PFM Insights Project

Knowledge Transfer Report

Context

Goal of the project

Fiserv would like to understand the capabilities and constraints of machine learning on mobile devices such as an iphone and an android phone. A Personal Finance Management application is used as proof of concept for the machine learning capabilities. Only machine learning on iOS is covered in this report.

The context of the Proof of Concept (PoC) App

The app that will be used to evaluate the development and application of ML is a personal finance management app (PFM). A PFM typically involves a user manually categorising transactions and then using this to analyse the user's spending habits over time with a view to the app making suggestions on what spending and savings changes to make to meet pre-defined financial goals.

The app will handle the input of a user's transactions and they will be able to categorize their transactions as a Need, Want, or Joy. The app will also provide the user with an analysis of their spending information.

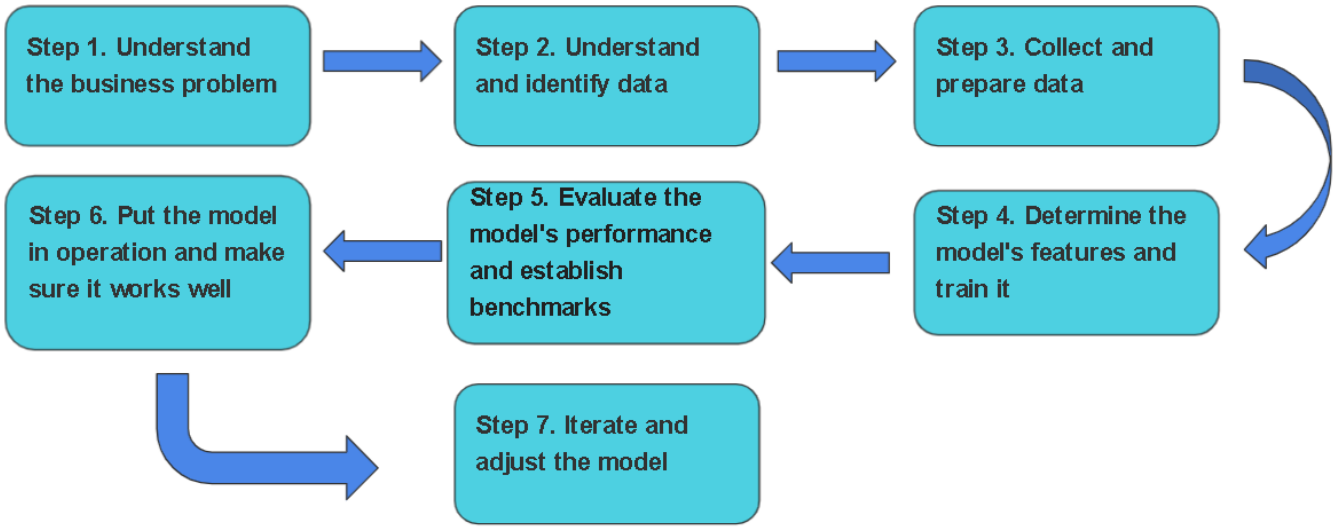
The Machine Learning capabilities will be applied to the automated categorization of transactions. The app will automatically predict and suggest categories for new transactions, which the user can review, confirm or change, reducing the effort of manual categorisation of all previous and future financial transactions..

The following questions will be addressed through research and experimentation and application of the PoC PFM application.

1. Can the ML model be trained on the mobile device? What constraints or capabilities?
2. If the ML model is trained off-device, how can it be transferred to the mobile device?
3. What libraries and ML algorithms are available for text classification? What constraints?
4. What data is available for training the model and how is it input into the ML algorithm?
5. What is a suitable input vector for training the model?
6. How is the quality of the model measured and tested?
7. How can the model be integrated with an iOS app such as the PoC PFM app?

Overview of ML model training and testing

The ML areas to investigate are summarised in the following diagram. Each of these phases needs to be considered when investigating how to train and apply a ML model



Our team spent some effort in each of these steps at different times, both researching options and upskilling in the principles and implementation details on iOS. MOst of the effort was in upskilling our knowledge and capability so we could iterate through steps 4-7. The next section summarises the main areas of effort.

Summary of Work Done

Understanding the requirements for the proof of concept application and the PFM domain (Step 1)

Upskilling of understanding of machine learning concepts

Upskilling in what iOS offers in terms of machine learning libraries, tools and capabilities.

Upskilling in specific iOS development tools eg Xcode, SwiftUI, CoreML, XCTest

Trained and created a ML Model that can categorize transactions

Looked into different methods for data labeling. (Steps 2 and 3)

Finding and cleansing suitable data sets for model training and testing (Steps 2 and 3)

Manually labelling the training and test data (Step 3)

Experimenting and evaluating the suitability of ML algorithms Step 4)

Selected a suitable algorithm and trained a model (Step 4, 7)

Developed a PFM app to test the categorization model and create a proof of concept (Step 5 and 6)

Investigated which algorithms would accommodate on-device updates to the transaction categorization model with new data points (e.g. ML labelling corrections) (Step 7)

Researched a potential work around for on-device updates with tabular data

Investigated TensorFlowLite as an alternative cross-platform ML framework to CoreML

Identified Future possibilities of emerging on-device ML technologies

We checked various sources with no findings on future updates of CoreML and TensorFlow Lite

ML Frameworks

CoreML3

https://developer.apple.com/documentation/coreml/core_ml_api

CoreML an ML framework for integrating machine learning models into iOS and MacOS apps. This is the framework we used when creating our PoC app to test our transaction classification model.

CoreML3 is optimized for on-device performance. Previous machine learning tasks performed on-device required data to be sent/received from a server. CoreML minimizes the memory footprint, uses less power, can function without network connection, and protects user data by keeping it on-device.

CoreML's new on-device MLUpdateTask API can update models on a user's device, allowing models to be personalized with an individual user's data, without having to retrain the model on another machine.

CoreML supports conversion of other models into CoreML using CoreMLTools. This allows for the creation of a model in another framework like Keras or TensorFlow and converts it into a coreml model to make use of the on-device capabilities of CoreML3.

This conversion is also required for the creation of an updatable model because CreateML does not support the creation of models with the updatable algorithms. This means to create a model that is capable of being personalized on-device based on an individual user's new data, the model must be created in a third-party tool such as Keras, and converted into an updatable CoreML model using CoreMLTools.

Constraints

The new MLUpdateTask feature currently only supports two updatable algorithms, K-Nearest Neighbor, and Neural Networks. The CoreML API restricts these algorithms to be used as image classifiers, to work with the on-device updates.

A tabular classification model like the transaction categorization model we used for our PoC app, cannot make use of the newer on-device update capabilities. This restricts the usage of CoreML in these cases to getting predictions on-device from a pre-trained model.

Iphone/IOS compatibility

CoreML apps require iOS 11.0+ or macOS 10.13+

CoreML on-device updates requires iOS 13.0+

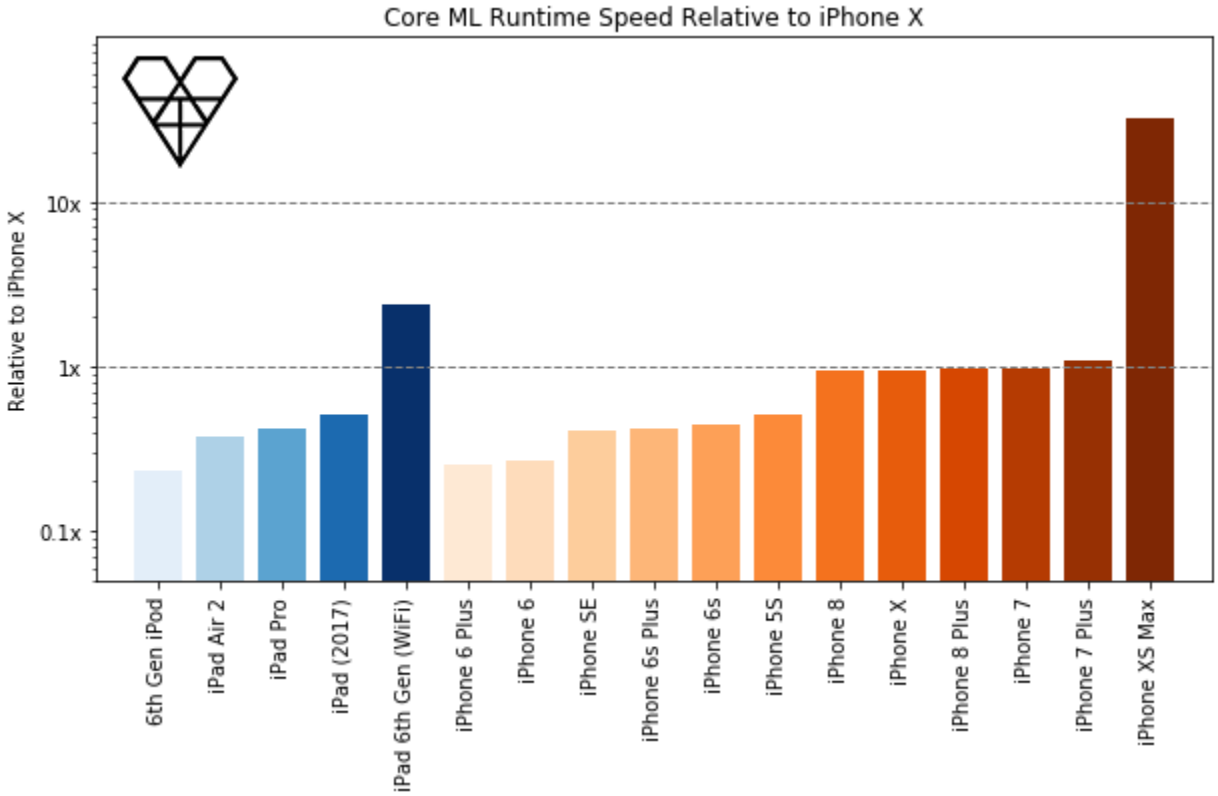
Devices that can run iOS 11.0:

- iPhone 5S
- iPhone 6
- Includes all later models

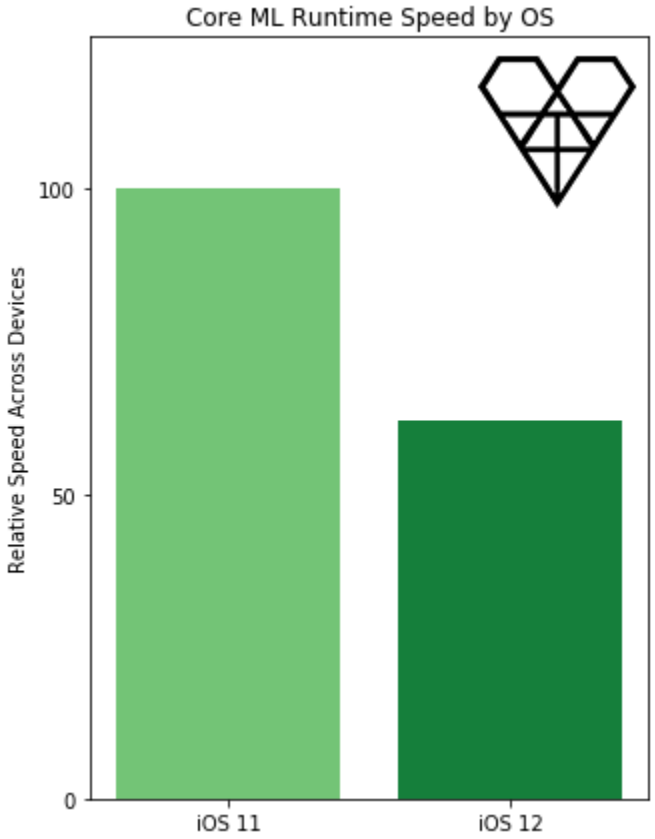
Devices that can run iOS 13.0:

- iPhone 6s & iPhone 6s Plus.
- iPhone SE & iPhone 7 & iPhone 7 Plus.
- iPhone 8 & iPhone 8 Plus.
- iPhone X.
- iPhone XR & iPhone XS & iPhone XS Max.
- iPhone 11 & iPhone 11 Pro & iPhone 11 Pro Max.

CoreML Device Benchmarks



CoreML iOS Benchmarks

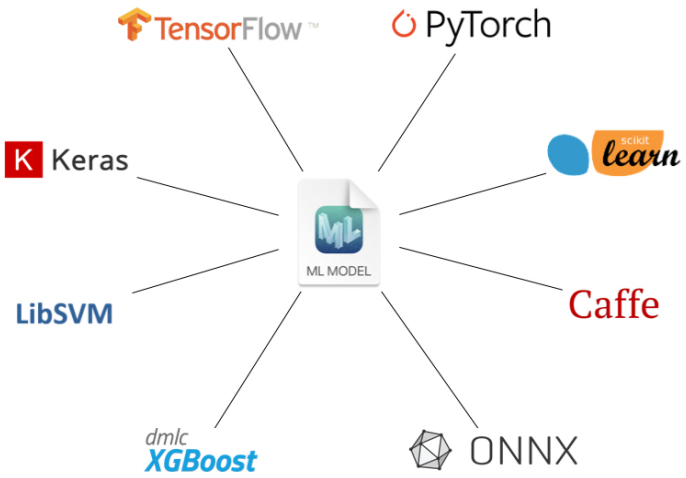


CoreML runs 38% faster on iOS12 compared to iOS11. We couldn't find any information on iOS13.

https://heartbeat.fritz.ai/ios-12-core-ml-benchmarks-b7a79811aac1

CoreMLTools

CoreMLTools is a Python package that allows for the conversion of other models to CoreML. It also provides functionality to read, write, and optimize CoreML models. CreateML does not currently support the creation of updatable models, this means to create an updatable CoreML model, you have to train an updatable algorithm (K-NearestNeighbor or Neural Network) with another ML framework such as Keras or TensorFlow Lite, and convert it into an updatable CoreML model using CoreMLTools.



CoreMLTools supports conversion from these frameworks and formats

Model Type	Supported Frameworks
Neural Networks	TensorFlow 1 (1.14.0+) TensorFlow 2 (2.1.0+) PyTorch (1.4.0+) Keras (2.0.4+) ONNX (1.6.0) Caffe (1.0)
Decision Trees	XGboost (1.1.0) scikit-learn (0.18.1)
Support Vector Machines	LIBSVM (3.22) scikit-learn (0.18.1)
Pipelines (pre/post-processing)	scikit-learn (0.18.1)

https://coremltools.readme.io/docs

TensorFlow Lite

TensorFlow Lite is Google's mobile version of their TensorFlow ML framework. It cannot train models but is instead used for running pre-trained models on-device, similar to CoreML. Unlike CoreML it does not support on-device updates for any models/algorithms at this time.

TensorFlow Lite is optimized for on-device use, by addressing the same constraints as CoreML. It reduces latency, as there's no need to send/receive data from a server, it can perform ML tasks without internet connection, it reduces the memory footprint, and consumes less power.

Unlike CoreML which is restricted to iOS devices, and SwiftUI, TensorFlow Lite is cross platform, and can work with Android and iOS devices, along with embedded Linux and microcontrollers. TensorFlow Lite supports multiple languages including Java, Swift, Objective-C, C++, and Python.

Training the Model

CreateML

We created our classification model in CreateML. The model we created for our PoC app was able to accurately classify our transactions but would not function with the CoreML on-device updates.

CreateML is Apple's model creation tool for Mac, it allows for easy creation of machine learning models. You can import a labeled dataset into CreateML, select the algorithm to use, choose the features to base the classification on (e.g. price, date, time), and choose the target location to output the new model.

The different algorithms in CreateML have some parameters that can be adjusted which will affect the models performance. These can be changed based on the models requirements and the data being used to get a model that's best fit for purpose. CreateML also has a function to automatically suggest the best machine learning algorithm and settings for a labeled dataset.

Keras

Keras is a high-level API interface for the TensorFlow ML framework. Keras provides a simpler way to build and train models in TensorFlow. Keras models still run through TensorFlow which currently has no on-device update functions like CoreML3.

Keras has more options available when training a model, allowing for greater optimization and customization of a model. Unlike CreateML Keras supports the creation of K-Nearest Neighbor and Neural Networks, when creating an updatable image classifier, the model must first be trained with Keras or another third-party ML Framework, and then converted into an updatable CoreML model with CoreMLTools.

Training/Testing Data

The amount of data required for machine learning depends on many factors, the biggest being the complexity of the problem, i.e. the unknown function that best relates your input variables to the output variable. In general, more data is better, however too much data could result in overfitting which will impact the models performance.

Overfitting: Too much data. The model can start to learn from all the noise in the data which can lead to less accurate classifications.

Underfitting: Too little data. The algorithm cannot capture the underlying trend of the data, and so the resulting model does not fit the data well enough.

Our Dataset

When developing our app, we couldn't find any datasets of transactions that could be used to train our model. We had to go through our personal transactions and enter them into a dataset. We were able to train a model that could categorize transactions similar to ours, but because it was trained purely off our data the model was inaccurate when applied to a wider range of transactions. A free dataset of personal transitions was uploaded on kaggle.com during the project and we were able to combine this with our initial dataset to get a more widely applicable model with a training set of 1000 transactions.

Labelling Data

For the data to be useful for our classification model it needed to be labeled as need, want, or joy. We did some research into possible methods for labeling our dataset. Altexsoft.com describes a variety of methods for labeling larger datasets, however with our relatively smaller dataset we chose to manually label the data ourselves.

PROS AND CONS OF LABELING APPROACHES

Approach	Description	Pros	Cons
Internal labeling	Assignment of tasks to an in-house data science team	✓ Predictable results ✓ High accuracy of labeled data ✓ The ability to track progress	× It takes much time
Outsourcing	Recruitment of temporary employees on freelance platforms, posting vacancies on social media and job search sites	✓ The ability to evaluate applicants' skills	× The need to organize workflow
Crowdsourcing	Cooperation with freelancers from crowdsourcing platforms	✓ Cost savings ✓ Fast results	× Quality of work can suffer
Specialized outsourcing companies	Hiring an external team for a specific project	✓ Assured quality	× Higher price compared to crowdsourcing
Synthetic labeling	Generating data with the same attributes of real data	✓ Fewer constraints for using sensitive and regulated data ✓ Training data without mismatches and gaps ✓ Cost- and time-effectiveness	× High computational power required
Data programming	Using scripts that programmatically label data to avoid manual work	✓ Automation ✓ Fast results	× Lower quality dataset

altexsoft
software r&d engineering

https://www.altexsoft.com/blog/datascience/how-to-organize-data-labeling-for-machine-learning-approaches-and-tools/

Measuring Performance

The best way to measure a model's performance varies based on the model's purpose.

Accuracy:

Accuracy measures the ratio of how often the model is right based on all predictions. This is a good measure of the model's performance in cases where the value of false positives and false negatives are the same.

Precision:

Precision measures how accurate the model is out of all predicted positives, how many are actually positive. Precision doesn't care how many needs it misses, it only measures what percentage of transactions it categorized as needs are correct.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$= \frac{True\ Positive}{Total\ Predicted\ Positive}$$

Recall:

Recall measures how many of the actual positives were correctly identified as positives. Recall doesn't care how many wants/joy it incorrectly categorizes as needs. It measures what percentage of total needs it correctly categorizes as needs.

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$= \frac{True\ Positive}{Total\ Actual\ Positive}$$

F Score:

The issue with recall and precision is they only measure one aspect of the model's success. For example, if our model categorized every single transaction as a need, it's Recall for needs would be 100%. This is clearly not ideal which is why we should use an F Score to judge our models accuracy.

The F Score combines precision and recall into an effective metric for measuring a model's performance. The F1 Score, or balanced F Score is used when both precision and recall are equally important. When either precision or recall are more important, the formula can be adjusted to provide a more relevant F Score. This is done by changing β so that recall is considered β times as important than precision.

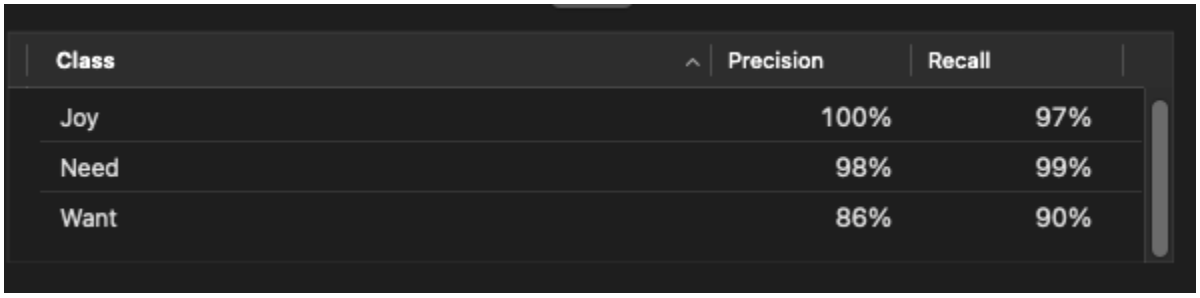
$$F - score = \frac{(1 + \beta^2)\, precision \times recall}{\beta^2\, precision + recall}$$

$$F1 = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

Algorithm selection

For our PoC app CreateML automatically suggested using the boosted tree algorithm for our dataset. We did some testing of our own with other algorithm options and measured the performance of each. For our relatively small dataset, the results were showing better F1 scores for the Logistic Regression algorithm, and Boosted Tree was a bit lower. However after testing the classification with more varied transactions, we had better results using the Boosted Tree algorithm.

Logistic Regression:

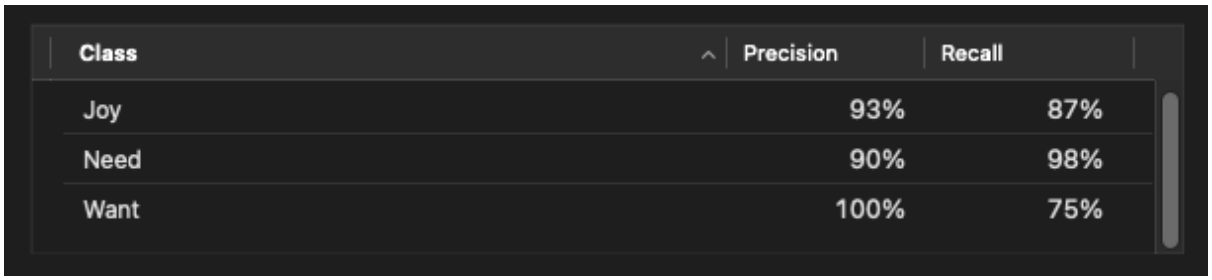


Class	Precision	Recall
Joy	100%	97%
Need	98%	99%
Want	86%	90%

F1 score for Logistic regression:

Joy = 98	Need = 98	Want = 92

Decision Tree:

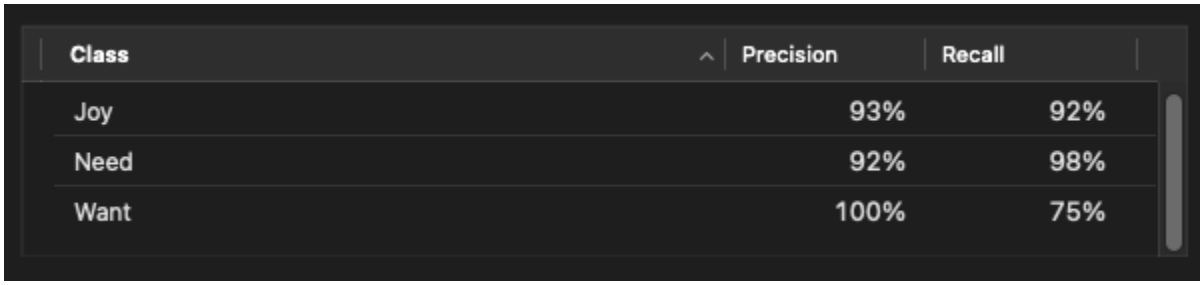


Class	Precision	Recall
Joy	93%	87%
Need	90%	98%
Want	100%	75%

F1 score for Decision Tree:

Joy = 89	Need = 93	Want = 85

Boosted Tree:



Class	Precision	Recall
Joy	93%	92%
Need	92%	98%
Want	100%	75%

F1 Score for boosted tree:

Joy = 92	Need = 94	Want = 85

App Integration

Getting Predictions

CoreML allows for easy integration of ML Models into SwiftUI apps. CreateML lets you easily export a .mlmodel file, and CoreML tools can be used for converting other formats to CoreML, or creating an updatable model.

A model can be easily imported into XCode by dragging the .mlmodel file into the XCode project. This will automatically create a class for the model that can then be used with the swift interface.

Once the model class exists in the project, you can create an instance of the model.

```
let model = PersonalTransactions()
```

The model also requires the correct input features to generate a prediction. The transaction categorization model we created for our PoC app used vendor name, price, and date. We had to ensure the price variable was a double, and the rest of the input features were strings. After formatting the input features, the model can be used to give a prediction in a do/catch block. The model will output the predicted category as a string, and has the option to output the confidence of the prediction as well.

```
do {
    let prediction = try model.prediction
                        (description: transDesc,
                        price: transPrice,
                        date: transDate)
} catch {
    // something went wrong!
}
```

On-Device Updates

On-device updates are a new feature of CoreML3, allowing for updatable models to be updated on-device at runtime. This allows for models to become personalized and more accurate for a specific user.

The current CoreML API only supports image classification models as updatable models leaving us unable to implement this functionality with our PoC app.

How it would work for our app

The app can currently predict the categories of users transactions and store them in a transaction list, if the predicted category is wrong the user can manually correct it. If CoreML's MLUpdateTask supported tabular classification models, this newly corrected transaction could be used to update the model for the individual user.

An ideal scenario would be a user installs the app with the initial categorization model. This model would need to be trained with a large amount of labelled transactions to get accurate categorization that could correctly categorize most transactions.

The user would enter a transaction into the app and the app would automatically categorize these as a Need, Want, or Joy. If the app incorrectly categorizes a transaction, the user can manually correct the category. This newly corrected transaction would then be used to update the model on the user's device using an MLUpdateTask. The model will learn from the users data and become more accurate for how this individual user categorizes their transaction.

How to use CoreML on-device updates

This is currently only supported for updatable image classifier models. We couldn't find any information on whether future updates of CoreML will add support for on-device updates with tabular classification models.

To update the model, each feature used by the model needs to be wrapped in an **MLFeatureValue**. All the feature values of a transaction get wrapped in an **MLFeatureProvider**. All the feature providers must then be wrapped in an **MLBatchProvider** which can then be used to update the model with the new transaction data.

After wrapping the data to be used, the model can be updated by creating an **MLUpdateTask**. This requires an **MLBatchProvider**, the location of the model to be updated, an **MLModelConfiguration** if applicable (handles model settings, such as using GPU instead of CPU for calculations), and a completion handler with a MLUpdateContext parameter.

```
// Create an Update Task.
guard let updateTask = try? MLUpdateTask(forModelAt: url,
                                         trainingData: trainingData,
                                         configuration: nil,
                                         completionHandler: completionHandler)
    else {
        print("Could't create an MLUpdateTask.")
        return
}
```

Retraining the model on-device

We were looking for a way to use on-device updates for a tabular classification model like our PoC app. We found that CoreML can be used to train a model on a device. If the device has access to the complete dataset, the model can be retrained on the user's device. This could achieve similar functionality to the on-device updates, as the users personal data could be added to the initial dataset before retraining the model.

The initial dataset will need to be either bundled with the app, or streamed from the cloud. This will either increase the memory footprint of the app, or require an internet connection for updates, also limiting the speed of retraining by download speed. This method is much slower than CoreML's on-device updates, which solves these issues of speed, size, and network connection.

Recommendation

In relation to personal finance, Machine Learning can bring value through the ability to automatically categorize transactions and learn from how an individual user categorizes their transactions. In many other contexts a general categorization model may be effective, however individuals categorize their spending differently, e.g. one person may categorize coffee as a want and another would say they need it. This makes a model capable of learning from the user more important.

CoreML 3 is optimized for on-device performance. It minimizes memory footprint, uses less power, keeps private user data on device, functions without internet connection, and allows for on-device updates to get a personalized categorization model for individual users. Currently the CoreML MLUpdateTask supports only two algorithms, restricted to image classifiers by the CoreML API. This means that tabular classification models like the transaction categorization model we created for our PoC app cannot make use of these on-device updates.

It is possible to retrain the model on the device but requires access to the initial dataset used to train the model, and retrains it with the user's new data included. This work around allows for other models like tabular classification to be updated on device while maintaining privacy of user data, however it loses some of the other benefits from the updatable CoreML models.

We wanted to know whether future updates of CoreML would allow for more algorithms to be updatable. This would make it much easier to implement a transaction categorization model capable of learning from user corrections to get a personalized model that can accurately predict how an individual would categorize their spending. After checking various sources we could not find any information on future updates.