AI IN EQUIPMENT
AND AUTO FINANCE
PART 2: USING MACHINE LEARNING IN THE WILD
ABOUT THE AUTHORS

Martyn Tamerlane is a Solution Architect at Alfa, with a focus on project delivery. During his Master’s degree in Mathematics and Computer Science, Martyn developed a lasting passion for latent technologies and how they can be integrated with legacy systems.

Maurice Buckberry has been both Senior Software Engineer and Senior Solution Engineer at Alfa. He has a Bachelor’s degree in Computer Science, with experience in all stages of the software development lifecycle and a particular interest in performance-tuning large-scale applications.

Contributors

Andrew Flegg, Chief Technology Officer
Andrew Ingram, Global Head of Product Strategy

FOREWORD

BLAISE THOMSON

Cofounder of Vocaliq and Alfa IQ, and Founder of Bitfound

Machine learning (ML) is a technology that thrives in situations with large datasets and clear objectives. It was only when I met the Alfa team earlier this year that I realised how many exciting AI problems there are in the asset finance industry. There are obvious tasks like credit scoring, but many areas that have been neglected, including delinquency prediction, optimisation of forbearance strategies, and portfolio analysis.

This technical paper describes two ML-related projects that Alfa engineers have tackled out of their own initiative. It has interesting insights into the trade-offs between adopting off-the-shelf approaches and building up models internally, as well as showing how much can be done with a relatively small investment.

As Alfa’s investments in ML evolve to be more systematic, the opportunities that are unlocked will be significant. I am excited to see what Alfa IQ will soon bring to the asset finance market.

SEE BACK COVER FOR INFORMATION ON ALFA IQ
In our recent position paper on AI, Part 1: Balancing Risk and Reward, we outlined the high-risk, high-reward nature of using AI in our industry, and machine learning (ML) in particular. We discussed why ML can be an excellent choice, but also highlighted the risks of misusing data, and spelled out the challenges inherent in developing a high-quality solution.

In this follow-up technical paper, we explore in detail two specific use cases which take very different approaches to ML implementation. The first, which discusses licence plate detection, showcases the increased accessibility of ML solution development through the use of AI-as-a-Service (AIaaS) products now offered by many cloud providers. The second use case concerns the automated analysis of testing output from our own codebase. An in-house solution based on internal data, it forms a foundation for future ML solution development.

We hope these examples convey genuinely useful and practically applicable advice. Throughout the piece, we'll compare the methodology, data and cost characteristics of each solution. Furthermore, we'll provide a guide to deciding which approach you should use for your solution, based on your requirements.
TERMINOLOGY

In Part 1: Balancing Risk and Reward, we explained machine learning as the scientific study of algorithms and statistical models that a computer can use to perform a specific task effectively, without being given explicit instructions, instead relying on patterns and inference. We also discussed the two approaches of supervised and unsupervised learning, and set out the process for training a neural network.

Deep learning is a further subcategory of ML, in which there are multiple layers of neurons, as opposed to just one. Some other useful definitions referenced in this paper:

- **Neural network:** A machine learning algorithm modelled on the way our brains’ neurons interact with each other. Neural networks commonly have layers of these neurons: an input layer, an output layer, and one or more middle layers (also known as “hidden” layers).

- **Deep neural network:** A neural network that has more than one “hidden” layer.

- **Convolutional neural network:** A neural network that uses a convolutional operation to determine the activation of neurons in the next layer. Rather than connecting all neurons within a layer to all neurons in the adjacent layer, neurons are connected in a grid-like structure (a convolution) in which the weights for each convolution are shared.

- **Residual neural network:** A deep neural network where layers are connected not only to the adjacent layers, but also include skip connections to other layers. This allows the network to retain accuracy when trained over very large datasets and with a high number of hidden layers.

Our first use case is based on the use of automated object recognition for vehicle licence plates. Over the last decade, ML techniques for the field of object recognition have progressed at an impressive rate. This has helped make self-driving cars not just possible but achievable, which in turn has further fuelled research into the optimisation of ML techniques in this area.

Alongside advancements in the way machine learning models are trained, cloud providers have taken the opportunity to automate the surrounding infrastructure. They offer AIaaS products which reduce the time needed to set up the various hardware and software required for an experiment or proof-of-concept exercise. Most providers also offer on-demand computing power, eliminating the need for expensive, high-powered machines to train the neural network.

This use case draws on Alfa’s AIaaS products to integrate licence plate recognition with business logic in Alfa’s own asset finance software platform, Alfa Systems. It demonstrates how these services enable experimentation with ML techniques, at little cost and without requiring great expertise.

**USE CASE 1. LICENCE PLATE DETECTION**

Our first use case demonstrates how AI-as-a-Service (AIaaS) products enable experimentation with ML techniques, at little cost and without requiring great expertise.
This approach seeks to streamline two areas that require the most expertise in developing ML models:

- Acquiring and preparing a large volume of high-quality data
- Determining the most suitable ML algorithm for the problem, and tuning it

By leveraging existing cloud services and publicly available data, we can cut the costs of setting up the experiment in terms of both time and money. The solution is built around AIaaS products offered by Amazon Web Services (AWS) along with a camera called AWS DeepLens, also produced by AWS and designed specifically to be programmable to recognise objects in images.

AWS DeepLens will be programmed to take a snapshot of its view every few seconds. It will then analyse that snapshot by inputting it into an ML algorithm which is pre-trained to detect whether a license plate is present. If detected with sufficient accuracy, AWS DeepLens will then send the section of the snapshot that contains the license plate to a text recognition service provided by AWS called Rekognition. This managed service will then use its own ML model optimised for text recognition to extract the text in the image, outputting the license plate in text format. The license plate text can then be used in web service calls to Alfa Systems to trigger business logic, such as:

- Contract termination
- An update to the current location of the vehicle
- A workflow case creation or progression that prompts manual action

The experiment uses Amazon Web Services’ AIaaS and computing resource, its AWS DeepLens camera, and AWS Rekognition text recognition service.

WHY AI?

Two parts of the proposed solution use ML: the recognition of the license plate in any given image, and the recognition of text within an image. Both use a subcategory of ML called deep learning, which trains a neural network using vast amounts of data alongside correct outputs, in order to classify new data.

Recognising a license plate or text in an image are both examples of something which has traditionally posed problems for ML practitioners: object recognition. Over the last few decades, many different methods of object recognition have been attempted. Deep learning techniques are a particularly apt tool for this, since they typically exhibit a much higher predictive performance than standard techniques, and can be applied to a much wider variety of input data than other methods – which are usually bespoke to narrow varieties of input. By taking advantage of vast amounts of data (images that have objects correctly labelled in them) and processing power to train a neural network to learn the patterns between the labels and the objects, deep learning has become the most commonly used solution for recognising a wide range of objects under different conditions.

Recognising text is a smaller, simpler subset of object recognition and there are many publicly available deep neural networks, pre-trained on high-quality data, which exhibit very high accuracy. In our case, we are assuming that text on a license plate will be clear and is designed to be easily readable. There is little value in training our own neural network because many publicly available ML models, such as that used by AWS Rekognition, are sufficiently accurate.
For slightly more complex objects, such as recognising a vehicle that has a licence plate showing in the image, we can train our own deep neural network since it may require further tuning to increase its accuracy for the specific image we want to recognise. Since we are already using an AWS service, it makes sense for us to use another of its services designed for automating the ML training process, SageMaker. Using a number of preset machine learning algorithms, as well as our own data, AWS SageMaker allows us to train our own ML model.

**BUSINESS VALUE**

The ability to trigger an event in Alfa Systems (Alfa’s asset finance software platform), based on detection of licence plates or vehicles, allows us to further increase automation. Ultimately, this ability should make the business process more efficient, leading to lower costs and/or better customer service.

For example, a licence plate on a dealer’s forecourt could be recognised, triggering a web service request to Alfa Systems to determine whether a vehicle damage claim has been made. If the response from Alfa Systems is positive, a screenshot of the vehicle can be sent to a vehicle damage detection company for post-processing. Alternatively, the screenshot could be sent preemptively to a vehicle damage detection company to provide a likelihood of damage, which uses Alfa Systems’ API to trigger a follow-up process in Alfa Systems, prompting investigation. In this situation, the automated request to the vehicle damage detection company will save the dealer the cost of calling out damage inspectors when no damage is detected. The lessee of the vehicle receives better customer service through a quicker response to their damage claim.

**TECHNICAL OVERVIEW**

The stages of this process are spelled out below:

1. Train a deep neural network in AWS SageMaker to recognise licence plates.
2. Deploy the trained deep neural network to AWS DeepLens. Program AWS DeepLens to take a snapshot of what it can see every few seconds, and feed the input to the deep neural network. The output will indicate whether there is a licence plate in the image with a given accuracy percentage.
3. If the accuracy is higher than 90%, AWS DeepLens is programmed to send the image to a storage location in AWS called an S3 bucket.
4. Code is stored in AWS in a serverless function called a Lambda, which executes when any new image is saved to the S3 bucket.
5. When a new image is saved, the Lambda passes it on to AWS Rekognition in order to extract the licence plate text.
6. Once the text is returned to the Lambda, it can then trigger any type of integration with Alfa Systems, such as a web service call to terminate the vehicle’s agreement.
A simple example could be to update an asset event related to the vehicle:

```
curl --header "Content-Type: application/json"
  --data '{"query": "REG=LPL4TE", "maxPageSize": "2", "pageNumber": "1", "segments": []}'
  https://alfahostedclient.alfasystems.io/json/asset/v2/asset/searchByIdentifier
```

The response, as shown right, provides us with the identifier used internally by Alfa Systems, which can then be used to drive business logic relating to the vehicle by calling a rich variety of web services. As an example, we can use the identifier to call the asset event web service to create a particular event in Alfa Systems.

There are many more examples that could be extended from this, such as the one described in the Business Value section above.

---

**POINTS OF COMPARISON**

**DATA PROCUREMENT**

A high-quality set of training data for object recognition was used. Its small size meant that the training time was short, and less computing resource was required.

In order to train a neural network to recognise licence plates, we needed a large quantity of high-quality data, in the form of images with objects labelled in them. Fortunately, object recognition is a very common ML problem and there are many publicly available, high-quality datasets that can be used. Of those publicly available, CALTECH-256 is a small, hand-labelled set of images that, most importantly, includes the labelling of licence plates. The images are obtained from public search sites such as Google Images and are carefully selected, manually, to cover an extensive taxonomy of objects that are clearly recognisable in the image. Its small size of 30,607 images means that, although the neural network won’t be as accurate as one trained on a larger set, the training time required is short and requires less computing resource.

While a small dataset is ideal for a quick end-to-end setup, we may want a larger dataset to increase the accuracy, either by finding a larger publicly available dataset or by creating our own. Creating or extracting this data ourselves would be very time-consuming but might be necessary if we want direct control over the quality of data. For example, we might want to train it specifically to recognise licence plates in particular weather conditions, in which case we would need to ensure the training data contains enough examples in those conditions.

**DATA PRIVACY**

Even though we can enforce appropriate levels of access security and encrypt all data used within the cloud provider’s infrastructure, owners of the data may not allow their data to be shared with third parties like the cloud provider. Additionally, some may not allow custom ML training algorithms to be imported to the cloud, since it would be considered intellectual property, despite the tight security that can be configured to protect it.
For the purposes of this experiment, we used a publicly available dataset and an ML algorithm that is open source, avoiding this issue. If either of these is identified as an issue, we would need to work with the appropriate data controllers and processors, within the relevant regulatory framework, to mitigate the concerns. This may require using other ML approaches and tools, rather than AlaaS.

**ML TRAINING ALGORITHM**

A popular type of neural network used for object recognition, called a residual neural network, was chosen to process the images, and trained to classify the labelled objects. 80% of the images were used for training and 20% reserved for verifying the accuracy of the network once it was trained, which is a common split between training and verification data.

AWS SageMaker was used to coordinate and manage the creation of the ML model. SageMaker automates the provisioning of hardware required to run the training algorithm, the environment configuration of software installed on the hardware, the selection of the training algorithm itself, and the execution of code that we wrote in order to fit it all together. It is configurable enough for us to import or create our own training algorithm and import our own datasets to use for the training process.

The key benefit of using AlaaS in this situation was the automation and abstraction of infrastructure provisioning and configuration. Even though everything that the service was providing could be achieved on Alfa’s own infrastructure, it would have required extra effort to automate and to measure, compared to the very low pay-as-you-go fees that the service was providing.

**DEVELOPMENT PROCESS**

Development time (~7.5 days, 1 person)

- High-level design, research into AWS services (3 days): Most of this time was spent learning about how the different AWS services interact with each other, what was possible and how much it would cost.
- Finding training data, creating the training script in Python (2 days): Some time was spent researching existing datasets and the possibility of creating our own training data.
- AWS DeepLens (2 days): Some time was lost here learning about how AWS GreenGrass works (an AWS framework for IoT devices) and some incompatibilities with certain versions of Python.
- Lambda script and remaining integration with AWS Rekognition and Alfa Systems (0.5 days): The remaining integrations are simple calls to well-documented web services.

Another benefit of using AlaaS was the short amount of time it took to get a solution working end-to-end and the opportunity for parallelising the effort. The integration between each service and device in AWS, the selection and configuration of the training algorithm, along with the procurement of training data, could all have been worked on independently.

**REQUIRED LEVEL OF EXPERTISE**

Many of the managed services used provide preset infrastructural configuration, which saves time tuning hardware and software. The configuration may need to be tuned later on but provides a quick starting point.
There were many different options available for integrating services in AWS. Python seemed like the natural choice since most of the example projects were written in it, which was especially useful for AWS GreenGrass, which is a very bespoke framework for IoT devices that we had not used before. While a proficient level of coding is sometimes required, not much code is written for this initial experiment, so would still be achievable with basic coding experience.

The selection and tuning of the training algorithm followed the trend of many other object recognition ML models. Using residual neural networks for object recognition is hugely popular and so is the training dataset we used. However, it’s clear that if these were not available and widely tested, this experiment would have taken significantly longer.

Although some coding experience was required, AIaaS products saved us considerable time in software and hardware configuration, as well as data selection and training.
SUMMARY

At Alfa, within our application development infrastructure, we maintain a comprehensive suite of over 2,000 automated, “end-to-end” tests. These tests run over the source code of our software product, Alfa Systems, on a daily basis as part of the continuous integration process. They ensure that, every time we make improvements to the application code, the product remains fully functional - from both business process and technical perspectives.

When an enhancement or code change is made which results in a test failure, identifying its cause can be a time-consuming process. There are sometimes complex related patterns which are not immediately obvious. As part of this experiment, we wanted to see whether these patterns could be detected using ML techniques - namely neural networks.

In contrast to the licence plates use case detailed above, this time we built our own framework for training and deploying ML models. This could then be used for any use case in the future where data is stored within Alfa’s internal network. The application would consume data from an internal Alfa application (in our initial use case, an internal tool that manages test failures), transform the data into a format that can be consumed by a neural network, and run a training algorithm to produce an ML model to be used for outputting predictions.

WHY AI?

The usual way to identify the cause of a failure in end-to-end testing is for an individual to examine all the code changes made since the last successful test run. That person investigates, using their technical experience and familiarity with the code. At Alfa, we use an internal tool that lists all the Java classes that have been changed, which narrows down the potential cause, but nevertheless can be large and impractical for that person to comb through - especially when considering Alfa Systems’ high rate of improvement.

Instead, a neural network trained on historic failures should be able to learn patterns between the areas of code that are causing failures and the failures themselves.

Alongside the exploration of ML techniques, we considered a number of alternatives that didn’t use ML. One alternative could be to program a set of heuristics that describe the relationships between the test failure and the location of the code that caused it, in a way similar to rules-based
credit decisioning applications. The main reasons against using this, or any of the other alternatives, were based on the following assumptions:

- The relationship between test failures and the location of code that caused them are complex and would require more effort to write than training a neural network.
- The relationship changes over time, sufficiently that maintaining a set of heuristics would require a significant ongoing cost.

If the resulting ML model is inadequate, we can revisit some of these alternatives and reassess our assumptions.

BUSINESS VALUE

With the proposed solution in place, we should be able to eradicate much of the time we spend investigating test failures. The accuracy of identifying the root cause should also improve over time, as it is trained on more test failures in the future.

Aside from our specific use case, the effort spent developing the framework can be reused for future use cases with which we want to experiment with ML. This removes much of the up-front cost involved in setting up the infrastructure and ML training pipeline.

TECHNICAL OVERVIEW

Instead of building an application solely for our use case, we decided to build a framework for training and deploying ML models that could be used for any future use cases involving ML models.

The application would consume data from an internal Alfa application (in our case, an internal tool that manages test failures), transform the data into a format that can be consumed by a neural network, and run a training algorithm to produce an ML model.

We're looking not only to save significant time spent on investigating the cause of test failures, but also to reuse the time spent developing the framework for future ML requirements.

Our reusable application is trained on past test failures to return the Java classes most likely responsible for new ones.

Our model is trained on past test failures, for the purposes of returning the Java classes that are most likely responsible for those failures. An internal application that
manages all test failures then sends our ML application any new failures. This then uses our trained ML model to provide a list of Java classes with a high probability of being the cause of the failure. The model also accommodates instances where multiple files have received code changes to fix the test failure.

After establishing the feature set we wanted to use, we detailed the limitations and edge cases that might challenge the model:

- When a test fails for the first time, there will be no historical data. However, the model can still be useful and find patterns in other inputs. If a specific server-side error occurred on the new test that has also been seen on other tests, and been correlated to an existing area of Alfa’s code, a trend could still be identified and an accurate suggestion made for a test not seen before.

- If a portion of code in the dataset undergoes a significant refactor, it will be impossible for the model to correlate any new test failures with historical failures that also occurred in the refactored class.

**POINTS OF COMPARISON**

**DATA PROCUREMENT**

At Alfa, we automate the recording of end-to-end test failures along with its root cause and resulting fix. This data is stored in a highly structured manner and captures many years of analysis. Having this data accurately labelled, stored in one place and organised logically in a database reduces the amount of work needed to label the data and transform it into a format ready for our neural network to process.

At the start of the experiment, we discussed the exact properties, or features, within the data that we would need to extract. These features constitute one row of training data that we feed into our neural network model, of which we managed to extract 10,000.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Example Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java class name for failing test</td>
<td>TestObject</td>
</tr>
<tr>
<td>Java class method name for failing test</td>
<td>testUpdateObject, testDeleteObject</td>
</tr>
<tr>
<td>Client Java Exception type</td>
<td>AssertionError, NullPointerException</td>
</tr>
<tr>
<td>Server Java Exception type</td>
<td>NullPointerException, HibernateException</td>
</tr>
<tr>
<td>List of Java class names changed during code fix</td>
<td>TestObject, ObjectService, ObjectEntity</td>
</tr>
</tbody>
</table>

Since we have control over access to the data, we have the ability to change which features we want to extract in the future in order to tune the accuracy of our ML model. The source data is also in one place, accessible by the ML application in order to extract the features it needs directly. Comparatively, there would need to be duplication of this training data if we were to use AIaaS, since we would first need to extract the features within Alfa’s internal network, then transfer the data to the cloud provider’s infrastructure ready to be used by its AI services.

**DATA PRIVACY**

All of the data used for this experiment is stored within Alfa’s internal network, as are the ML training application and any of the resulting ML models. The data describes Alfa’s software products, and contains no personal information. This avoids putting data on the third party infrastructure of a cloud provider, which could prompt concerns from data owners.
ML TRAINING ALGORITHM

For the implementation of ML, we decided to use an open-source Java library called Deeplearning4j. Using the data described above, we trained our ML model to identify trends between the supplied data and link this data to the Java class names where the fix was identified and made. As with the first use case, the neural network was trained on the accepted standard of only 80% of the available data, with the remaining 20% used to test the accuracy of the neural network - given we know the cause of each test failure in our training set.

One benefit of neural networks that we can leverage is its ability to be retrained in the future on a regular basis, to learn new patterns of why tests fail. For this use case in particular, the model will become less accurate over time since we expect the nature of test failures to change. This is otherwise known as model drift, which we can mitigate by periodically retraining the model on recent data.

For the first iteration of this solution, the retraining of the ML model took only a couple of hours on a personal computer before we began to reach diminishing returns. This means we can retrain our ML model on a daily basis to pick up and report new test failures. We must regularly retrain the ML model; otherwise, there is a risk it will not learn about new code changes that have been made to our software that can have caused failures. This would make our predictions less accurate over time. We are accustomed to regression-testing existing functionality in our applications, to ensure that new features do not break existing ones.

DEVELOPMENT PROCESS

Development time (~25 days, 2 people)

• **High-level design (1 day):** Since each part of the architecture is within Alfa’s network and we are familiar with how the internal services work, design time was comparatively short.

• **Infrastructure and ML training pipeline setup (4 days):**

• **Code refactoring to make the pipeline sufficiently generic for other use cases to use it in future (10 days):**

• **Iterative tuning of the training algorithm for the test failure use case (10 days):** When testing our trained model, it became apparent that we needed to tweak various parameters on our neural network to maximise the performance of our model. We used the same baseline set of tests to extract data, train, and test the accuracy of our model in order to make it easier to compare the performance of various configurations of our model.

Working in a small team on this project allowed us to be agile in our approach and iteratively add value to the solution. By evaluating the accuracy of our model against our training data, we made sure we tested the value we were adding.

After determining a set of parameters that seemed to be giving us the best results, we tried training our model on our laptops as the amount of data was manageable for such computing power. After an hour or so of training, we began to see diminishing returns from our model. Given the relatively slow rate at which the test dataset is increasing in size, and the fact that the time to train the model is correlated with the size of the dataset, it would take some time before we outgrew the computing power that we have available internally at Alfa.

REQUIRED LEVEL OF EXPERTISE

In general, the level of expertise required for this use case was much higher and broader than that demanded by AIaaS. We needed to be comfortable with setting up the infrastructure, including streaming data from internal applications to our ML application. Comparatively, with services managed by AWS, infrastructure is provisioned and software is installed automatically.
Learning how to use Deeplearning4j required some software development experience, as well as experience with amending a machine learning algorithm.

A greater understanding of the data was also required, since we needed to identify the features we wanted to extract from the dataset. Fortunately, everyone at Alfa is familiar with how our internal data is structured, so this wasn’t as much of an issue compared to analysing an unfamiliar dataset.

The two use cases outlined in this paper demonstrate two very different approaches to using ML to solve a problem. One creates an in-house framework for developing ML models, while the other relies on AIaaS. The following guides summarise the points of comparison between the two approaches.

<table>
<thead>
<tr>
<th>Data Procurement</th>
<th>Using AI-as-a-Service</th>
<th>Building your own ML pipeline</th>
</tr>
</thead>
</table>
|                   | Services that come with a pre-trained ML model remove the need to find a large quantity of high quality data. | You’ll need to find your own data but will have direct control over exactly what is included, using data you have procured will require in-depth domain knowledge of the data.
|                   | If you’re using a service that uses a pre-trained ML model, you don’t have anything to worry about. If you’re providing data to a service that facilitates the training of the model, you need to be considered when sharing their data with third-party providers. | Direct control over the management and administration of the data removes the involvement of a third party.
|                   | Using a service that provides a pre-trained ML model removes the need to understand exactly what training algorithm to employ for your particular use case. | An understanding is required of the appropriate techniques to use when tuning the training algorithm.
|                   | The set-up process of an end-to-end solution is fact for problems commonly solved by ML techniques. | An initial up-front cost is required to automate the ML pipeline, but ultimately you have full control over its architecture, which may be critical in integrating with other systems.
|                   | An end-to-end ML pipeline can be achieved with basic programming skills. | A greater understanding of setting up the infrastructure is required.

The DIY approach required more effort and expertise than the AIaaS one, but this was necessary to the nature of the task.
DIY OR AI-AS-A-SERVICE. WHICH IS RIGHT FOR ME?

The experiments detailed in this paper explore different ways to develop ML solutions, and each requires different levels of time, effort and expertise.

However, both solutions rely on domain knowledge of the data used; the first relies heavily on research in object recognition and a resulting dataset that was created by domain experts in that field, while the second is based on data with which most people at Alfa are familiar.

Because we have an intimate knowledge of the structure of Alfa Systems data, we are currently looking into the possibilities and potential of the data stored and used in our platform by our clients. It’s clear that, when it comes to high-value solutions involving client data in this area, the level of expertise and time required for handling such sensitive data is paramount.

The innovation culture at Alfa is strong, and this has enabled the contributors of this paper to spend their time building solutions using AI. We have also been able to take advantage of high-quality supporting infrastructure, AWS accounts and help from various teams across the company during the process. It is undeniable that there is currently widespread advancement and adoption of ML in asset finance and thus it is important for everyone at our company to be aware of it.

Being able to openly discuss and challenge each other while working on solutions, such as the ones detailed in this paper, have provided us with many opportunities to share our knowledge.

These two examples have provided a glimpse of what could be achieved when applying ML techniques to use cases in asset finance. With accelerated, full-time development of ML and AI solutions, paired with the high-quality data produced from Alfa Systems, it’s exciting to think of the potential value that can be gained.

We hope you’ve found this technical paper useful. Alfa’s next AI publication will further discuss the future of AI and its impact on asset finance.
Coming soon...

Alfa iQ
An Alfa and Bitfount Company

The intelligence platform for asset finance.

Bringing intelligence to asset finance. We make access to assets efficient and successful by delivering intelligence to the world’s auto and equipment finance providers. Our models improve credit decisions, delinquency support, fraud detection and more.

The Alfa iQ model

- **Industry** knowledge from the creators of the world’s leading asset finance software
- **Technical** expertise from renowned artificial intelligence experts
- **Data** that’s tailored to your needs
- **Impact** that improves your bottom line and social good

Every decision system can be improved. Learn more and stay updated at alfaiq.com