ILLINOIS DATA SCIENCE INITIATIVE

TECHNICAL REPORT

Comparing Credit Card Fraud Detection between Spark and Spark Streaming

Author:
Arshia Malkani, Caren Zeng

May 2, 2017

Final Comparing Credit Card Fraud Detection between Spark and Spark Streaming

ARSHIA MALKANI, CAREN ZENG[1] AND PROFESSOR ROBERT J. BRUNNER[2]

[1] *National Center For Supercomputing Applications (NCSA)*
[2] *Laboratory for Computation, Data, and Machine Learning*

Compiled May 2, 2017

To predict credit card fraud accurately in real time using machine learning in Spark Streaming.

https://github.com/lcdm-uiuc

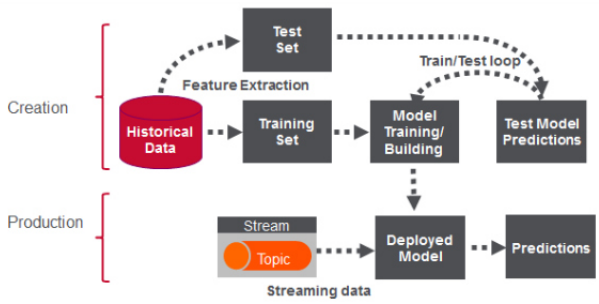
1. INTRODUCTION

Comparing different classification and regression machine learning models in Spark, we wanted to find the suitable algorithm to train data on credit card fraud prediction. Considering the massive amounts of credit card data though, we then wanted to then used the pipelining process of Spark Streaming combined with the machine learning algorithms to accurately predict credit card fraud on live data.

2. BACKGROUND

Credit card fraud detection is a hot field as a multitudinous host of classification, regression, and machine learning algorithms can be applied to the data, which provide results that can be very valuable in business and technology. Research focusing on credit card fraud detection can be split into real-time and static fraud detection. Static fraud detection involves splitting a dataset into training and test data, and building and testing a given model on such a dataset. Static fraud detection allows us to learn about the nature of credit card data and understand common anomalies. What is more applicable to companies is real-time fraud detection: usually, the model is trained with a "stream" of data, loaded in windows, and tested on data as well to create a model and results that update live. Real-time credit card fraud detection often follows the following schema:

The first phase involves analysis and forensics on historical data to build the machine learning model. The second phase uses the model in production to make predictions on live events. [include citation]



A common model to use for credit card fraud detection is a logistic regression model. Models are built using the following formula:

$$Y_i = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + \varepsilon$$

Read as: "AmntFraud = intercept+ coeff1 age + coeff2 claimedAmnt + coeff3 * severity + error."

The formula produces a probability that an interaction is fraud. Providing a range of data allows for an accurate regression model. However, different sets of data may not provide as many features, thus limiting how spread out the data will result to be.

Additionally, anonymized credit card data is usually severely unbalanced, as most credit card transactions are non-fraudulent. This is a feature of the field that makes research so interesting. Equivalently interesting fields include other types of fraud.

3. THE DATA

The dataset we was a csv file of 65535 rows of anonymized credit card data, including features such as the frequency of usage of the card, the transaction amount, and a binary value indicating fraud ("1") or non-fraud ("0"). The other 28 features in the dataset were the were the results of a PCA transformation, but due to confidentiality we don't know the original features and more background information. Principle Component Analysis, or PCA, is a statistical procedure that uses orthogonal transformations to convert the dataset of possible correlated variables into a set of values of linearly uncorrelated variables called principle components. The dataset was collected and analyzed during the research of Worldline and the Machine Learning Group of ULB in which the credit card transactions in September 2013 in Europe were recorded over a span of two days.

Before developing algorithms to predict features of our data, we analyzed our data to understand what its characteristics were and how they would influence building our models and how reliable each should be considered. Running a percentage-generating script on our dataset, it was apparent that the dataset was very unbalanced, with only 0.172% (492 out of 284,807 transactions) labeled as fraud. This skews our models as they could predict everything as not fraud and still have an incredibly low error rate. To compensate, we made sure that after every model,

we didn't just return the test error, but a confusion matrix and the Area Under the Precision-Recall Curve (AUPRC) to see if the predictions were meaningful in the real world. The goal of this project is to train the datasets to eventually handle live data, meaning that despite imbalanced datasets, the generated predictions must still be relatively accurate.

The data was stored in a csv file that needed to be mapped and filtered to remove the first line and turn the columns containing the amount of money spent, number of times the card was used, whether not it was fraud, and the PCA values into into a labeled point. Then we split the datatset 80-20 into a training and test dataset. We ensured that we were maximizing the training data due to the imbalanced dataset. To ensure that all the fraud cases didn't end up entirely on the training or test dataset, we ran each machine learning model multiple times with random splits and analyzed the confusion matrix. After the dataset is formatted and split, we began training the dataset with different machine learning models.

4. SPARK BINARY CLASSIFIERS: LOGISTIC REGRESSION

Credit card fraud detection comes down to a binary classification of fraud or not fraud, in which the categories are distinguished by a threshold. For instance, credit card transactions will be marked as fraud if the models predict fraud with greater than 90 percent probability. The validity of the threshold can be evaluated using a precision recall curve.

Precision is the number of true positives over the number of true positives plus the number of false positives. Recall is the number of true positives over the number of true positives plus false negatives. Both the precision and recall can be used to calculate the F score, which is 2 times time precisions times recall divided by the sum of precision and recall.

Available metrics

Metric	Definition
Precision (Positive Predictive Value)	$PPV = \frac{TP}{TP+FP}$
Recall (True Positive Rate)	$TPR = \frac{TP}{P} = \frac{TP}{TP+FN}$
F-measure	$F(\beta) = (1+\beta^2) \cdot \left(\frac{PPV \cdot TPR}{\beta^2 \cdot PPV + TPR}\right)$
Receiver Operating Characteristic (ROC)	$FPR(T) = \int_T^\infty P_0(T)\,dT$ $TPR(T) = \int_T^\infty P_1(T)\,dT$
Area Under ROC Curve	$AUROC = \int_0^1 \frac{TP}{P} d\left(\frac{FP}{N}\right)$
Area Under Precision-Recall Curve	$AUPRC = \int_0^1 \frac{TP}{TP+FP} d\left(\frac{TP}{P}\right)$

The precision-recall curve graphs the precision vs the recall at different thresholds. The area under the precision recall curve (AUPRC) is a value between 0 and 1, where the higher the number the better classifier. A higher AUPRC signifies that the model is at the upper right, meaning it is maximizing true positives and is a near perfect classifier.

The precision recall curve doesn't account for true negatives, which is what the ROC accounts for. The ROC curve graphs the number of false positive rates against the true positives rates. The area under the curve is between 0 and 1, where the higher the are is the more accurate the predictions are.

We decided to run run logistic regression on the dataset and then get the ROC and AUPRC values.

Logistic regression is a predictive regression analysis that fits the data points as if they are along a continuous function. It is used to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

```
training, test = data.randomSplit([0.8, 0.2])
training.cache()
model = LogisticRegressionWithLBFGS.train(training)
predictions = test.map(lambda lp:
(float(model.predict(lp.features)), lp.label))
metrics = BinaryClassificationMetrics(predictions)
print("Area under ROC = %s" % metrics.areaUnderROC
+ "Area under PR = %s" % metrics.areaUnderPR)
```

In our case, the ordinal independent variables would be the purchase amount, number of times the card was used, and the PCA values. The dependent variable would be the binary classification of fraud. Based on the data, logistic regression models try to draw a best-fit function which is used to predict whether or not its fraud.

Logistic Regression with LBFGS Results

Weighted Recall	0.99839
Weighted Precision	0.99834
Weighted F(1) Score	0.99837
Weighted F(0.5) Score	0.99835
Weighted False Positive Rate	0.49913

*seed for random split = 10

Trial	Area under the ROC curve	Area under the PR curve
1	0.6530227	0.2783491
2	0.6517289	0.2630561
3	0.6418172	0.2387083
4	0.6610563	0.2825437
5	0.6322924	0.2231790

*5 trials with random seeds

The right recall, precision, and f scores show that it was accurately predicting most of the fraud cases, but the false positive rate shows that there were a significant number of cases tagged as fraud incorrectly. False positives, in the scope of credit card data, aren't as alarming since they can be resolved through further investigation, but false negatives indicate that instances of fraud were completely missed.

We can understand more of what the data looked like through the area under the ROC curve 25 percent and the area under the PR curve 60 percent. It shows that the predictions were relatively accurate but can be improved using more training data or different machine learning algorithms.

5. SPARK: BINARY CLASSIFICATION : LINEAR SUPPORT VECTOR MACHINE

Linear SVM's algorithm plots the data point in the n-th dimension depending on how many variables we give it, and then fits a function (hyperplane) to separate the classes of data. The function it generates tries to maximize the space between the two classes, finding the widest margin. It tends to inaccurately predict data points near the decision boundary, which can be an issue if the data is all grouped at the boundary. Linear SVMs

tend to be accurate and work well on smaller, cleaner datasets in which the data splits evenly. It isn't suited for larger datasets with overlapping classes. With credit card fraud data, all the fraud cases aren't all going to be absurdly high or low prices, transactions, or other features. This means its harder to distinguish which ones are fraud since there aren't clear boundaries in the dataset. This makes it more challenging, but there should be general minute trends in the data that would enable us to actually classify which cases were fraud.

```
Linear SVM Results
        Area under the Precision Curve: 0.999154
        Area under the ROC Curve: 0.500002
```

Our results have a higher area under the PR curve, which means a higher precision and recall. A high precision is indicative of a low false positive rate and a high recall is indicative of a low false negative rate. This overall means that the linear SVM predicted fraud more slightly accurately that a logistic regression.

6. INTUITION BEHIND MACHINE LEARNING ALGORITHMS CHOSEN

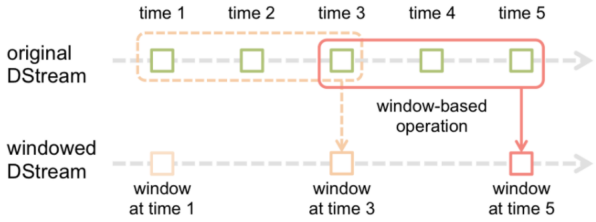
Due to the topic of credit card fraud detection, we are dealing with a classification algorithm to accurately predict fraud. While they are other classification models such as random forests and gradient boosted trees would have fit the data, there is no way to find the number of false positives and negatives. This made us inclined to focus on logistic regression and linear svm's as the PR and ROC curve can be generated. The next step would be using Spark Streaming while making predictions using these machine learning algorithms.

7. SPARK STREAMS

When hadoop and spark first emerged, it opened up opportunities to do computation on massive amounts of customer data. This enabled research toward analyzing credit card data, but the next stage was analyzing the data in real time. Every time a credit card is swiped, the request must be analyzed to confirm whether or not the purchase is legitimate. Due to the time-sensitivity of this process, this means the model needs to be pre-trained and the data needs to be sent in batches to the model to make predictions.

Spark streaming is an extension of spark that enables high-throughput, fault-tolerant stream processing of live data streams. This means the data is treated like a DStream, or sequence of RDD objects that are sent to the machine learning model in batches. Data can be taken from sources such as Kafka, Flume, Twitter, Zero MQ or TCP sockets and processed using processed using functions such as map, reduce, filter, join, and window. The processed data can be pushed toward file systems or databases.

Spark streaming accomplishes the batched data using windowed computation, in which the transformation is applied to a sliding window of data. In the picture below the window length is 3 and the interval is 2, but these parameters can be changed depending on the application.



The advantages of the spark streaming architecture are dynamic load balancing and resource usage, fast failure and straggler recovery, and performance. Spark streaming relies on a system of worker nodes each running on or more continuous operation. The "source" operators receive data from ingestion systems and the "sink" operators output to downstream systems. This improves load balancing, which is an issue caused by the uneven allocation of processing between the worker nodes. This bottlenecks operations since computation becomes dependent on a subset of the worker nodes that are taking longer. Stark streaming load balancing features help evenly distributes the work load between the nodes by dividing the data into small micro-batches and then allocating computation resources. If a node is taking longer due to an unevenly partitioned system, more nodes are allocated to that task as shown below.

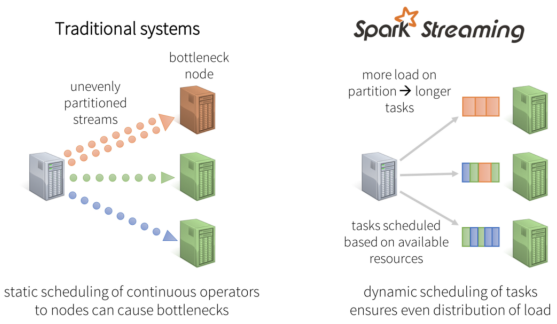


Figure 3: Dynamic load balancing

Spark streaming also enables fast failure and straggler recovery. With a larger scale, there is a high likelihood of a cluster failing or unpredictable slowing down(stragglers). Normally, the system would have to restart the failed continuous operator on another node and recompute the data. The computation in spark though is divided into small deterministic tasks, so failed tasks can be relaunched in parallel to the other. This recomputation is distributed across multiple nodes, which makes the recover rate significantly faster.

The distributed nature of spark streaming and the ability to leverage the spark engine improve performance. The micro-batching increases throughput, or the number of actions executed per unit of time. In terms of latency, or the time required to perform some actions, sparks streaming breaks it down to a few hundred milliseconds per action. Micro-batching inherently adds some latency, but it is only a small component of the pipeline that's automatically triggered only after a certain time period. This prevents it from having a significant effect on performance.

8. SPARK STREAMS : PREDICTIONS USING LOGISTIC REGRESSION

With Spark Streaming, the dataset is converted to a .txt file on which the data is trained normally in Spark. We tested this with both logistic regression and linear svms. The data is then reloaded as a stream from the .txt files. Using the transform function, the section data currently in the stream is converted into an RDD object and used to predict whether or not that instance is credit card fraud. Since the RDD object only contains a window of the data, it is more efficient and is able to take in live data to make predictions.

The code below shows how this is accomplished:

```
ssc = StreamingContext(sc, 1)
lines1 = ssc.textFileStream("file:///mnt/vdatanodea/
datasets/creditcards/credit/b")
trainingData = lines1.map(lambda line:
LabeledPoint( float(line.split(" ")[1]),
    [(line.split(" ") [0]),
    (line.split(" ") [2])])).cache()
trainingData.pprint()

lines2 = ssc.textFileStream("file:///mnt/vdatanodea/
datasets/creditcards/credit/c")
testData = lines2.map(lambda line:
LabeledPoint( float(line.split(" ")[1]),
[ (line.split(" ") [0]) ,
(line.split(" ") [2]) ])).cache()
testData.pprint()

def handle_rdd(rdd):
    for r in rdd.collect():
        print( r.map(lambda p: (p.label,
         p.features, lr.predict(p.features))) )

labelsAndPreds =
testData.transform(lambda rdd: handle_rdd)
labelsAndPreds.pprint()
ssc.start()
```

Ideally though, Spark Streaming should be used to predict live data, allowing users to immediately tag potential credit card fraud and address it in a timely fashion. The real world applications for Spark Streaming give it a huge advantage in the real world. The only disadvantage though is that one can't perform functions on the entire set of data (ie. sort) since there is only access to a certain portion of the dataset at a given time. As a result, depending on the type of computation needed, Spark Streaming can be may or may not be useful. When it comes to detecting credit card fraud in real time though, it can be extremely useful.

9. ANALYSIS

We did two levels of analysis: comparing different machine learning algorithms to train credit card fraud data and comparing Spark vs Spark Streaming.

We chose logic regression and linear svms since we were doing binary classification of whether or not it was fraud. We could use parameters to set the threshold that would differentiate it into two classes to determine to tag it as fraud or not. Upon initial review, it is apparent that while the test error rates were low, this may be due to the unbalanced nature of the dataset. In order to compensate, we used the area under the precision recall curve and the ROC curve to understand the number of false positives and false negatives. This indicated that there were a number of mis-flagged fraud cases, but this could be resolved through further investigation.

The data from the logistic regression did show that the model could be trained to predict fraud cases accurately enough that it could be applied to live data through spark streaming. This opened up many real world applications as there is a vast amount of credit card data that needs to be tagged in real time in order to prevent fraud. A high false positive rate may lead to too many investigations and a high false negative rate means that fraud cases aren't being tagged, meaning that both error rates can have significant implications. Spark Streaming with an accurate machine learning model would allow the right credit cards to be tagged in real time, which would allow companies to close credit cards and minimize damages.

The Spark Streaming research showed that we could train the model using our current set and then turn the test data into a stream to make predictions. The test data can be replaced with real data, and as more data is gathered the model can be retrained. Our research demonstrated how to accomplish spark streaming with credit card data and how spark streaming was more efficient in making predictions.

10. CONCLUSION

Through this exploration of many Spark mllib classifiers and Spark Streaming, we were able to find the most effective model for our dataset of credit card data, which is more or less representative of all anonymized credit card data. Our analysis and conclusions tied together the fundamental concepts of each model with our observed results, allowing us a deeper understanding of each model and an intuition of the pipeline process for Spark and Spark Streaming. Usages of this research include further credit card data research and learning about other imbalanced datasets, live or static, applicable to the current intersection of data science and machine learning at large.

11. SCRIPTS

The scripts are publicly available on github. https://github.com/CodeBrew28/CreditCardFraudResearch

12. SOURCES

1̂. Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson and Gianluca Bontempi. Calibrating Probability with Undersampling for Unbalanced Classification. In Symposium on Computational Intelligence and Data Mining (CIDM), IEEE, 2015

2."Apache Spark Streaming." Apache Spark Streaming | MapR. Accessed May 01, 2017. https://mapr.com/products/product-overview/apache-spark-streaming/.

3. Diving into Apache Spark Streaming's Execution Model. (2016, October 27). Retrieved May 01, 2017, from https://databricks.com/blog/2015/07/30/diving-into-apache-spark-streamings-execution-model.html

4. Evaluation Metrics - RDD-based API. (n.d.). Retrieved May 02, 2017, from https://spark.apache.org/docs/2.1.0/mllib-evaluation-metrics.html

5.Classification and regression. (n.d.). Retrieved May 02, 2017, from https://spark.apache.org/docs/2.1.0/ml-classification-regression.html