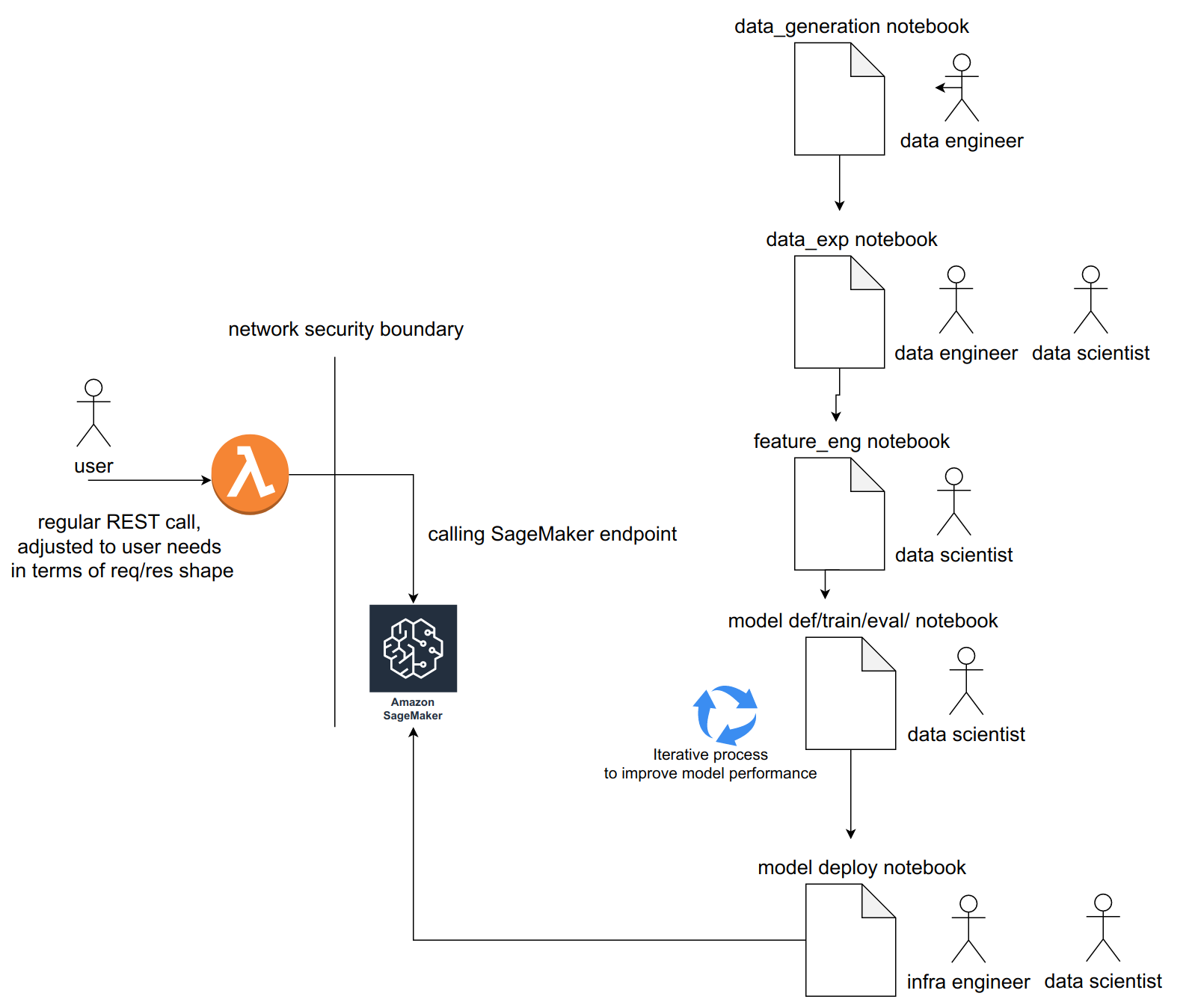
The Lightweight IBM Cloud Garage Method for Data Science

Architectural Decisions Document for fitness wearable project

­

Architectural Components Overview



Reference Architecture

## Data Source

### Technology Choice

I wanted to explore if fitness wearables improve people's discipline when it comes to doing sports.

Unfortunately, such data is not available as wearable manufacturers probably don't want to disclose it even in anonymized form.

As a result I decided to generate data myself. I did it with Python and several libraries to randomize the data: numpy, faker and random.

I also have some correlation between different columns, but as is visible in feature exploration module, those correlation are also not very noticeable because of the degree of randomization.

### Justification

As I don’t have datastores or databases as a source, it’s easier to just use the same language I use for the rest of the project to generate the data, hence Python and related libraries.

## Discovery and Exploration

### Technology Choice

I used pandas to load my csv data as a dataframe and matplotlib/seaborn for plotting.

### Justification

Very comvenient to work with in jupyter notebook environment.

## Data quality assessment

### Technology Choice

I used pandas and visualization(matplotlib/seaborn) to identify problmes with data. For example, I used pandas dataframes with filters to find empty cells, identify sparsity. I used visualization to identify correlation between features. THis was important as my data was synthetic

## Feature engineering

### Technology Choice

pandas, sklearn encoders

### Justification

For feature engineering I had to:

1. Remove some columns I was not interested in(pandas);

2. Encode categorical columns;

Overall in the project I used two types of encoding:

1. one-hot encoding;

2. label encoding;

Categorical features should be encoded because most models are a lot more comfortable working with encoded features rather than non-numeric ones. Some models will fail right away if non-numeric features are used(keras+tensorflow), some others will not, but will probably perform poorly.

## Model algorithm

### Technology Choice

Random forest for traditional model.

Two layer ANN.

Both are tuned with different sets of hyperparameters.

### Justification

Let's start with traditional classification model. Among the classification methods I selected Random Forest. Comparing it with other algoriths I know:

Naive Bayes assumes there's no correlation between features. While we saw in data exp module that the correlation is very low, I know that in data generation I made some features somewhat dependent on others.

As for SVM, it might be better for binary classification problem. However, I've leared that SVM is quite picky about the input data(features) and also doesn't produce probability, so there's an extra step to calculate it. Also, some specific issues SVM handles better than Random Forest, like high number of features, is not applicable. And Random Forest is also in most cases faster to train.

Random Forest is hyperparameter tuned with:

'n\_estimators', 'max\_features', 'max\_depth', 'min\_samples\_split', 'min\_samples\_leaf'

ANN

Overall, our model has two layers. It’s the simplest we can get to have it classified as deep learning model. It’s also sufficient, as it turns out, for the project of this complexity. Hidden layer will have ReLU activation, output layer will have Sigmoid activation. This is the most common approach as Sigmoid suffers from vanishing gradient problem and is usually not used for hidden layers. Sigmoid is best suited in output layer of binary classification.

optimizer, init function, epochs and batch size are all hyperparameters and are tuned in the process.

## Model performance indicator

### Technology Choice

f1 score

### Justification

I started with accuracy as most basic usecases suggest.

However, let's look at other metrics as accuracy is not a great measure of classifier performance when the classes are imbalanced. We remember that we have twice as many sticklers as non-sticklers.

With that said, we need to identify a metric that best fits our usecase.

Apart from accuracy, I looked at precision and recall. All of them are good metrics for binary classification problem.

However, before we even start, let's think of usecases when precision and recall are applicable. Precision is best suited for situations when we need to be very sure about identified positives(FP should be low). Recall is well suited when we need to identify as many true positives as possible(FN should be low).

In our case, we don't really know if our model will be used with aiming for high precision or recall. Is there a middle groud? It turns out, there is! It's called F1 score.