\). This defines the lift denominator:

\[ b = \text{rate of 1s along all test set records} \]

The lift method can be seen as an attempt to "catch" 1 values which are probably missed in the prediction accuracy method. Of course all these assumptions are not granted for all algorithms. For some algorithms, such as AdaBoost, where the results are not influenced by the class distribution, prediction accuracy works as well as explained below.

4 Logistic Regression

We apply Logistic Regression using regularization with two-norm squared and do batch gradient descent to minimize our objective function. The objective is:

\[
inlim_{\lambda \to 0} \lambda \|w\|^2_2 + \sum_{t=1}^{T} \frac{Loss(y_t, \sigma(w \cdot x_t))}{T}
\]

where \(\lambda\) is the regularization parameter, \(T\) is the number of records (in this learning approach we used a subset of 2000 records with sampling instead of the original dataset), \(w\) is the features weight vector, \(y\) is the real class vector of the test set, \(x\) is the features vector, and \(\sigma\) is the sigmoid function, as described below:

\[
f(z) = \sigma(z) = \frac{1}{1 + e^{-z}}, z = w \cdot x_t = \sum_{i=1}^{n} w_i x_{t_i},
\]

where \(n\) is the number of the features, \(x_{t_i}\) are the features for transaction \(t\) and \(w_i\) are their weights (regression coefficients).

Our stopping criterion in the gradient descent is until the 1-norm of the objective gradient becomes less or equal to \(10^{-4}\), i.e.:

\[
\text{norm}_1 = \|2\lambda w + \sum_{t=1}^{T} \frac{Loss(y_t, \sigma(w \cdot x_t))}{T}\|_1 \leq 10^{-4}
\]

The logistic loss function is defined as:

\[
Loss(y, \hat{y}) = y \ln(\frac{\hat{y}}{y}) + (1 - y) \ln(\frac{1 - \hat{y}}{1 - y})
\]
and with $\hat{y} = \sigma(w \cdot x)$ we get:

$$\text{Loss}(y, \hat{y}) = \text{Loss}(y, \sigma(w \cdot x)) = \ln(1 + e^{-wx}) - ywx$$

Since our dataset class is skew (only 1% class-1 values), we use different weighting for class-1 records and class-0 records. For class-1 we use weight $\beta = 0.7$ and for class-0 we use weight $\beta = 0.3$. The batch gradient descent algorithm we apply is described below (precision $10^{-4}$):

**Initialization:** $n = \# \text{ features}, T = \# \text{ transactions}, w_i = 0, 1 \leq i \leq n$

while $\text{norm}_1 > 10^{-4}$ do

$\hat{y}_t = \text{sigmoid}(w \cdot \beta(y_t)x_t)$, $1 \leq t \leq T$

where $\beta(y_t) = \begin{cases} 0.7, & \text{class}(y_t) = 1 \\ 0.3, & \text{class}(y_t) = 0 \end{cases}$

$\text{gradient} = 2\lambda w + \frac{1}{T} \sum_{t=1}^{T} (\hat{y}_t - y_t)x_t$

$w_{t+1} = w_t - \eta \cdot \text{gradient}$

Update $\text{norm}_1(\text{gradient})$

end while

Regularization prevents over-fitting on the training data and noise fit.

The lift results we get after applying logistic regression according to the above weighting scheme is $\text{lift} = 3.54$, while the average error on the predictions is $\text{avg.error} = 0.03829$. We keep these values to compare to the boosting results later.

## 5 AdaBoost

AdaBoost is a method which aims to predict records class focusing on the missed predictions of each previous step. In a short description, the algorithm starts with an initial weight distribution for all elements and on each step it calculates a new distribution based on the quality of predictions produced in this step. In this new calculation it assigns higher weight to those predictors which were wrong in the previous iteration. This proceeds for $T$ rounds.

We convert all 0,1 class values into ±1, and use the class column as the sign of each class. Positive sign means that class is 1, i.e transaction contains an anomaly, while negative sign means class is 0, i.e. no presence of anomaly. We used initial distribution $1/m$, where $m$ is the number of transactions in the reduced dataset, and we used the features as weak hypotheses:

$$h(x(t)) = x(t), \forall 1 \leq t \leq T$$

where $T$ is the number of features and $h_t(x) \in \{-1, +1\}$ is the sign that each learner $x$ predicts for each transaction $t$. In order to get better performance, we added as learners also the negation of the features. Hence our set of features is:

$$h(x(t)) = \begin{cases} x(t) & \text{if } 1 \leq t \leq T \\ -x(t) & \text{if } T + 1 \leq t \leq 2T \end{cases}$$

In total we got 2052 learners and 2000 transactions. This allowed the use of “non-presence” of each of the 2000 features as predictors for the class, giving about 95% performance of AdaBoost. We implemented the generalized version of AdaBoost algorithm as described in [4]:

Given: $(x_1, y_1), ..., (x_m, y_m) : x_i \in X, y_i \in \{-1, +1\}$

Initialize $D_1(i) = 1/m$

for $t = 1, ..., T$ do

- Train weak learner using distribution $D_t$.  

}
- Get weak hypothesis $h_t : X \rightarrow R$
- Choose $a_t \in R$
- Update

$$D_{t+1}(i) = \frac{D_t(i) e^{-a_t y_i h_t(x_i))}}{Z_t}$$

where $Z_t$ is a normalization factor chosen so that $D_{t+1}$ will be a distribution.

end for

Output the final hypothesis:

$$H(x) = \text{sign}(\sum_{t=1}^{T} a_t h_t(x))$$

In our implementation the “Train weak learner” part is omitted as we directly use the features of the dataset. We run the above algorithmic for some iterations (rounds) and get the average loss based on the different $\alpha$ weights derived in each round. The average loss is defined as the number of transactions whose sign was not predicted correctly, i.e. which were not classified correctly as class 1 (anomaly presence) or class 0 (anomaly absence):

$$\text{Average Loss}(H, y) = \text{count}(H(i) - y(i)), \forall 1 \leq i \leq m$$

To tune the number of rounds needed to derive the minimum average loss we applied cross validation, which showed that the optimal number of rounds is 50, after trying 40, 50, 60 and 70 rounds.

The error on each step of the algorithm is shown below:

The average Loss of the algorithm for the training and the test sets is shown below:
AdaBoost results are not probabilistic, since the algorithm returns sign values $\pm 1$. In this case the answer cannot be used to calculate the lift as it was described in Section 3. As we see in [1] and [3], there have been implemented two AdaBoost extensions for supporting biased data, i.e. one-sided data. The first is InfoBoost and the second is SoftBoost. InfoBoost builds on AdaBoost, and SoftBoost build on InfoBoost, as shown in the figures below.

- **Update rule:**

  $$D_{t+1}(i) = \frac{D_t(i) \exp\{-y_i h(x_i) \alpha_t[h(x_i)]\}}{Z_t},$$

  where $\alpha_t[\pm 1] = \frac{1}{2} \ln \frac{1+\gamma_t[\pm 1]}{1-\gamma_t[\pm 1]}$ and

  $$\gamma_t[+1] = \frac{\sum_{i: h_t(x_i) \geq 0} y_i h(x_i) D_t(i)}{\sum_{i: h_t(x_i) \geq 0} D_t(i)},$$

  and $\gamma_t[-1] = \frac{\sum_{i: h_t(x_i) < 0} y_i h(x_i) D_t(i)}{\sum_{i: h_t(x_i) < 0} D_t(i)}$.

- **Final hypothesis:** Sign of $H(x) = \sum_{t}^{T} \alpha_t[h_t(x_i)] h_t(x_i)$

  ![Figure 4: InfoBoost](image)
6 BOOSTING TREES

The final hypothesis of InfoBoost can be written as linear combination of $h_t^+$s and $h_t^-$s,

$$H(x) = \sum_t \alpha_t[+1]h_t^+ + \sum_t \alpha_t[-1]h_t^-,$$

where $h_t^\pm(x) = \pm 1$ if $h_t(x) = \pm 1$, and $h_t^\pm(x) = 0$ otherwise.

SemiBoost: Instead of using the set of base hypotheses $H$, use

$$H^\pm = \{h^+, h^- | h \in H\},$$

and run AdaBoost

![Diagram](image)

Figure 5: SoftBoost

These algorithms would fit our input dataset since the class is characterized by strong bias in favor of class 0 and their experimental results would be very interesting. In particular, the method described in [3], called SoftBoost, studies InfoBoost described in [1] and proposes a biased approach on weights of the learners, by adding a constant vector $1$ to each iteration, besides the $h_t$ step. Because of time restrictions we are not including experimental results of these approaches in this project, but they consist a very interesting approach to apply especially on the kind of our dataset.

6 Boosting Trees

Gradient Boosting Trees

The idea of gradient boosting trees [2] is to compute a sequence of simple trees, where each successive tree is built for the prediction residuals of the preceding tree. This method builds binary trees, i.e., partitions the data into two samples at each split node. At each step of the boosting trees algorithm, a simple partitioning of the data is determined, and the deviations of the observed values from the respective means (residuals for each partition) are computed. The next tree is then fitted to those residuals, to find another partition that will further reduce the residual (error) variance for the data, given the preceding sequence of trees.

This idea is based on additive weighted expansions of trees which can eventually produce an excellent fit of the predicted values to the observed values, even if the specific nature of the relationships between the predictor variables and the dependent variable of interest is nonlinear, happens in our class-features case. Hence, the method of gradient boosting - fitting a weighted additive expansion of simple trees - represents a very powerful machine learning algorithm.

We run the algorithm in the whole 92498 records dataset multiple times. In each run we specify a particular penalty value $k$ which penalizes the missclassification of each class. Since our dataset target class is too skewed, we increase this penalty for missclassifying records of class 1, while we keep the penalty for missclassifying 0s low. We start with $k = 1$ and we increase it is successive runs, up to $k = 50$. As it is shown in figure 7, when $k$ is increased, the missclassification error for the class-1 data is reduced, while it is increased for the rest data. The total missclassification error seems to increase too, as it is highly influenced by the class-0 records, which are the majority in the dataset. The algorithm is shown below:

To evaluate the algorithm using the Lift metric described in Section 3, we produce the following graph.
Algorithm 1: Gradient TreeBoost

1. \( F_0(x) = \arg\min_y \sum_{i=1}^{N} \Psi(y_i, \gamma) \)
2. For \( m = 1 \) to \( M \) do:
3. \( \bar{y}_{im} = \left[ \frac{\partial \Psi(y_i, F(x))}{\partial F(x)} \right]_{F(x) = F_{m-1}(x)}, i = 1, N \)
4. \( (R_{tn})^\dagger = L- \text{terminal node tree} \ (\{\bar{y}_{im}, x_i\}_i^N) \)
5. \( \gamma_{tn} = \arg\min_{\gamma} \sum_{x_i \in R_{tn}} \Psi(y_i, F_{m-1}(x_i) + \gamma) \)
6. \( F_m(x) = F_{m-1}(x) + \nu \cdot \gamma_{tn} \cdot 1(x \in R_{tn}) \)
7. endFor

Figure 6: Gradient Boosting Trees Algorithm

Figure 7: Missclassification Errors Rate
In this experiment we calculate the lift produced for each of the different runs of different $k$ penalty values (ranging from 1 to 50). We notice that the best penalty value, for which the algorithm detects the most class-1 values is $k = 13$:

![Figure 8: Evaluation-Lift](image)

Finally, we evaluate the algorithm performance by examining the total Loss for the training and the test set. We see that after 70 iterations the test set error is reduced and remains almost stable until the $400-th$ iteration is reached (figure 9).

### 7 Conclusion

Based on the above experiments, we conclude that the skewed nature of our dataset ($class = 1 \sim 1\%$) required more tentative evaluation. Most classifiers return wrong results unless tuned with “prediction weighting” (e.g. Logistic Regression calculations fail due to big sparsity of class feature). By weighing the features weights in the sigmoid function according to the scheme below, we managed to get probabilistic results and calculate the lift. $\beta = \begin{cases} 0.7, & \text{for records with } class1 \\ 0.3, & \text{for records with } class0 \end{cases}$.

AdaBoost performed relatively well, although it can be tuned using the bias techniques of InfoBoost and SoftBoost described in [1] and [3]. This should yield very interesting results, which there was no time to include in this report. Overall Boosting Trees outperformed other classifiers (in both lift and error metrics), as this method predicted the highest lift of $3.986$ and the lowest average loss ($0.0205$).

### References


Figure 9: Evaluation-Loss
