Project Report

Model Details:

Building A Streaming application to predict Credit Risk Score System with Kafka And Python

Context

It is important that credit card companies can recognize fraudulent credit card transactions so that customers are not charged for items that they did not purchase.

Content

The datasets contain transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features Attribute1, Attribute2, … Attribute28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.

Inspiration

Identify fraudulent credit card transactions.

Given the class imbalance ratio, we recommend measuring the accuracy using the Area Under the Precision-Recall Curve (AUPRC). Confusion matrix accuracy is not meaningful for unbalanced classification.

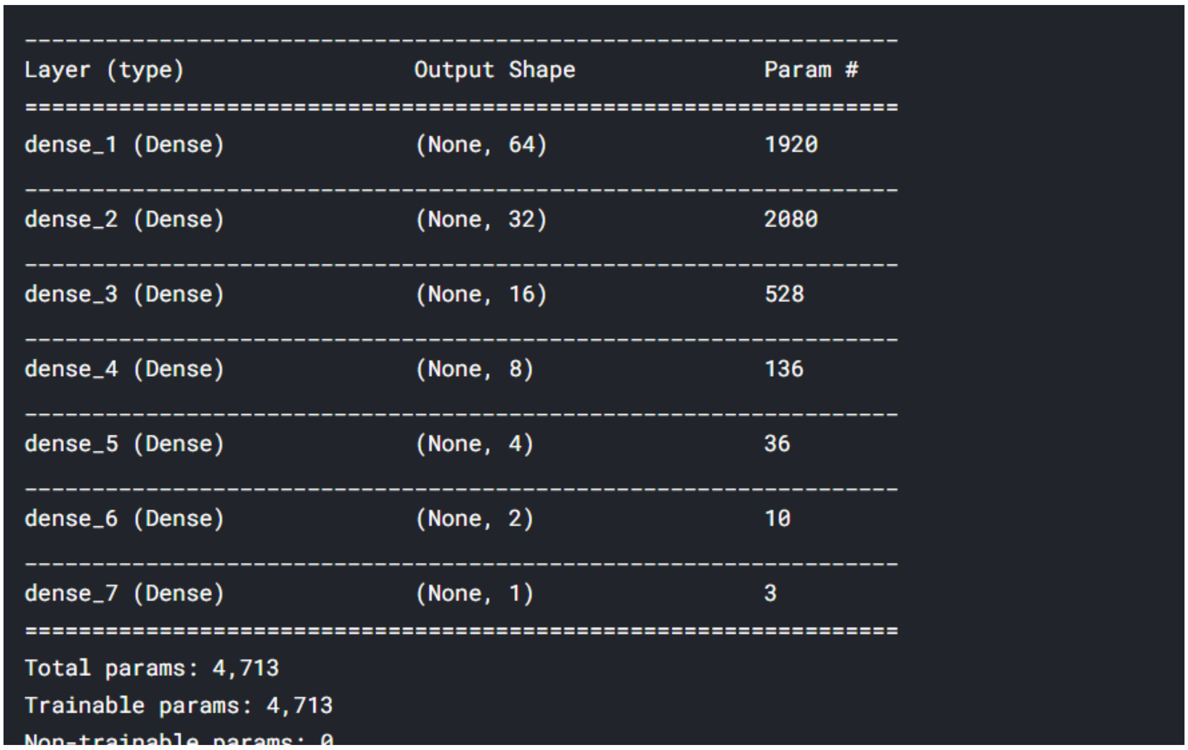
Data Pre-processing:

Due to the huge class imbalance in dataset, a model that simply identifies all transactions as not being fraudulent would score high accuracy. There also would not be many fraudulent samples for the model to learn from to be able to accurately identify what a fraudulent transaction is. Therefore, we should find a way to balance out the number of positive and negatives instances in our training set. This can be done by either oversampling the positive instances, or under sampling the negative instances. Under sampling the negative instances would involve reducing the number of non-fraudulent transactions until the ratio between positive and negative instances was approximately 1-to-1. Since we don't have that many data samples, I fear doing so would severely limit our model's performance, since it would have much less data to train on. We shall therefore oversample the positive instances in our training data.

To oversample the positive instances in our training set, we will add copies of them to it, but with their feature values slightly tweaked. This slight manipulation of the data is done so as to have the copies being different from their original counterparts, but not too different as to end up teaching our model false information. This tweaking will be done by multiplying each positive sample copy's feature values by a number between the uniform distribution of 0.9 and 1.1.

Neural Network Architecture:

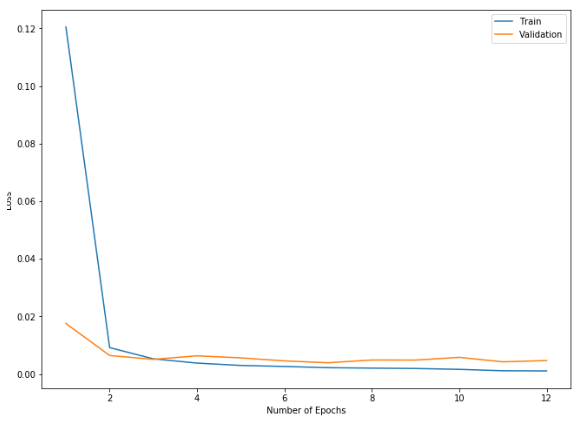
Train a feedforward neural network to detect fraudulent activity. The neural network will contain several layers of hidden units with ReLU activation functions, and a sigmoid output unit to output the probability of a given transaction being fraudulent. The Adam optimizer will be used with an initial learning rate of 1e-4 and a binary cross-entropy loss function. The model will be trained for a maximum of 50 epochs, though the learning rate of the Adam optimizer will be reduced by a factor of 0.1 if the validation loss does not improve after 3 epochs of training. This will continue until a minimum learning rate of 1e-6 is reached. If the validation loss does not improve after 5 epochs of training, we will halt training of the neural network.

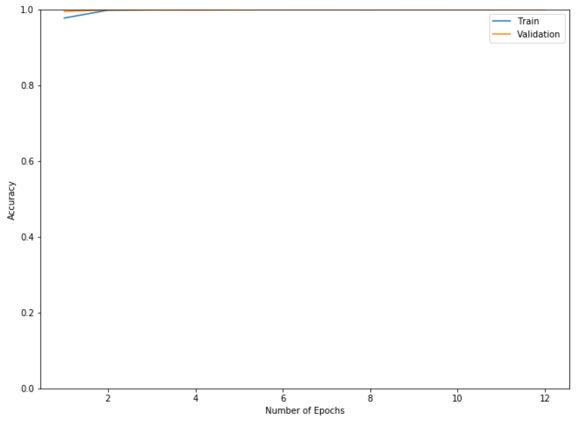


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 64)                1920
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_3 (Dense)              (None, 16)                528
_________________________________________________________________
dense_4 (Dense)              (None, 8)                 136
_________________________________________________________________
dense_5 (Dense)              (None, 4)                 36
_________________________________________________________________
dense_6 (Dense)              (None, 2)                 10
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 3
=================================================================
Total params: 4,713
Trainable params: 4,713
Non-trainable params: 0
```

Model Performance:

Achieving almost 100% accuracy on the test set.





Spin up the cluster!

go to your project root (where `docker-compose.yml` lives) and run:

```
$ docker-compose up
```

Prerequiste:

Faust: stream processing library, using the async/away paradigm and requires python 3.6+

Kafka: we will use the confluent version for kafka as our streaming platform

MLFlow: an open-source platform used to monitor and save machine learning models after training

Step1:

Serving the model as rest API:

mlflow.sklearn.save_model(model, path='./credit_card_score_prediction')

Step2:

start a terminal in the root of our project and type the following command:

mlflow models serve -m credit_card_score_prediction

Listening at: http://127.0.0.1:8088

Step3:

docker-compose up

The docker-compose will start zookeper on port 2181 , a kafka broker on port 8088

Step4:

faust -A predict_credit_score_worker worker -l info

Unit Test Case:

!curl http://127.0.0.1:5000/inAttributeocations -H 'Content-Type: application/json' -d '{

"columns":["Attribute1","Attribute2","Attribute3","Attribute4","Attribute5","Attribute6","Attribute7","Attribute8","Attribute9","Attribute10","Attribute11","Attribute12","Attribute13","Attribute14","Attribute15","Attribute16","Attribute17","Attribute18","Attribute19","Attribute20","Attribute21","Attribute22","Attribute23","Attribute24","Attribute25","Attribute26","Attribute27","Attribute28","Amount"],

"data":[[12.8,0.029,0.48,0.98,6.2,29.1,3.33,1.2,0.39,75.1,0.66,1.2,1.3,44.2,12.8,0.029,0.48,0.98,6.2,29,3.33,1.2,0.39,75.3,0.66,1.2,1.3,44.2,1.1]]

}'