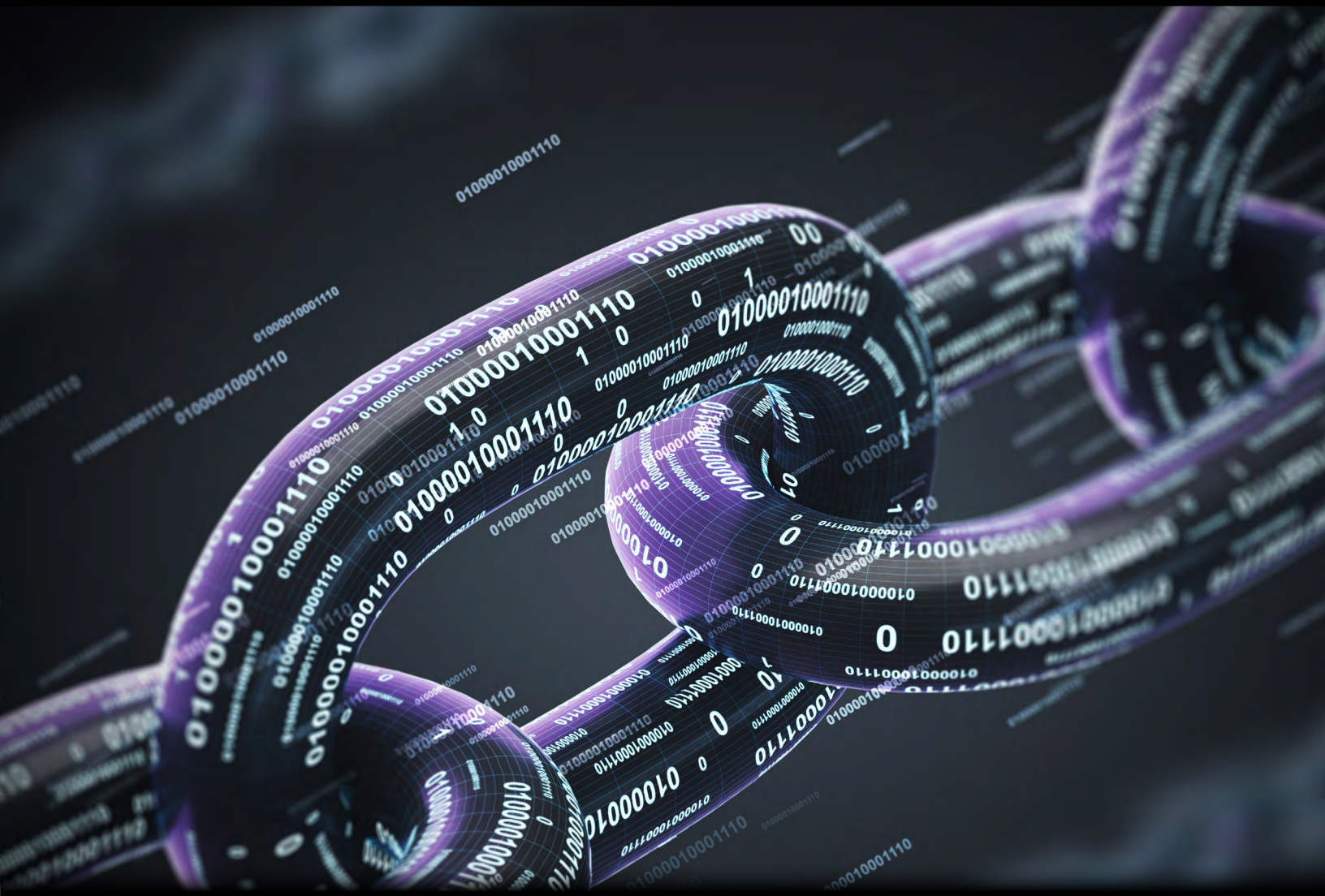
GlobalLogic®



Artificial Intelligence in FinTech: Part 2

Authored by
Dmytro Vlasenko

Contents

Analyzing Market Sentiment Using Twitter

Any kind of market, whether it is commodity, stock or cryptocurrency, is driven by the sentiment of the investors. If the majority investor sentiment about a particular cryptocurrency is positive, there is a good possibility that cryptocurrency will grow. With millions of people expressing their views about different topics every minute of the day, Twitter is a goldmine of sentiment data. As a part of our portfolio-building process, we will be doing sentiment analysis on Twitter data related to Bitcoin to judge the overall sentiment of investors.

Procuring the Twitter Data

Twitter exposes a public REST API for its users to get a Twitter feed about a particular topic. To use this method, an app registration is required at Twitter. After the app registration, Twitter provides us with four tokens: consumer key, consumer secret, access token, and access token secret.

We make use of all four keys for authorization at Twitter. We use the Python library **Tweepy** to get the Twitter feed for our sentiment analysis. First, an auth object needs to be generated and passed to the Tweepy **API** method to procure the API object. Then we can use the **search** method of the API to get the tweets for analysis. The code snippet for the same is as follows:

```
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)

auth.set_access_token(access_token, access_token_secret)

api = tweepy.API(auth)

public_tweets = api.search('bitcoin')
```

Once the data is procured, the next step is to determine the sentiment score of each tweet. The sentiment score lies between -1 and +1. -1 signifies the overall sentiment of the tweet is highly negative, and +1 signifies that the tweet is predominantly positive.

We use the **TextBlob** library provided by Python to complete our sentiment analysis. Once we clean the tweet of any unknown characters, we can perform the sentiment analysis by using the following snippet:

```
txt = clean_tweet(tweet.text)

analysis = TextBlob(txt)

sentiment = analysis.sentiment[0]
```

The complete code and notebook can be found here.

Later, we find the percentage of tweets that are positive and set up a cutoff for investment. For example, if 50% of all tweets have a positive connotation, we consider the market conducive for investment.

Completing the Portfolio

We can combine the tree analysis – namely the correlation, predictive analytics, and sentiment analysis – to form a perfect portfolio. Once we find the combination of cryptos that provide a hedge against each other, we can forecast their prices and analyze the market sentiment around them to determine the right time to enter and exit the cryptocurrency.

Risk Analytics for Loan Eligibility

Credit and loans have become a big part of our lifestyles. With everything available on loan now and cheap EMIs available, a myriad of loans are being applied for on a daily basis. Although banks do have risk analysis executives to scrutinize loan applications, it is possible an officer could miss something in his/her analysis. Also, each officer can judge a loan application to the best of their ability, but there is still a chance of personal biases impacting the loan decision.

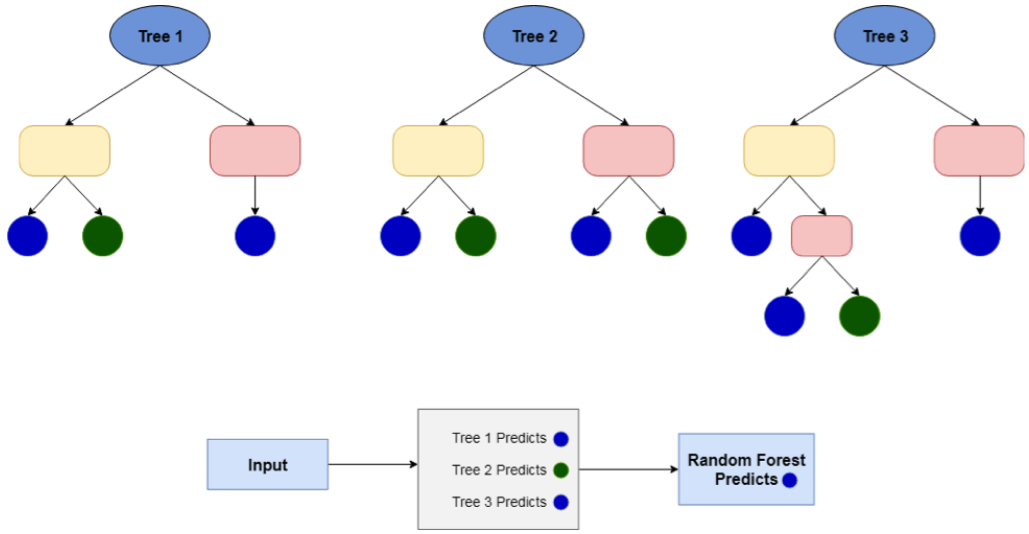
On top of that, with loans now available for smaller amounts, it is not sustainable for a bank to recruit officers to analyze all small loans. This is where the machine learning models shine because they can perform the preliminary filter to accept or reject a loan application.

In this analysis, we make use of a dataset of previous loans given by a company to study the pattern that leads to a loan default. The dataset has a column called **loan_status**, which has three values: **Fully Paid, Charged Off**, and **Current** for the ongoing loans.

We will be training a **Random Forest Classifier** model in order to predict the possibility of any future default when we get a new loan application.

About Random Forest Classifier

Random Forest Classifier is an ensemble machine learning algorithm that uses a set of decision trees to reach an outcome. In order to decide on the final output, it takes into consideration the majority of the results given by all these decision trees.



Dataset

The dataset consists of around 100 different features, such as the employment length of the applicant, annual income, installment amount, purpose, etc. We need to determine how much of an impact each feature has on our loan application determination.

Data Cleaning

The first task while analyzing any dataset is to clean the data. In our analysis, we remove the columns in which all the rows are empty, NA, or null. In addition to this, we drop the columns that have more than 60-70% missing data because it is not possible to impute such columns without introducing significant bias.

Imputing Missing Data

We impute the missing data by analyzing the type of column and the percentage of missing values. We impute the categorical columns using the mode of the data where possible and use the forward fill mechanism to impute the missing values for the date type columns.

Dimensionality Reduction Using Correlation

If two columns are highly correlated, dropping one of them is a good idea because it does not add any value to the model. We determine such features and drop those columns from the dataset.

Convert Categorical Features to Numerical

Since Random Forest Classifier works only on the numerical data, we need to convert the categorical columns into numerical. In order to preserve the data characteristics, we use **One Hot Encoding** for this. Once we have the encoded values, we drop the original column and keep these values. The code for the snippet is as follows:

```
for variable in cat_variables:

dummies = pd.get_dummies(cdf_final[variable], prefix=variable)

cdf_final = pd.concat([cdf_final, dummies], axis=1)

cdf_final.drop([variable], axis=1, inplace=True)
```

Split Dataset into Training and Test Dataset

Now we proceed with splitting the dataset into training and test datasets. We make use of the train_test_split method of model_selection in the scikit-learn package to split the dataset, such that the training dataset contains 70% of the observations and the test dataset contains 30% of the observations.

```
train_x, test_x, train_y, test_y = train_test_split(cdf_final[ind_headers], cdf_final['loan_status'],

train_size=0.7)
```

When we inspect the dimensions of the training and test dataset, we get the following:

```
Train_x Shape :: (26920, 48944)

Train_y Shape :: (26920,)

Test_x Shape :: (11538, 48944)

Test_y Shape :: (11538,)
```

We see that there are about 26,920 records in the training dataset and 11,538 records in the test dataset.

Training Random Forest Classifier

Once we have all our features encoded and the training dataset ready, our next step is to train a Random Forest Classifier model. We use the **RandomForestClassifier** method of **ensemble** package in scikit-learn to train our Random Forest Classifier.

```
clf_A = RandomForestClassifier(n_estimators=20, random_state=100, n_jobs=4)

predictions = train_model(clf_A, train_x,train_y,test_x,test_y)

measure_accuracy(test_y, predictions)
```

Here **train_model** and **measure_accuracy** are two custom functions that train the model on the training dataset and determine various accuracy metrics for the model. Our trained model has the following hyperparameters:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',

max_depth=None, max_features='auto', max_leaf_nodes=None,

min_impurity_decrease=0.0, min_impurity_split=None,

min_samples_leaf=1, min_samples_split=2,

min_weight_fraction_leaf=0.0, n_estimators=20, n_jobs=4,

oob_score=False, random_state=100, verbose=0, warm_start=False)
```

We use the Gini index in each decision tree to decipher the direction at each node. For the minimum number of samples required to split the node, we use the default value of two. Similarly, for the minimum number of samples required to distinguish a node as leaf node, we use the default value of 1.

Checking The Accuracy Of The Model

Checking the accuracy of the model is important because it helps to see if the model is predicting the true positives and false negatives accurately. We use the accuracy score and confidence matrix provided by the scikit-learn library to determine the accuracy of our classifier.

In addition to this, we check Sensitivity, Specificity, F1 score, and Precision to verify that our model has good predictive power. The following are the results of the same for the **Random Forest Model**:

Model Evaluation Metric	Metric Value
Accuracy	0.92
Sensitivity	0.49
Specificity	0.99
Precision Score	0.99
Recall Score	0.49
F1 Score	0.66

It is clear from the above table that, although the accuracy is very high, it cannot be considered as the gospel truth in terms of model performance because sensitivity for the model is low compared to the specificity. Since specificity is very high and most of the data points are of non-default cases, the accuracy is biased towards the specificity and thus has a high value.

Low Sensitivity

Low sensitivity is an indicator that the model is not able to predict the positive class well. This means its tendency to predict true positives is low and false negatives in the model are on the higher side. In our case, positive classes are the default or charged-off cases, so we cannot afford to have a low sensitivity. It would result in more than 50% of default cases going undetected. As such, we need to consider other models for improving our prediction accuracy.

Considering Alternative Models for Classification

Since there is a need for high sensitivity in the model, we need to consider alternative classification models in order to achieve this. The following are models were considered, along with their respective metrics:

Model	Accuracy	Sensitivity	Specificity	Precision	Recall	F1-Score
Logistic Regression	0.96	0.74	0.99	0.98	0.74	0.85
Linear SVM Classifier	0.97	0.85	0.99	0.98	0.85	0.91
XGBoost	0.98	0.90	0.99	0.99	0.90	0.95
Naive Bayes	0.30	0.85	0.20	0.15	0.85	0.26

Considering the model evaluation metrics, we can clearly see that the XGBoost model outperforms all the other models in the problem because it predicts both the default and non-default cases with a high amount of accuracy. We can consider this model for predicting the default cases.

Side Note About XGBoost

XGBoost, like Random Forest, is an ensemble machine learning algorithm based on an ensemble of decision trees. The acronym expands to Extreme Gradient Boosting. The difference between XGBoost and Random Forest is that XGBoost uses a technique called gradient boosting in which new models are created such that they predict the residuals or errors of prior models, and then they are added together to make the final prediction. It is called gradient boosting because it uses a gradient of the loss function to minimize the loss when adding new models.

In boosting, there is a stagewise fitting model where at each stage we have a loss function, the response, and the current model. We are trying to figure out the parameters/weights of a new function of predictors that, when added to our existing model, would improve its accuracy. In gradient boosting, this new function is represented by a decision tree.

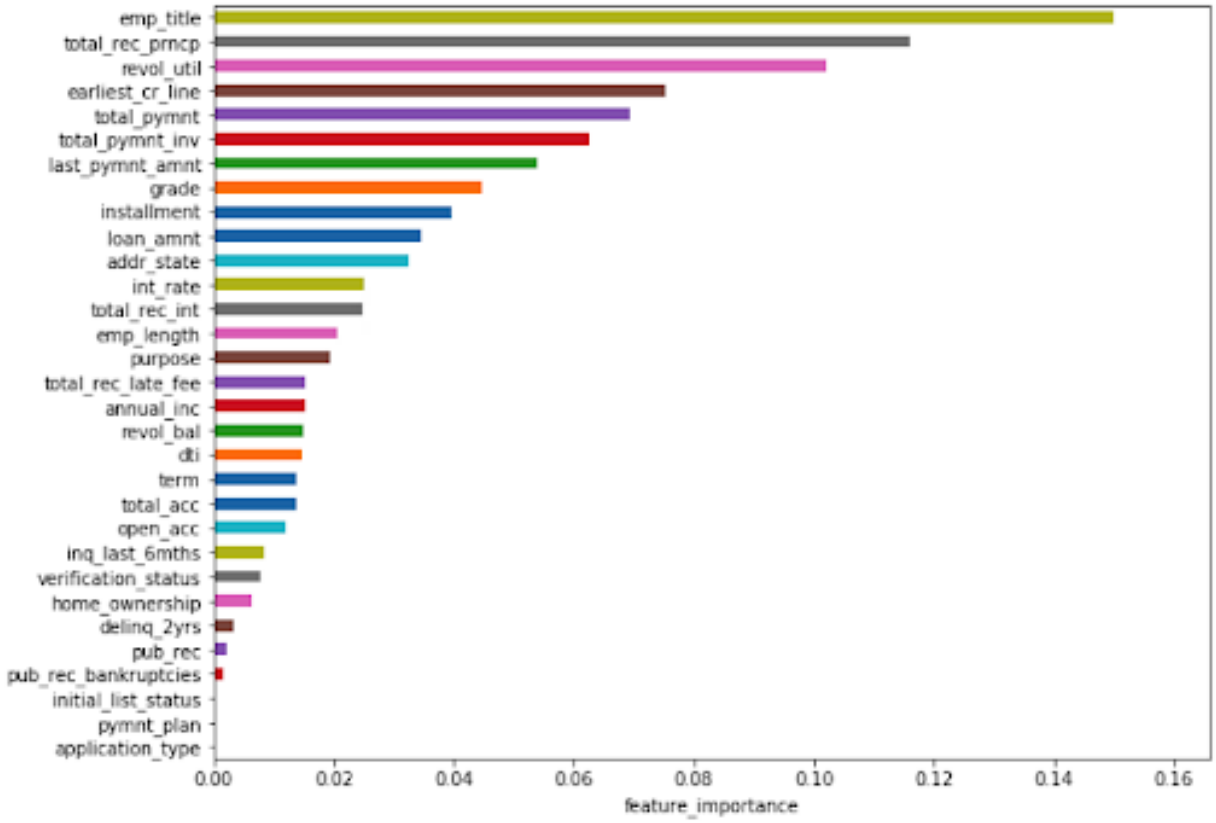
In a nutshell, XGBoost looks at the loss function at each stage of model building, evaluates the gradient of the loss function at the observations, and approximates that gradient by a tree. The determined parameters are used to formulate a function of predictors, which will be added to the existing model in order to decrease the error rate.

The ensemble is grown in an adaptive fashion, then simply averaged at the end to predict the class. If there are a lot of small trees, lasso regularization is normally applied during post-processing on these trees to select a smaller subset before averaging.

Feature Selection Using Importances of the Model

We can improve the feature selection of the model by graphing the feature importances for the model. After plotting the feature importance for our XGB classifier, we get the following graph:

graph_feature_importances(trained_model, cdf_no_ls.columns, summarized_columns=cat_variables)



In the above bar plot, we can clearly see each of the features in the order they impact the loan default possibility. We can then drop the features that have very low importance value, as this can help train the model faster without impacting the bias-variance trade-off.

Forecasting Default

Now that we have the trained model, we can use its predict method to predict default, given all the other features. As a result, we don't need manual intervention to prevent loan defaulters.

Putting the Model into Production

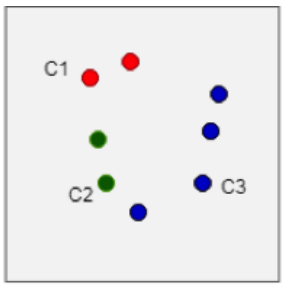
In order to put the model into production, we will need to serialize the model into a pickle file from which it can be loaded back into the memory and used to predict incoming data points. The full notebook and codebase for this use case can be found here.

Detecting Credit Card Fraud with Unsupervised Learning

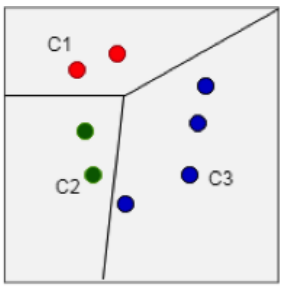
Credit card and debit card fraud swindles banks and customers out of millions of dollars every week. The transactions occur at such a large scale that it is impossible for a manual authority to check each and every transaction for its sanctity. In many cases, banks cannot find a labeled dataset with transactions identified as fraudulent or genuine. In such cases, unsupervised machine learning techniques need to be used for classification. Here we look at one such technique called K-Means Clustering.

K-Means Clustering

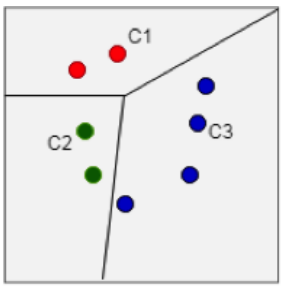
K-Means Clustering is a clustering technique that works by identifying K centroids and assigning each data vector to one of the centroids, with an aim to minimize the cosine distance between that vector and the centroid. The procedure is seen in the figures below:



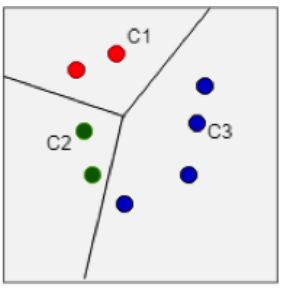
1. K Centroids are randomly generated within the data domain.



2. Each of the data point is assigned to a centroid that is nearest to that point such that K clusters are formed.



3. Mean of each cluster becomes the new cntroid for each cluster.



4. Steps 2 and 3 are repeated until convergence is achieved.

Fitting the Model

We use the K-Means method provided by the cluster package in scikit-learn. The following snippet demonstrates the procedure:

```
clf = KMeans(n_clusters=2)

clf.fit(X)
```

We get the trained model as follows:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,

n_clusters=2, n_init=10, n_jobs=1, precompute_distances='auto',

random_state=None, tol=0.0001, verbose=0)
```

Predicting Fraud Transactions in the Future

Once the model is trained, we can predict any future transactions, given the feature vector as follows:

```
predict_me = np.array(X)

predict_me = predict_me.reshape(-1, len(predict_me))

prediction = clf.predict(predict_me)
```

The output will be 1 or 0 because we have two clusters, with 1 being the fraudulent cluster. The full notebook can be viewed here.

Exposing Fintech AI as Chatbots

Once we have modeled a given problem and have achieved the required confidence to launch it to the world, the algorithm can be wrapped in the form of a chatbot. Chatbots reduce the workforce needed for a project and provide a new form of customer engagement.

For instance, in the case of our portfolio management algorithm, once we have concluded that the market is conducive for investment, our closed domain chatbot can provide insights to customers who hold cryptocurrency wallets with us.

Similarly, chatbots can be built to help people improve their CIBIL score by identifying the major factors that are negatively affecting their credit score.

The opportunities in the area of chatbots are tremendous, and the area is still relatively unexplored. With the advent of API.ai, Rasa, and similar chatbot services, it is very easy to train and set up a closed domain chatbot that can go a long way to improving customer-centric strategies.

Conclusion

The continuous evolution in the finance industry – with the plethora of digital transactions and new technologies like Blockchain – has been mind-boggling. Scaling such a huge industry with a rapid metamorphosis rate is not possible without bringing in machine intelligence. In the above whitepaper, we saw glimpses of the power of machine learning (ML) in Fintech. With the advances in deep learning algorithms and ever more powerful GPUs, the future of ML looks bright. As ML models become more accurate, financial organizations will begin to rely on them more frequently. This, in turn, will inspire ML engineers and data scientists to come up with new algorithms and optimize the existing ones.

Appendix

Source Code and Datasets

References

Fraud Detection in Credit Card by Clustering Approach - Vaishali

ARIMA for Time Series Forecasting - Jason Brownlee

7 Methods to Perform Time Series Forecasting - Gurchetan Singh

K-Means with Titanic Dataset - Practical Machine Learning Tutorial with Python - Sentdex

Twitter Sentiment Analysis - Siraj Raval

Forecasting Stock Index Movement: A Comparison of Support Vector Machines and Random Forest by Manish Kumar, M. Thenmozhi :: SSRN

scikit-learn: Machine Learning in Python — scikit-learn 0.24.2 documentation
(-forest-classifier-python-scikit-learn/)

GlobalLogic®

GlobalLogic is a leader in digital product engineering. We help our clients design and build innovative products, platforms, and digital experiences for the modern world. By integrating strategic design, complex engineering, and vertical industry expertise, we help our clients imagine what's possible and accelerate their transition into tomorrow's digital businesses. Headquartered in Silicon Valley, GlobalLogic operates design studios and engineering centers around the world, extending our deep expertise to customers in the communications, automotive, healthcare, technology, media and entertainment, manufacturing, and semiconductor industries.



www.globallogic.com