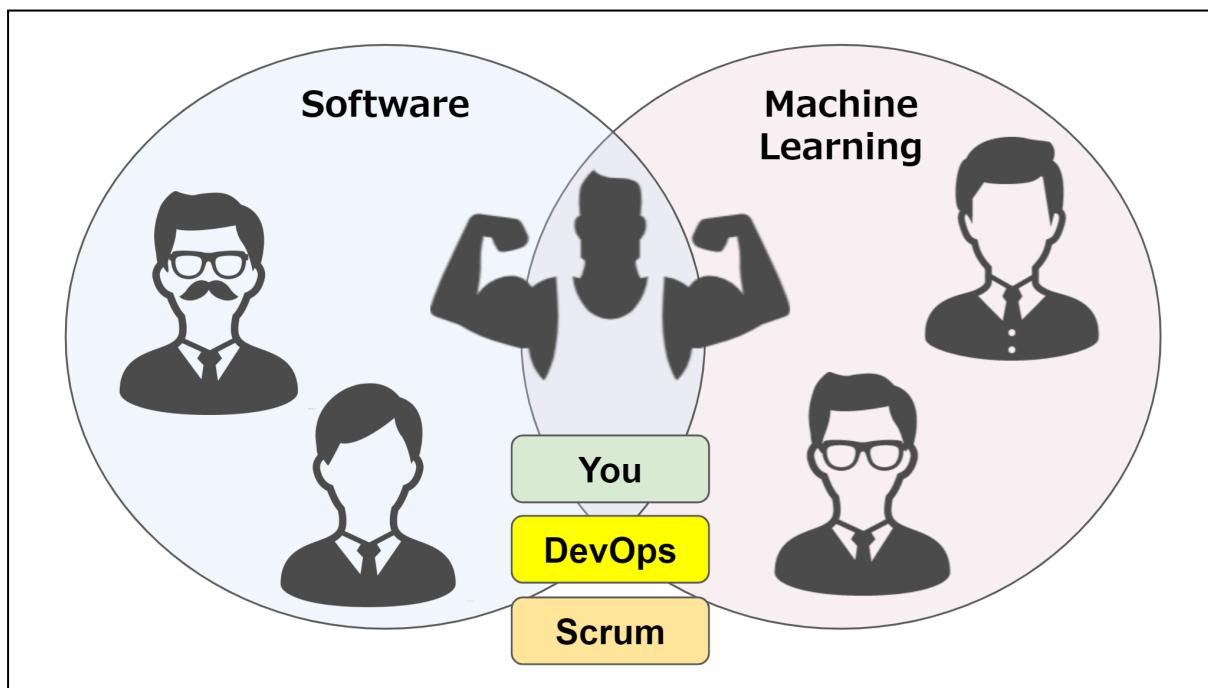


ソフトウェアエンジニアが 機械学習プロダクト開発を始めるときの心構え



1.0版

2022年8月1日
永和システムマネジメント Future Design Project
チームBoat一同

はじめに	3
扱わない内容	3
想定している読者	3
背景	6
ソフトウェアエンジニアの強みを活かそう	7
Step-1: 開発を始める前に	10
機械学習を勉強する	10
できること / できないこと	10
関わる職種	11
機械学習で使う数学	12
予測精度は日々変化する	15
ソフトウェアエンジニア文化との違いを知る	16
Step-2: 開発が始まったら	18
既存のライブラリやサービスを上手く使う	18
早めにやること / やらないことを見分ける	19
早めにデータ収集の目途をつける	20
早めに評価尺度を考えておく	20
パラメータチューニングは後回しでも良い	20
機械学習システムとアプリを分離する	21
機械学習モデル開発サイクル	25
MLOpsの仕組みを準備する	25
スクラムでリズムを安定させる	27
機械学習モデル開発サイクルをスクラムで回す	27
適切なスプリントの長さを考える	28
作業の完了条件は見積もり可能なものにする	29
Step-3: 次の一歩	32
機械学習のエキスパートチームと共に働く	32
参考文献	33
使用したライブラリやツール	33
開発リポジトリ	34
開発メンバより	34

はじめに

「機械学習のプロダクト開発を始めたい！」

そう思った際に

「でもどうやって始めたらいいのだろう？」

と立ち止まってしまうことはないでしょうか。

私たちは、機械学習初学者のソフトウェアエンジニアによるチームを結成し、1年間に渡って機械学習を活用したプロダクトの開発を行ってきました。

そして、そこで実際に手を動かしたことにより、たくさんの学びを得ることができました。

本書では、これから機械学習プロダクトを開発することになったソフトウェアエンジニア(= あなた)が、開発を始める際に持っておくべき心構えについて、私たちの経験を踏まえてお伝えします。

扱わない内容

本書は、機械学習プロダクト開発の進め方を書いたチュートリアルではありません。開発手順等は、本やインターネットで調べてみてください。

想定している読者

本書で主な読者として想定しているのは、以下のようなスキルセットを持ったソフトウェアエンジニアです。

- スクラムの経験もしくは知識がある
- 普段からコーディングしている
- Git/GitHubを使っている
- クラウドサービスを使っている
- コンテナを使っている
- CI/CDを導入している

ただしこれは、限られた紙面の中で記述できるスコープをある程度制約するためであり、上記に該当しないソフトウェアエンジニアや、更にはソフトウェアエンジニア以外の人にとっても有益な内容であると確信しています。是非ご一読ください。

[実例] 私たちが開発した機械学習プロダクト

1) AI検温計

顔の皮膚温度から、機械学習のロジックを用いて、実際の体温を予測する検温計。

取り組んだ機械学習の問題

・回帰による予測(教師あり学習)

実際に使った技術/アルゴリズム

- ・線形回帰
- ・ランダムフォレスト
- ・GBDT
- ・ニューラルネットワーク
- ・画像認識



2) 書籍レビュー投稿アプリ

本のレビューを投稿・共有できるSlackアプリ。

蓄積したレビューのデータをもとに、機械学習のロジックを用いて、ユーザごとに次に読む本をおすすめする。

取り組んだ機械学習の問題

- ・推薦(レコメンデーション)

実際に使った技術/アルゴリズム

- ・協調フィルタリング
- ・内容ベースフィルタリング



ホーム メッセージ ワークスペース情報

読書レビュー共有アプリ「Bee (Book Erabu Eiwa)」

読んだ本のレビューを投稿して、データ蓄積に協力お願いします

Beeは、FDOが開発・提供する、本のレビュー共有アプリです。

仕事で役立った本のレビューを投稿・共有できます。

データがたまればたまるほど、AIはより賢くなりあなたに合ったおすすめの本をお伝えすることができます。

書籍購入制度で購入した本などのレビューを投稿してみましょう！！。

岡本卓也さんへのおすすめ本

最新の推薦データ : 2022/07/14 02:00:45

Googleのソフトウェアエンジニアリング

Titus Winters, Tom Mansreck, Hyrum Wright

ISBN-9784873119656

[Google Booksで見る](#)



なるほどデザイン

筒井美希

ISBN-9784844365174

[Google Booksで見る](#)



基礎から学ぶReact Native入門

WINGSプロジェクト 中川 幸哉

ISBN-9784798169552

[Google Booksで見る](#)



興味なし

[この本のレビューを見る](#)

興味なし

[この本のレビューを見る](#)

背景

近年、社会的にAIや機械学習への関心が高まっており、「機械学習を活用したビジネスをやりたい」「機械学習の技術を使ってプロダクトの魅力を向上させたい」と考えるケースが増えています。

しかし、機械学習に馴染みの薄いユーザ企業にとっては、「そもそもどうやったら機械学習を始められるのか?」「どこに相談すればいいのか?」といった点から戸惑うことになり、まずは馴染みのある身近なITベンダに相談を持ちかけるケースが多いのではないでしょうか。

そのようなケースでは、ITベンダとしても最初から機械学習エキスパートを巻き込んで本格的なプロジェクトを開始することは難しいはずです。まずは社内のソフトウェアエンジニアだけでPoC(Proof of Concept:概念実証)レベルのトライアルを行って、機械学習についての経験と理解を獲得する必要があります。

また、本格的なプロジェクトのフェーズに至り機械学習エキスパートが参画するとなった場合でも、プロダクト開発をそのまま丸投げできるわけではありません。チームとして機械学習エキスパートと上手く協働することが重要になり、そのためにはソフトウェアエンジニア自身も機械学習という特殊な領域についての知識を持っておく必要があります。

そこで本書では、ソフトウェアエンジニアが、自分たちの手で機械学習に取り組むときの心構え(ポイント)についてお伝えします。

[コラム] 機械学習エキスパートの雇用は難しい?

機械学習エキスパートとしては、データサイエンティストやMLエンジニアといった職種が挙げられます。彼らの仕事は、データを分析して適切な機械学習モデルを作ったり、作成されたモデルを実運用する仕組みを作ったりすることで、まさに機械学習プロダクトの開発に必要な人材です。

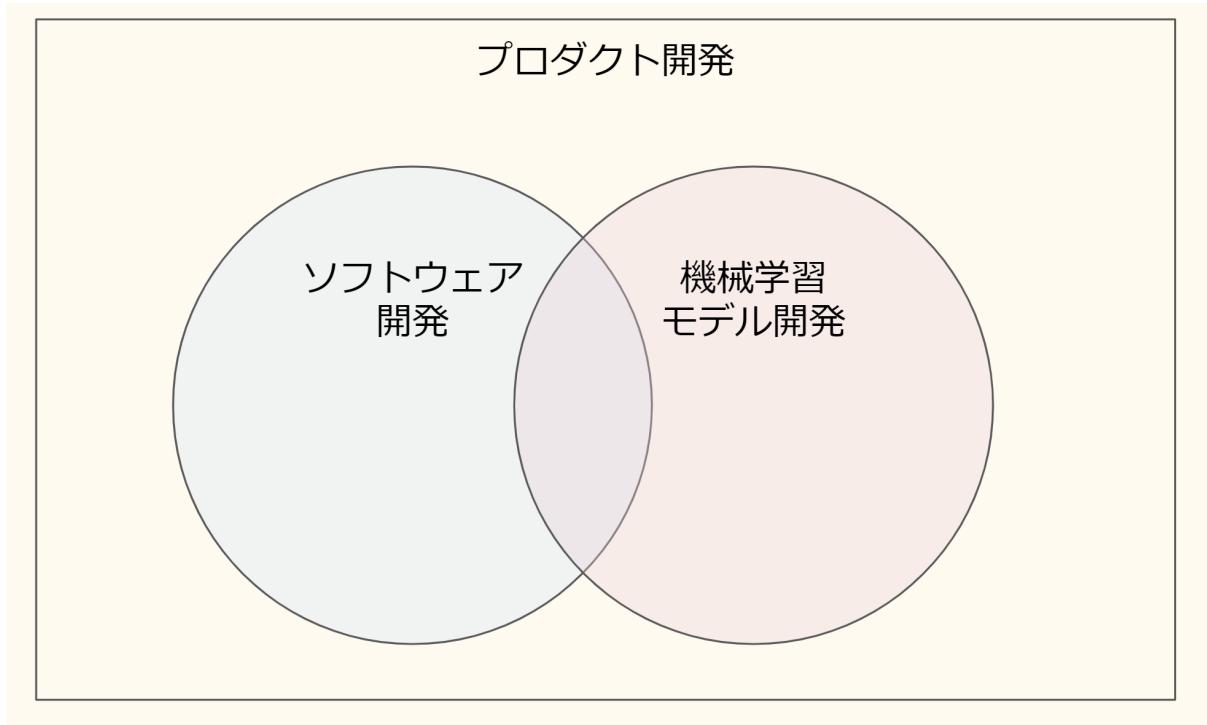
しかし、最初から彼らをあなたのチームに迎え入れるのは、簡単なことではありません。そもそも人材を見つけることが難しかったり、たとえ見つかったとしても条件面で折り合いがつかなかつたりするかもしれません。

また、ITベンダにとって、本格的に機械学習ビジネスに参入する前の段階でエキスパートを固定的に抱えることは難しいのではないかと思います。

だからこそ、私たち(そして、あなたたち)ソフトウェアエンジニア自身が、機械学習に向き合い挑戦する価値があるのではないかでしょうか。

ソフトウェアエンジニアの強みを活かそう

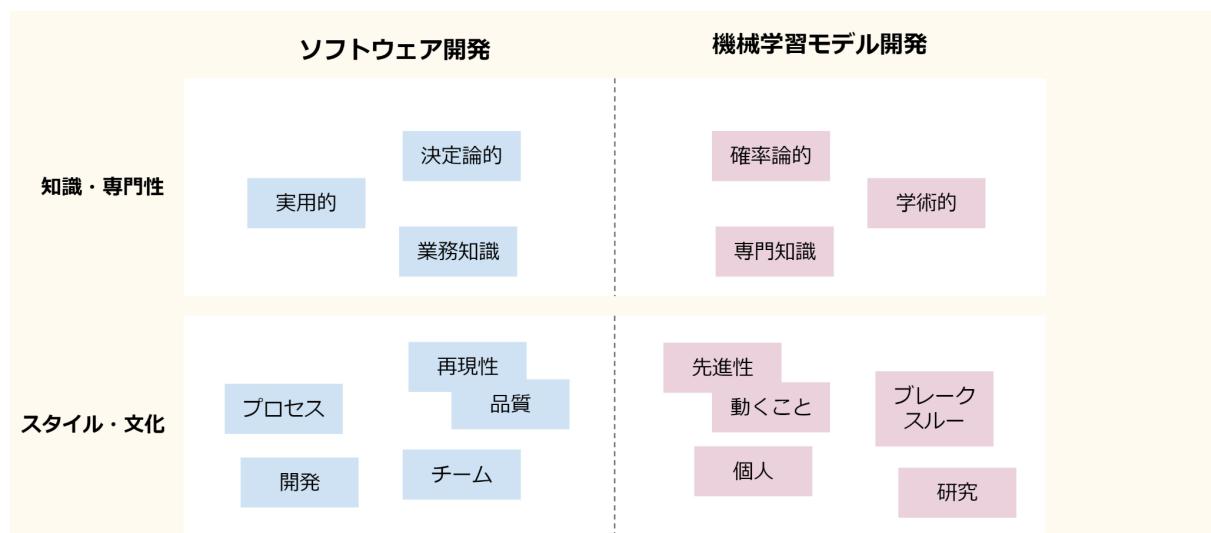
機械学習は一般的なソフトウェア開発とは別の領域です。どちらもプログラムを介して実現されるので共通する部分はありますが、その距離は近くありません。



ソフトウェア開発と機械学習モデル開発の領域

例えば、機械学習のコアとなる技術として、数式で表現される機械学習アルゴリズムが存在します。機械学習プロダクトを実装する際、これらの機械学習アルゴリズムを完全に理解する必要はありませんが、ある程度の概念や用語については把握していないと使いこなすことができません。

また、求められる知識だけでなく、開発における作業の進め方も異なります。例えば、機械学習では予測結果は確率論的に出力されます。(正解率100%の機械学習モデルを作るのは困難です)出力された処理結果を見てOK/NGが一律に判定できるソフトウェアとは違い、機械学習の世界では、実装した処理を評価することも一筋縄ではいかないのです。



ソフトウェア開発と機械学習モデル開発の特徴

あなたが初めて機械学習の本を開いたとき、一般的なソフトウェア開発との違いに目を回すかもしれません！

とは言っても、重なる領域・活用できるプラクティスがない訳ではありません。例を挙げると、DevOpsとスクラムです。ソフトウェア開発の経験で馴染みあるこれらのエッセンスは、機械学習プロダクト開発でも、うまく働きます。あなたには、ソフトウェア開発で培った、強力な味方がついていることを忘れないでください！



DevOpsとスクラムは機械学習プロダクトでもうまく働くことが期待されますが、同時に機械学習の分野では発展途上でもあります。機械学習エキスパートの中には、これらをほとんど知らない人も多いです。ソフトウェアエンジニアの得意分野とも言えるこれらの領域を機械学習に適応することは、プロダクトがスケールし、データサイエンティストなどがチームに加わった際も、価値のあるものとなるでしょう。

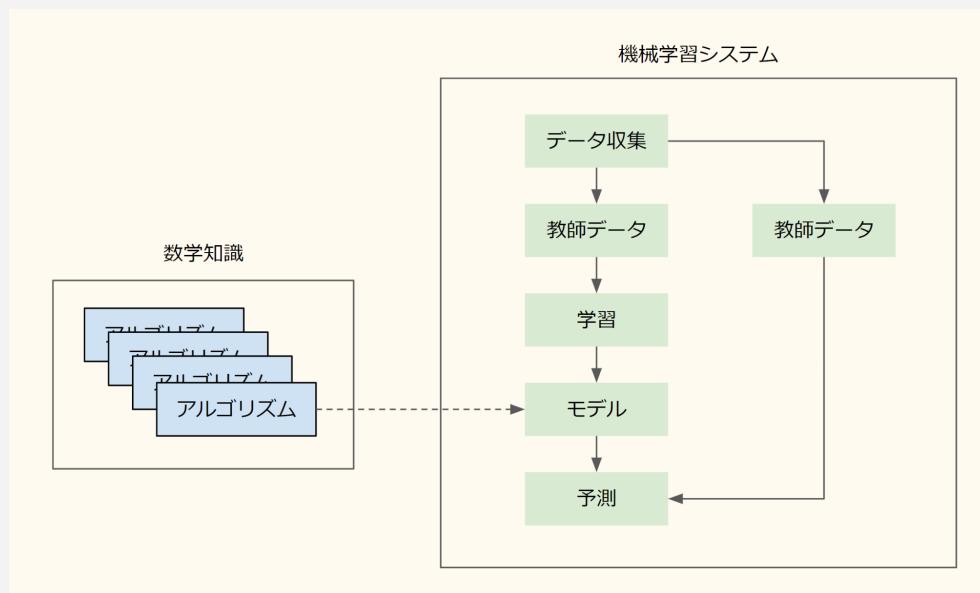
次の章からは、これらの経験をどのように活かせば良いか、具体的なコツと共にご紹介していきます。

[コラム] 機械学習のアルゴリズムとは？

あなたは機械学習について、全くの初心者かもしれません。実際の勉強方法は「機械学習を勉強する」で述べますが、この時点での機械学習へのイメージのために一般的な流れを説明します。

機械学習では、大量のデータをもとに、機械学習モデルを作成(学習)します。学習したら、実際にデータを入力して、なんらかの値を予測します。機械学習モデルは、さまざまな数学的アルゴリズムによるもので、適応する領域によって向き不向きがあります。

解きたい問題に対して、適切なアルゴリズムを選択することは、機械学習において重要なポイントです。



機械学習の流れ(教師あり学習の場合)

Step-1: 開発を始める前に

機械学習を勉強する

まずは機械学習とは何であるのか？について知っておくべきことがいくつかあります。

できること / できないこと

機械学習と聞くと、人間には不可能なことができるすごい仕組みであるかのようなイメージがありますが、そんなことはなく当然ながら、できること / できないことがあります。

機械学習プロダクトの開発に長い時間を費やしたにも関わらず、終わってみれば実は機械学習を使う必要はなかった、というケースがあるかもしれません。このようなケースを防ぐためにも、できること / できないことを事前に知っておくことが大事です。

できること(得意なこと)の例

- 大量のデータの中からルールを見つけ出し、そのルールに基づいた予測を行うこと
- 人が行うと時間がかかる複雑な計算(予測や最適化)を高速に扱うこと
- 同じ作業を繰り返し行うこと

できないこと(苦手なこと)の例

- 少ないデータで精度良く予測すること
- 企画立案などの新しいアイデアを生み出すこと
- 状況の変化に応じて、柔軟な対応をすること
- 自発的に考えること

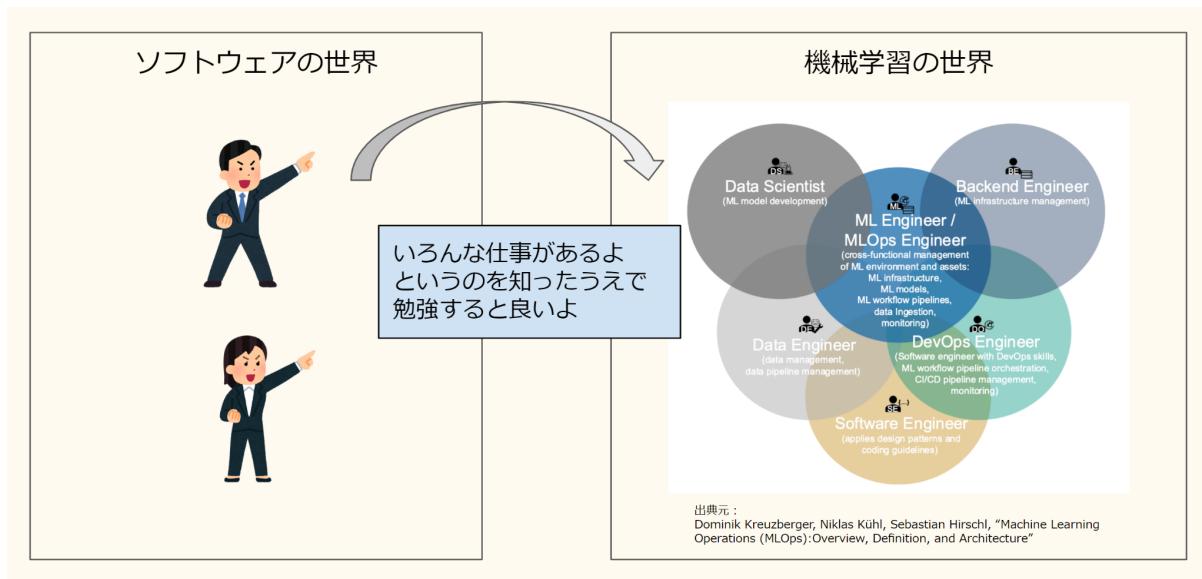
世の中に公開されているような機械学習の活用事例も併せて調べると、できること / できないことを、より理解できるでしょう。

[実例] 機械学習で何ができる？

私たちは、プロジェクトの最初3ヶ月を勉強期間と位置付け、本やインターネット上の学習コースを活用し、機械学習を勉強しました。その中で、AI・機械学習の活用事例についても勉強し、どんなケースに機械学習が有効か、を学びました。

例えば、AmazonやNetflixなどのおすすめ機能を持つサービスには、機械学習アルゴリズムが用いられています。それを学んだ私たちは、社内で本のレビューデータを収集し、機械学習アルゴリズムを使っておすすめサービスを作成してみようと考え、書籍レビュー投稿アプリを開発しました。

関わる職種



機械学習の分野にソフトウェアエンジニアが足を踏み入れる場合、どういう職種と関わるのか、を知っておく必要があります。上の図のように、機械学習の世界にはデータサイエンティストやMLエンジニアなどの職種が存在します。(ただし詳細な定義は企業や組織によって異なることもあります)

- データサイエンティスト: データを分析し、機械学習モデルを作る
- MLエンジニア: 機械学習モデルを実運用する仕組みを作る

私たちソフトウェアエンジニアだけで、機械学習の技術を取り入れたプロダクトを開発していく場合、このMLエンジニアやデータサイエンティストが担う部分を、本やインターネットなどから情報収集し、ノウハウを活用しながら実践していくことになります。

[実例] 本を選ぶときのポイント

私たちは、最初はとにかくたくさんの本を読んでいました。が、思い返すと、理論に寄りすぎていって(その時点では)直接役に立たないものもありました。

例えば、Deep Learningの理論について詳しく述べている本は、機械学習のプロダクトを開発したい私たちにとっては優先度が低く、後回しにするべきだったと思います。逆に、ソフトウェアエンジニアが機械学習を始めるときに最適な本もありました。例えば、「仕事ではじめる機械学習」[2]という本は、理論は少なめで、具体的な進め方や流れについて詳しく書かれていたので、私たちが最初に読む本として適していたと言えます。

最初から職種の違いについて知っているれば、自分たちにあった本をもっと効率的に選べたのは、と思います。

機械学習で使う数学

「機械学習に取り組むには数学の知識が必要」とよく言われますが、実際のところはどうでしょうか。

最近の機械学習ライブラリやマネージドサービスは非常によくできていますが、特別な数学の知識なしでも一通り動かすことができるようになっています。とりあえず機械学習に触ってみたい/動かしてみたいなど、目的によってはこれで十分なケースもあります。

ただしそんな場合でも、基礎的なレベルの数学は、勉強しておくことをおすすめします。具体的には、確率と統計の基礎、線形代数、微積分あたりの知識があると良いでしょう。

これらを勉強しておくことで、機械学習モデルが出力する結果に対して「なぜこうなったのか」を理解したり、「より精度を上げるにはどうするべきか」を考えることができます。

[実例] 機械学習で使われる数学知識

書籍レビュー投稿アプリでは、学習データを作成するために日本語で書かれた文章を比較したり類似度を計算する必要がありました。数値計算を行うコンピュータで自然言語の文章をどうやって扱うのか想像もつかなかったのですが、数学の概念であるベクトルによって文章を数値化することができ、そうしていったん数値化されたデータであれば、あとはコンピュータで自在に処理できるようになります。

例) 日本語で書かれた文章の類似度を計算する

- (a) 「私はリンゴが好きです」
- (b) 「私はリンゴが嫌いです」
- (c) 「私はミカンのことがとても好きです」

①文章をベクトル化する

色々な手法がありますが例えば単語の出現回数を元にしてベクトル化することができます。中学や高校で習うような平面や立体で用いるベクトルは2次元や3次元ですが、文章をベクトル化する際には100次元やそれ以上の、人間が直感的には把握できないようなベクトルを使います。

「私はリンゴが好きです」のベクトル: $\vec{a} = (a_1, a_2, \dots, a_n)$

「私はリンゴが嫌いです」のベクトル: $\vec{b} = (b_1, b_2, \dots, b_n)$

「私はミカンのことがとても好きです」のベクトル: $\vec{c} = (c_1, c_2, \dots, c_n)$

②類似度を計算する

ベクトルはコサイン類似度という計算手法を使って「どのくらい似ているか」を数値で評価することができます。

$$\cos(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|} = \frac{\sum_{i=1}^n a_i b_i}{\sqrt{\sum_{i=1}^n a_i^2} \sqrt{\sum_{i=1}^n b_i^2}}$$

この数式により(a)~(c)の文章のお互いの類似度を計算すると以下のようになります。

	a	b	c
a	[1.	0.657	0.937]
b	[0.657	1.	0.317]
c	[0.937	0.317	1.]]

つまり

「私はリンゴが好きです」と「私はリンゴが嫌いです」の類似度は0.657

「私はリンゴが好きです」と「私はミカンのことがとても好きです」の類似度は0.937

「私はリンゴが嫌いです」と「私はミカンのことがとても好きです」の類似度は0.317
と計算されました。

対角線は同じ文章同士なので類似度は1になります。

上記から、一番近い文章は「私はリンゴが好きです」と「私はミカンのことがとても好きです」という
ことが計算により求められました。

この仕組みを知った時には、「なるほど、これが数学知識を使うということか！」と深く納得するこ
とができました。

[コラム] 機械学習と数学(平鍋健児さん寄稿)

私が考えるソフトウェアエンジニアとしての数学の必要性を、分野ごと、重要度順に書いてみま
す。

必須:

- ・確率と統計の基礎(現在の高校の「数学B」程度)

推奨:

- ・線形代数(高校「数学C」および大学一般教養の基本部分)
- ・微積分(高校「数学II」程度)

さらに:

- ・数値計算、最適化数学、微分方程式

まず何が何でも確率・統計の基礎です。最近は高校の数学課程でこの分野の扱いが大きくなつ
ており、分布や可視化(分布のグラフや散布図、箱ひげ図)さらに、仮設検定なども含まれるよう
になりました。機械学習プロジェクトにおいても、得たデータや結果の精度を可視化したり、結果
の説明したり、議論して方向を合意したり、コミュニケーションにはこの分野の知識が必須にな
ります。特に用語は全員で理解している必要があるでしょう。

(Pythonプログラミングでは、pandasや matplotlib、streamlitあたりに相当します)

次に線形代数です。データはほとんど表形式で扱うことになるでしょう。表は行列そのものです。
このデータを加工したり計算したりしてモデルを作っていくことになります。データの概要を理解す
るのは上記の統計の基礎ですが、このデータに潜む本質的な情報を発掘したり、予測を行うモデ

ルを作成するには線形代数、すなわち、行列とベクトルの理解が必要となります。ただし、アルゴリズムの中身を理解することができなくても、名称と特性だけ知っていればプログラミングとしては可能でもあります。また、最近の線形代数では、データサイエンスとの接合点として特異値分解(主成分分析)や離散フーリエ分析などの次元削減を教えるようです。データを扱う部分の重要性が大きいことの現れでしょう。

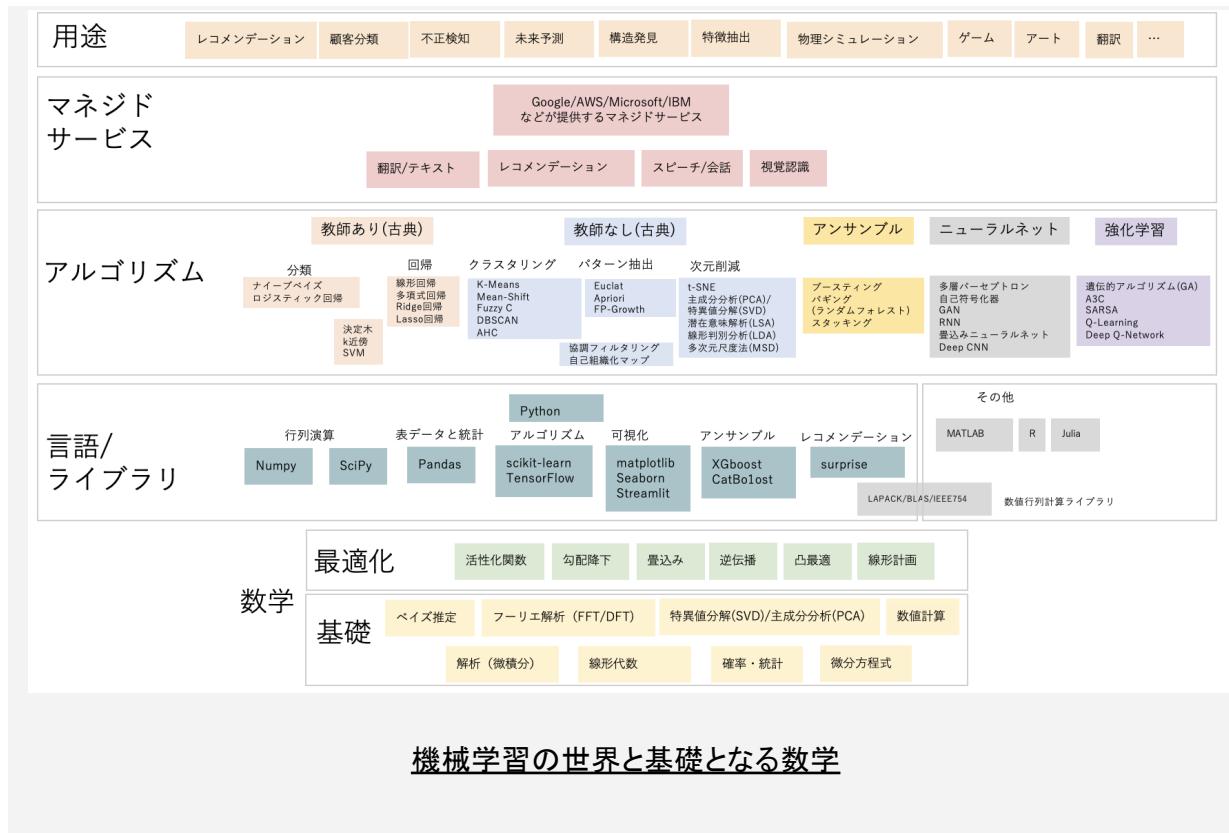
(Pythonプログラミングでは、NumPy、numpy.linalg、SciPy のあたりです)

微積分ですが、ここは最適化問題の基礎になります。数値計算はほとんどの場合、繰り返しと近似によって行われ、その最適化のためには各計算ごとに勾配を得て、収束方向に計算を進めていきます。よって、この勾配を得るための微分が重要になります。また、もともと機械学習の問題が、「何かを最小(大)にせよ」と定式化されることが多く、その解法の基本的な理解にも、微積分が必要になります。さらに、機械学習の扱う領域として物理的要素が含まれる場合には、微分方程式の知識も要求されるでしょう。

これら数学の上に、機械学習の各種アルゴリズムが構築されています。中身に立ち入らなければ、推奨分野以上の数学について必要性は少ないでしょう。しかし、ソフトウェアエンジニアとしては、数学の上に乗っている、各種アルゴリズムの概要は知っている必要があります。アルゴリズムの中身について深く知ることが難しくても、どんな用途に応用され、どんな特性があるかについての理解は必要です。プログラマであれば、プログラミング言語とライブラリを利用して、その中身に踏み込まなくても「言葉」(クラスや関数名)の組み合わせでアルゴリズムを利用することができます。しかし、用途や特性を知らないとそのアルゴリズムを利用すべきかどうかの判断ができません。

(Pythonプログラミングでは、scikit-learn、TensorFlow あたりです)

最後に、残念ながら数学は得意不得意がはっきり別れる分野でもあり、苦手な人が深く勉強するには膨大な時間と努力が必要となり現実的ではありません。しかし、統計の基礎、可視化、アルゴリズムの名前と概要までは、ぜひ理解しておくことをおすすめします。この理解があれば、機械学習プロダクト開発の中では、ビジネス側とのコミュニケーションや、データ収集、ソフトウェインフラやツール整備など、アルゴリズム周辺以外の他分野でもエンジニアとして活躍する場面が多くあります。



予測精度は日々変化する

機械学習プロダクトでは、生成したモデルから予測が得られた後も、利用データなどを収集し学習を続けるのが一般的です。そのため、いったんモデルが決定した後でも、出力される予測は変動し続けます。

またその場合、データが増えたからと言って単純に予測精度が向上するとは限りません。例えば、新しく収集したデータに異常な値が含まれていれば、予測精度は低下することもあります。

そのためにも後述するMLOpsのサイクルの中で、予測結果を注意深く観察し検証し続けることが大事です。

[実例] 日々の変化を監視する仕組み

私たちも、書籍レビュー投稿アプリの開発において、予測精度やユーザアクティビティを継続的に確認していました。

具体的には下記のように、毎日定期的に自動実行される処理の中で監視対象のデータを収集し、データベースに蓄積するとともにチームのSlackに投稿する仕組みを構築しました。そして、ここから得られる情報を元に、開発方針やタスクの優先度などを決定していました。



ソフトウェアエンジニア文化との違いを知る

機械学習エキスパートの多くは、ソフトウェア開発に関する知識や経験を豊富に持っている訳ではありません。このため、彼らとソフトウェアエンジニアとでは文化のレベルから違っていることもあります。

例えば、ソフトウェアエンジニアが重視する、メンテナンス性や再利用性の高いコードを書くということは、機械学習エキスパートにとっては最優先の価値ではありません。その代わり、データの分析結果から次のアクションを導き出したり、予測モデルの精度を上げるために試行錯誤を繰り返すことを重要視します。

ソフトウェアエンジニアの文化も機械学習エキスパートの文化も、それぞれが持つ強みです。違いを理解しあいが得意とする部分を融合させることで、機械学習プロダクトを成功に導くことができます。

[実例] ソフトウェアエンジニアの譲れない習性

機械学習では、書いたものをその場ですぐに試せるなどの利便性から、Jupyter Notebook [17] がよく使われています。私たちも最初はJupyter Notebookを使っていたのですが、コードを変更した場合にファイル差分が見辛いなどの理由で、構成管理ツールであるGitとの相性が良くなかったです。しかし、ソフトウェアエンジニアである私たちにはGitによる管理を諦めるという選択肢はなく、後述のjupytextやVS Code Python Interactive Windowに移行しました。

NotebookファイルをGitで差分表示した例（とても見にくい）

[コラム] Jupyter NotebookをGitで運用する

Jupyter NotebookをGit管理下で運用する方法を2つご紹介します。どちらもPythonファイルをGitで管理するアプローチです。Pythonファイルで扱うことで、差分が見やすくなり、Gitの恩恵を十分に受けることができます。

一つ目は、jupytext [18] です。jupytextは、Jupyter NotebookファイルとPythonファイルの変換を行うライブラリです。VS Codeの拡張機能としても提供されています。PythonコードをJupyter Notebookファイルとして開くことで、ノートブックでの実験のしやすさの恩恵を受けることができます。

二つ目は、VS CodeのPython Interactive Window [20] です。Jupyter Notebookではないですが、Pythonコードをノートブックライクに使うことができます。

Step-2: 開発が始まつたら

既存のライブラリやサービスを上手く使う

機械学習のライブラリやサービスは、便利なものがたくさん用意されています。ソフトウェアエンジニアだけで、機械学習の実装に取り組む場合、これらをうまく活用することを意識してみましょう。一からいろいろなことを学習することも大事ですが、まずは実装して、その結果を確認してみましょう。

簡単に実装するには、ライブラリを活用するのがおすすめです。例えば、scikit-learn [8] には、古典的機械学習全般のアルゴリズムが実装されており、これを使えば機械学習のいろいろな機能を簡単に実装することができます。数学の難しい数式を理解できていなくても、ライブラリがやってくれるのです。

それに加えて、便利なサービスも活用するとより効率的に機械学習の機能を実装することができます。例えば、Google Colaboratory [21] は、Googleアカウントで利用できる、クラウド上のJupyter Notebook環境を提供するサービスです。インストールや環境構築は不要で、簡単にプログラムを実装し、すぐに結果を確認することができます。また、Amazon SageMaker [23] は、AWSが提供するクラウド上の機械学習モデル開発のためのサービスです。Jupyter Notebook環境をはじめ、実験管理機能やAPI構築機能など、さまざまな機能を利用できます。

このように、難しい数式を理解できていなくても、実行するための環境を構築しなくても、便利なライブラリやサービスを活用すると、機械学習プロダクトを簡単・効率的に実装することができます。

[実例] 私たちが利用したライブラリ

今回、私たちのチームで利用した機械学習のアルゴリズムに関しては、ほぼ全てがライブラリによって提供されており、驚くほど少なく簡素な実装で動かしたい機能が実現できました。

具体的には以下のようないライブラリを使っています。

- scikit-learn [8] (古典的機械学習アルゴリズム)
- TensorFlow [9] (ニューラルネットワーク)
- Surprise [11] (レコメンデーション)

以下に機械学習に関連する機能を簡単に実現できた例をいくつか紹介します。

1) カテゴリ変数をダミー変数に変換する

- 分類を表す変数を、重複なしの連続した数値データに変換する

```
le = preprocessing.LabelEncoder()  
  
df['label'] = le.fit_transform(df['ラベル'].tolist())
```

この2行で完了。

2) 教師データとテストデータに分割する

- ・学習対象のデータを、モデル学習のために使う教師データと、学習結果の評価に使うテストデータに分割する
- ・分割する比率は任意に指定する
- ・データが一様に分かれるように分割するが複数回実行しても毎回同じ分割とする

```
train_df, test_df = train_test_split(df, random_state=0, train_size=0.3)
```

この1行で完了。

3) モデルを学習し予測を実行する

- ・ナイーブベイズモデルを生成する
- ・学習データを使ってモデルを学習させる
- ・テストデータを使って学習したモデルから予測を行い評価する

```
model = make_pipeline(CountVectorizer(), MultinomialNB())
model.fit(train_df['data'], train_df['label'])
```

```
train_df['pred'] = model.predict(train_df['data'])
train_score = model.score(train_df['data'], train_df['label'])
```

この4行で完了。

これはほんの一例ですが、言葉で聞くととても難しく感じられる機械学習の機能を、たった数行のコードで実現できてしまうイメージが伝わるのではないかでしょうか。

早めにやること / やらないことを見分ける

本などを読むと、基本的な流れが書いてあるので、まずはその通りにやってみようと思うかもしれません。

しかし、実際に開発を始めてみると「これは早めにやっておくべきだった」とか、逆に「これは慌ててやる必要はなかった」ということが起こります。

ここでは、特に気をつけた方が良い例をいくつか紹介します。

早めにデータ収集の目途をつける

機械学習は、「データ収集が大事」とよく言われています。最初から必要なデータをすべて収集できれば良いですが、実装していく中で、収集するデータ項目が増えることもあります。それは、仕方のないことかもしれませんのが、早い段階で、少なくとも次の2点については考えておくべきです。

1. どの位のデータ量が必要になるのか？

機械学習のアルゴリズムによっては、最低限この位のデータ量（データ件数）がないと意味のある出力ができない、といった目安がありますので、まずはこれを把握しておきます。

2. 必要なデータを収集できそうか？

必要なデータ量の目安がついたら、それを実際に収集できる可能性があるか？についても考えておきます。この時点では正確な収集の計画を立てる必要はありませんが、少なくとも現実的に収集できそうかについては一度考えておきましょう。

この時点でもまったく目途がつかない場合は、下記を検討することも必要です。

- 公開されているデータから入手すること
- 少ないデータでもできること
- 機械学習を用いない方法

早めに評価尺度を考えておく

同じく、構築したモデルの評価尺度の確立も、早めにやるべきことのひとつです。

評価尺度が確立されていなければ、構築したモデルの性能を判断することができないからです。例えば、分類では正解率や再現率、回帰ではRMSE（平均平方二乗誤差）やMAE（平均絶対誤差）が候補になります。さらに、汎化性能のあるモデルを構築するために有用な交差検証において、何分割にするかなども明確にしておきましょう。これを明確にしておかないと、そのモデルを採用するか、をはじめとした議論が成り立たないからです。

パラメータチューニングは後回しでも良い

一方、構築したモデルのパラメータチューニングなどは、後回しにしても良いでしょう。

機械学習モデル開発では、パラメータを、試行錯誤しながら変えていくことになりますが、パラメータがデフォルト値でも、実装はできます。性能改善を目的としたパラメータチューニングは、効果が小さいこともあります。効果が小さいことに、真っ先に取り組む必要はありません。

[実例] 私たちが早めに取り組んでよかったこと / よくなかったこと

書籍レビュー投稿アプリを開発する際に、本命の協調フィルタリングや内容ベースフィルタリングに加えてランダムに推薦する機能も用意することで、早い段階から多くの学習データを収集することができました。

```

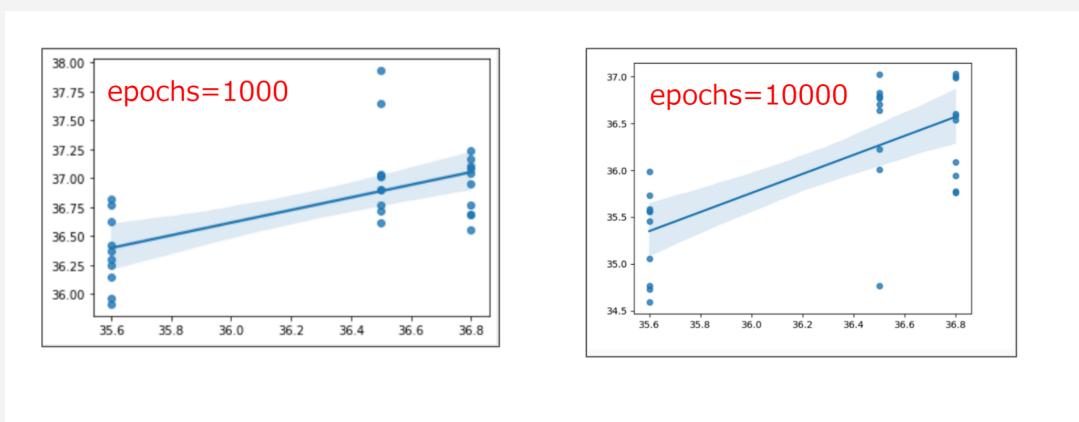
{
  "metadata": {
    "created_at": "2022-07-07T16:50:02+09:00"
  },
  "result": {
    "U02JP20NG02": {
      "random_recommendation": "9784798130088",
      "contents_based_filtering": "9784798130507",
      "collaborative_filtering": "9784295011217"
    },
    "U036L9R65H8": {
      "random_recommendation": "9784894712638",
      "contents_based_filtering": "9784004317227",
      "collaborative_filtering": "9784295011217"
    },
    "U02K1KEB4U9": {
      "random_recommendation": "9780525536239",
      "contents_based_filtering": "9784774185927",
      "collaborative_filtering": "9784894712638"
    }
  }
}

```

ランダムな推薦から
取得されたデータ

ランダムな推薦によるデータの取得

一方で、比較的早い時期にパラメータチューニングをいくつか試してみましたが期待するような効果が得られず、学習データの蓄積が少ない段階ではチューニングによる改善効果は望めないことも分かりました。



パラメータチューニングの例

機械学習システムとアプリを分離する

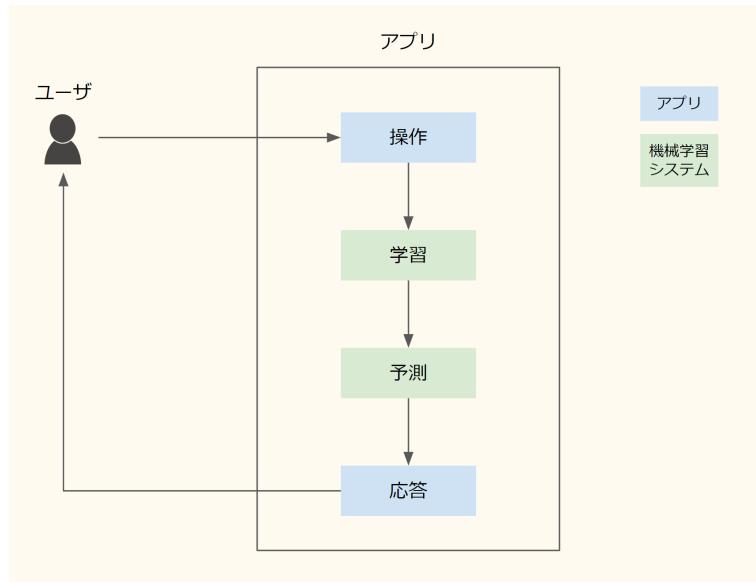
さて、技術検証は済んだでしょうか？

しかし、プロダクト開発でコーディングを始める前に意識しておいてもらいたいことがあります。それは機械学習システムとそれ以外を分離することです。

プロダクト開発を小さく始める場合、機械学習の仕組みをアプリに組み込んでしまいがちです。しかし、そのような構成では開発が進むにつれて機械学習特有の問題に苦戦することになるでしょう。

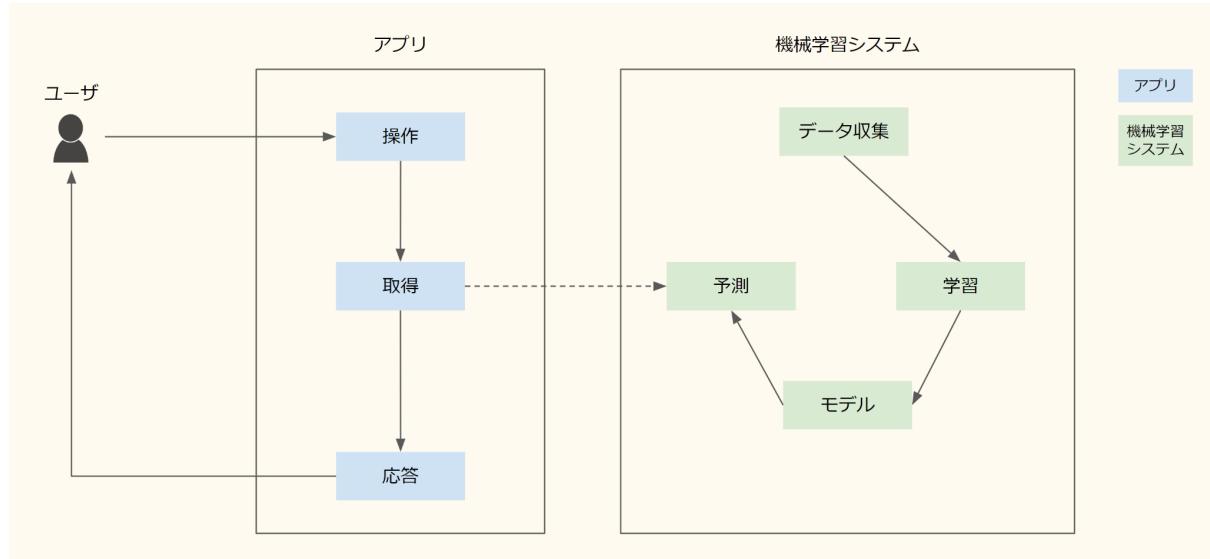
例えば、機械学習の学習や予測といった処理は、それ自体が重く時間がかかることが多く、動作タイミングもユーザ操作と同期する必要のない部分があります。これらをそのままアプリに組み込んでしまうと、ユーザは不必要に重いアプリを使わされる羽目になります。

また、予測では機械学習モデルを参照します。このモデルは学習データの更新などで定期的に更新されるものです。つまり機械学習システムとそれ以外のアプリでは、ライフサイクルが異なる場合が多いのです。この場合、機械学習システムとアプリが密結合になっていると、モデルを更新するたびに、本来は不要であるはずのアプリ全体もデプロイし直す必要があります。



機械学習システムとアプリが一体化した例

このような問題を回避するために、プロダクトの構造を機械学習システムとアプリで分離することをお勧めします。ソフトウェアエンジニアだけの1チームで開発するとしても、プロダクトの構造は明確に分離して疎結合にしておく方が良いです。分離することで、機械学習システムだけを簡単にデプロイできるようになりますし、アプリはあなたの得意な"ソフトウェア開発の領域"として、機械学習システムと切り離して考えることもできるようになります。



機械学習システムとアプリを分離した例

[コラム] 分離の実現方法

分離の実現方法としては、簡易的な方法から順に

- 1) 同一の実行プログラム中のファイルやフォルダ単位で分ける
 - 2) 実行プログラムを分けてWebAPIなどのエンドポイントを介して連携する
 - 3) それぞれの機能をマイクロサービス化する
- などが考えられます。

どれを採用するかは、開発体制によると思われますが、ソフトウェアエンジニア向けには、少なくとも実行プログラムの単位(上記の2)で分けておくことをおすすめします。

これによるメリットの一例として、機械学習システムとアプリ開発で別々のプログラミング言語を選択することが可能になります。機械学習システムのコードは、ライブラリの充実度から、どうしてもPythonで記述することになります。一方でアプリは、ソフトウェア開発の経験で慣れ親しんだJavaやJavaScriptなどの言語で記述した方が、開発がスムーズに進むでしょう。

詳しいアーキテクチャ例を知りたい方は、「AIエンジニアのための機械学習システムデザインパターン」[3] を一読することをおすすめします。

[実例] アプリと機械学習をリポジトリ単位で分離する

書籍レビュー投稿アプリでは、機械学習システムとアプリをそれぞれ別のリポジトリに分離して開発を行いました。これにより扱うコンテナのサイズを小さく保つことができるなど、開発効率の面で多くのメリットがありました。

一方で、リポジトリを分けることで開発中のエディタ画面を切り替えるのが面倒だったり、チケット管理が煩雑になるなどマイナス面もありましたが、得られるメリットはそれ以上に大きいものでした。

bee-ml

環境

BEE で用いる ML のモデル検討 & 生成を行うための環境

環境構築

VS Code Remote Development を利用してください。

環境変数の設定方法

- .env.sample をコピーして .env ファイルを作成してください。
- .env ファイルを編集し、適切な環境変数を設定してください。
- 開発コンテナをビルドすると環境変数がコンテナ内に設定されます。

環境変数に設定する項目

- AWS_ACCESS_KEY_ID: AWS アクセスキー ID
- AWS_SECRET_ACCESS_KEY: AWS シークレットアクセスキー
- AWS_DEFAULT_REGION: AWS リージョン(ap-northeast-1 固定)
- SCRIPT_ENV: スクリプトが実行される環境("LOCAL" or "CI" or "LAMBDA")
- SLACK_BOT_TOKEN: 通知を受け取る Slack アプリの Bot トークン
- SLACK_CHANNEL: 通知を受け取る Slack チャンネル

開発

```
# 依存パッケージのインストール  
make init  
  
# Wikipedia日本語記事による事前学習済みモデルをダウンロード、約1GBあり、時間がかかる  
make fetch-wikipedia  
  
# ローカルでレコード処理を実行する  
make recommend  
  
# 以下のファイルが生成される  
- dist/recommended_book.json
```

機械学習システムのリポジトリ

bee

プロダクトビジョン

永和社員のフィードバックをもらいながらみんなで育っていくプロダクト

サービス名

Bee(Book Erabu Eiwa)

環境構築

VS Code Remote Development を利用してください。

デプロイする場合は、下記の環境変数をローカルマシン（コンテナ外）に設定する必要があります。

- AWS_ACCESS_KEY_ID
- AWS_SECRET_ACCESS_KEY

詳しくは、<https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-envvars.html> を参照してください。

開発

```
# 依存パッケージのインストール  
make init  
  
# デプロイ  
make deploy
```

環境変数

AWS 環境で環境変数を設定する場合は、AWS Secrets Manager を介して設定します。
ステージ環境毎に設定できます。

シークレット ID は、slack_secret_<ステージ環境名>です。 (例: slack_secret_dev)

設定できる環境変数

- SLACK_APP_TOKEN Bolt 入門ガイドを参照
- SLACK_BOT_TOKEN Bolt 入門ガイドを参照
- SLACK_SIGNING_SECRET Bolt 入門ガイドを参照
- NOTIFY_POST_REVIEW_CHANNEL レビュー投稿通知を流す Slack チャンネル ID

性能測定方法

[こちらを参照](#)

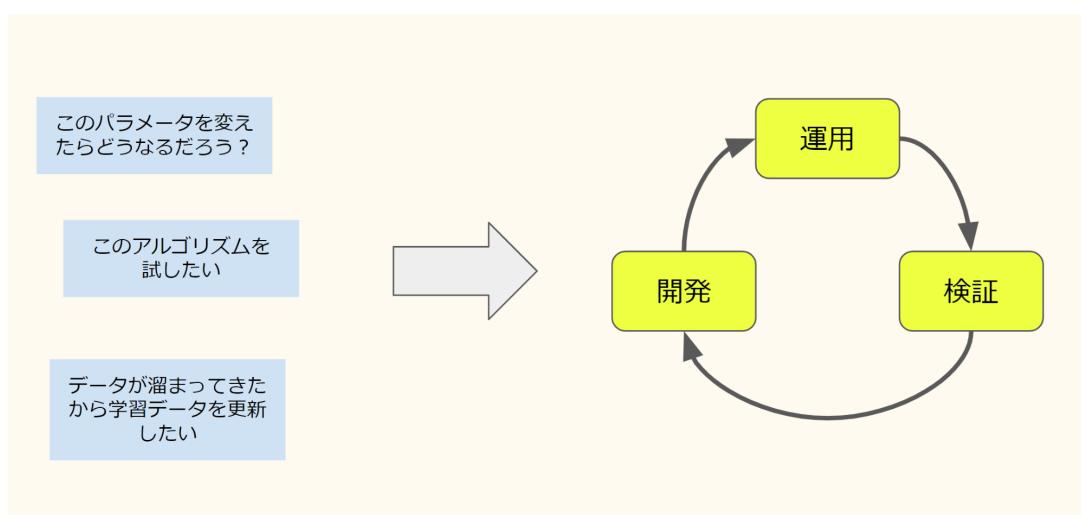
アプリのリポジトリ

機械学習モデル開発サイクル

機械学習プロダクトでは、下記のようなトリガーで、何かを試してみたいという検証的な要素を含む開発サイクルを回すことがあります。

- 自分たちの問題にマッチしそうなアルゴリズムを知った。適用してみたい。
- このパラメータを修正してみたい。
- データが溜まってきたから、学習データを更新したい。

また、機械学習に初めて取り組む場合、ライブラリのインストールがうまくいかなかったり、どのアルゴリズムが適切かわからなかったりと、どうしても失敗が多くなってくるかもしれません。そうなると、上記のサイクルにトライ & エラーの要素が追加されるので、さらに多くの繰り返しが必要になります。



機械学習モデル開発サイクル

以降では、これらのサイクルを効率的に回すためのアプローチを2つ、ご紹介します。
"MLOps"と"スクラム"です。

MLOpsの仕組みを準備する

MLOpsとは、機械学習モデル開発サイクルを持続的に回す取り組みです。ソフトウェアエンジニアには馴染みのあるDevOpsの考え方を機械学習の分野に拡張したものと言えます。

新しいバージョンのソフトウェアを提供するために実行する必要のある一連のステップをパイプラインと呼びます。下記は、MLOpsのパイプライン化の対象となるプロセスの一例です。これらのプロセスのインターフェースの整理、自動化、扱うデータのバージョン管理などを通して、パイプラインを構築します。

- データの収集
- データのバリデーション
- 特徴量の作成
- 機械学習モデルの学習
- 機械学習モデルの評価

- 機械学習モデルのデプロイ

このMLOpsの仕組みを最初に整えておくことを強くおすすめします！

これを最初に整えておかないと、機械学習モデル開発サイクルの素早い繰り返しができません。

例えば、初期の実験コードではありがちなのですが、データバリデーションや特徴量作成などのステップの分割がされていない（整理がされていない状態）と、新しいアルゴリズムを試したくなつたとき、それ用のコードを既存のそれとは全く別に用意することになり、非効率です。

他にも、一連の手順の中で人間が手作業で行う部分が残っている（自動化がされていない状態）と、パラメータ探索のために数百～数千回の実験を繰り返したくなった場合には、もはや実行することが不可能になってしまいます。

ただし、最初から全てを構築する必要はありません。

例えば、扱うデータの構造が壊れていることが多いなら、データバリデーションの仕組みを優先して構築するべきです。求められる精度が高いプロダクトなら、性能評価の仕組みの構築を優先しましょう。

そして、ソフトウェアエンジニアが持つDevOpsの知識は、MLOpsの構築において非常に役に立ちます！

例えば、CI/CDの技術は、MLOpsでも有用です。

MLOpsでは、基本的な構文チェックやテストに加えて、機械学習モデルのトレーニングや性能検証なども必要になってきます。

プロダクトが小さい場合（特にデータが少ない場合）は、これらの実行に、あなたが普段使っているCI/CDツールがそのまま使えます。

プロダクトが大きくなってきた場合（特にデータが増えてきた場合）は、分散処理などのアプローチをとることも必要です。その場合も、パイプライン構築という基礎的な考え方はDevOpsと共通ですから、あなたの経験はきっと役に立つでしょう。

また、DevOpsを支えるコンテナ技術は、MLOpsでも頻繁に用いられます。

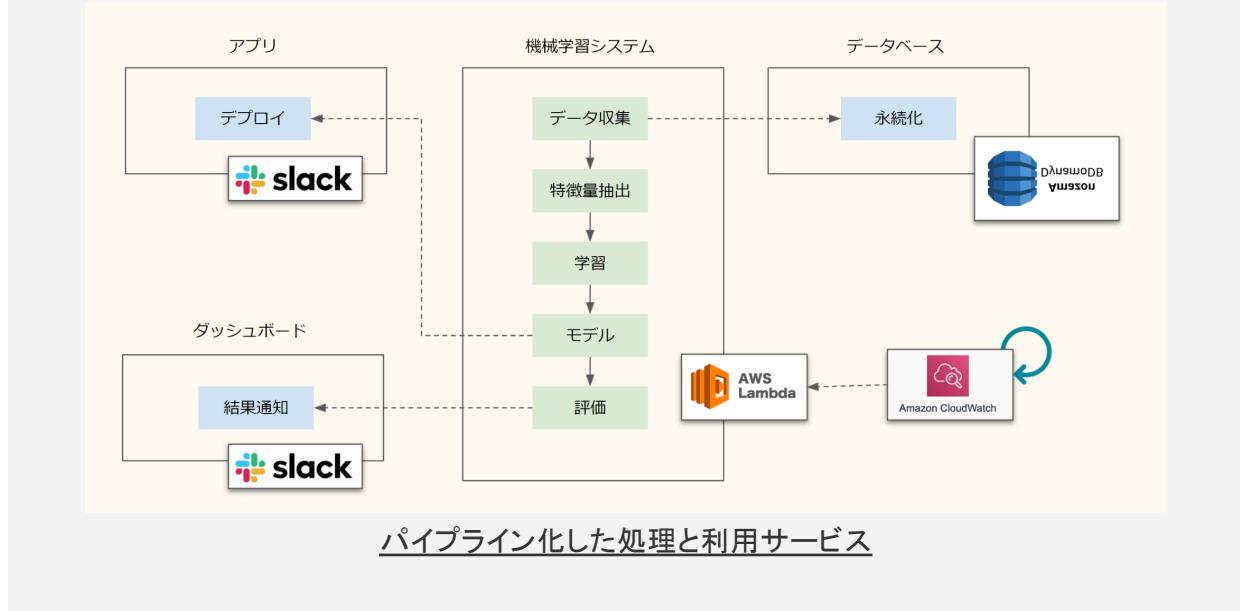
DevOpsと同様に、MLOpsパイプラインのステップ間や機械学習とアプリをつなぐ成果物の単位として、Dockerコンテナイメージを取り扱っている事例は多いです。これらコンテナ技術は、通常のソフトウェア開発でも広く用いられているので、ソフトウェアエンジニアの得意領域と言えます。

MLOpsの仕組み構築には手間と時間がかかりますが、投入したコストをはるかに上回るメリットが得られるはずです。あなたの経験が活かせる領域だと思われる所以、是非積極的に取り組むことをおすすめします！

[実例] 私たちのMLOpsへの取り組み

書籍レビュー投稿アプリでは、データの収集、特徴量の作成、機械学習モデルの学習・評価・デプロイまでを自動化しました。これにより、学習データが増えても、手動の作業なしで機械学習モデルを更新できるようになりました。

これらの仕組みは、ソフトウェア開発で使い慣れているAWS上に構築しました。扱うデータがそこまで巨大でないこともあり、AWS LambdaやAmazon DynamoDBなど、一般的なWebアプリケーション開発でもよく使うサービスを使って、少ない工数で構築することができました。

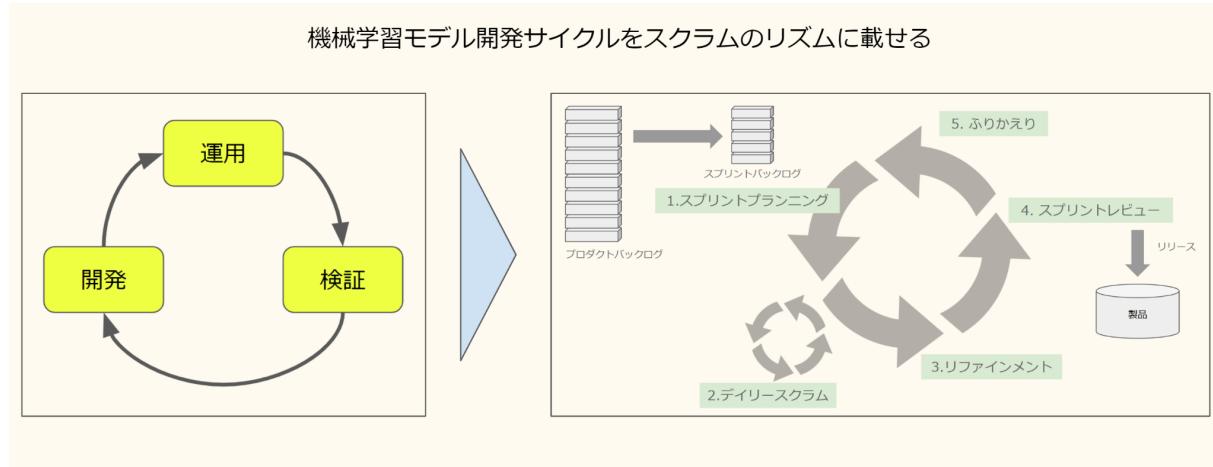


スクラムでリズムを安定させる

機械学習モデル開発サイクルをスクラムで回す

ソフトウェアエンジニアの多くは、スクラムの経験があるでしょう。その経験は、機械学習プロダクトに取り組むときの素晴らしい味方になります！

機械学習モデル開発サイクルも、スクラムのリズムに載せることで、小さいサイクルを保つことができるようになります。アルゴリズムの選定など、機械学習プロダクト特有の作業も、スプリントレビューをはじめとしたスクラムイベントの対象とすることで、小さなトライ＆エラーを行いながら、大きな手戻りなしで進めることができます。



ただし、機械学習×スクラムの取り組みはまだ発展途上であり、その実践例も多くありません。よって、あなたたちの取り組みも、試行錯誤しながらのチャレンジとなるはずです。

スクラムをソフトウェア開発以外で採用する例は、いくつか存在します。例えば、Saab AB社はスクラムを用いて、戦闘機を開発しています [6]。また、iSense社では、もともとソフトウェアチームはスクラムで開発していましたが、その後、営業チームもスクラムを採用するようになりました [7]。このような事例から、一般的なソフトウェア開発以外にも適用できるスクラムのエッセンスを得られれば、機械学習×スクラムの参考になるかもしれません。

適切なスプリントの長さを考える

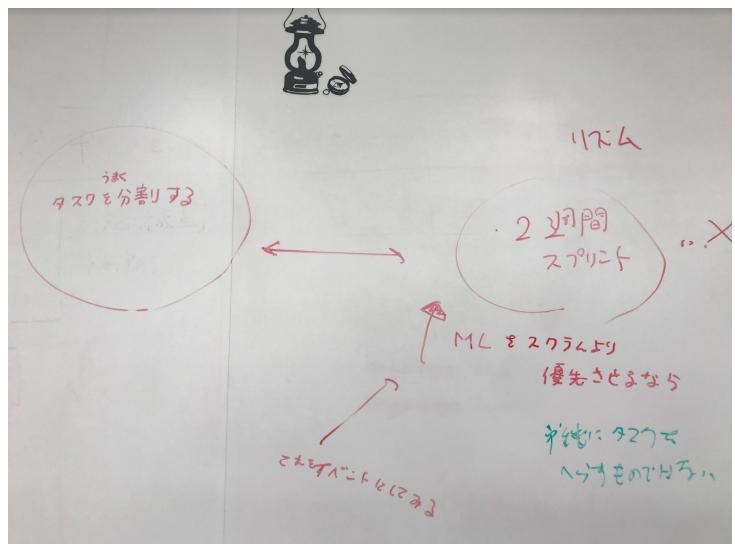
スクラムでは、1スプリントを何週間にするかは決まっていませんが、一般的に1週間にすることが多いようです。1スプリントが短いほど、サイクルを高速で回し、改善のためのフィードバックも多く得られますが、はじめて機械学習に取り組むあなたのチームに、合っているのは本当に1週間でしょうか？一度、じっくり考えてみましょう。

機械学習モデルの開発は、一筋縄ではいきません。便利なライブラリが提供されているとはいえ、普段のソフトウェア開発とは勝手が違う点も多く、実装には想像以上に時間がかかるかもしれません。機械学習の領域では、テストのノウハウあまり確立されておらず、テストコードを書くのも一苦労でしょう。また、作業を細かく分割しようとしても、どのような単位で完了を定義すれば良いかすらも、未知の領域となります。

そういう場合は、スプリント期間中に完了できない作業が頻発することも起こります。そうなると、スクラムのリズムが崩れてしまい、開発プロジェクトの進行にも影響が出るでしょう。

適切なスプリントの長さは、対象となるプロダクトの難易度や、チームの能力によって変わります。例えば、ソフトウェア開発以外の領域では、1スプリント1週間以外を採用するのは珍しいことではありません。前述の戦闘機をスクラムで開発するSaab AB社は、3週間スプリントを採用しています。

ソフトウェア開発の常識に囚われすぎず、あなた達にあったスプリントの長さを考えてみてください。



チームでスprint期間を議論

作業の完了条件は見積もり可能なものにする

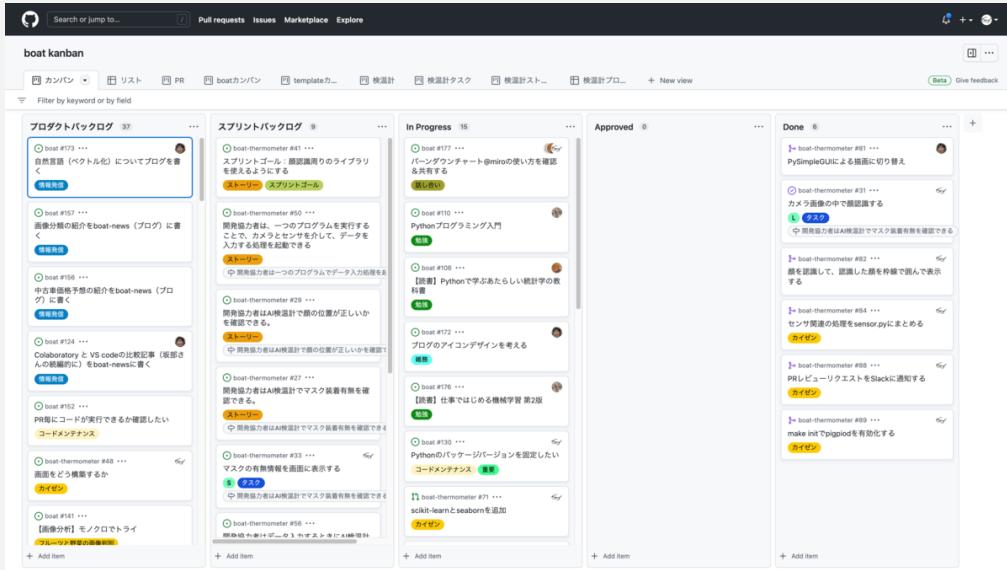
実際にスクラムで開発を始めると、作業の完了条件の設定に悩むかもしれません。

例えば、「モデルの正解率80%以上」を完了条件にしたとします。しかし、機械学習のモデル作成はトライ&エラーの側面が強く、これだけやればこれくらい正解率が向上する、といった予想をするのは非常に難しいです。その場合、この作業を次のスprintまでに達成できるか、という見積もりはほとんど不可能になります。一発で良いハイパーパラメータを見つける、すんなり達成できるかもしれませんし、どれだけ試行錯誤しても達成できないかもしれません。

よって、完了条件には「〇〇モデルが正解率80%以上となるか検証する」のように、自分たちで結果をコントロールできるアクションの形式で設定することをおすすめします。この場合は、検証を実施すれば完了なので、前述のような、いつ完了するかわからない問題は起こりにくくなります。

[実例] 私たちのスクラム

私たちが今回取り組んだプロダクトでは、最初からゴールや開発計画を決定して開始するのではなく、スprint毎にプロダクトオーナーと結果を確認しながら次のゴールを決めていく方法で進めました。その結果、自分たちのその時の実力に見合ったタスクを設定することができました。



私たちのカンバン(GitHub Projects)

1スプリントの長さは、ソフトウェア開発においてスタンダードな1週間で始めました。途中2週間に変更したらどうか？という議論もありましたが、プロジェクト期間が3か月と短く、その中でなるべく多くのサイクルを回したかったことから、そのまま1週間を継続しました。

しかし、不慣れな機械学習の分野では、1週間で動く成果物を作り出すことが難しかったため、スプリント内で完了できないタスクが発生するなど、1週間で行うことのマイナス面もありました。

私たちのようなソフトウェアエンジニアによる取り組みでは、ソフトウェア開発の慣習にとらわれず、少し長めのスプリントを検討しても良いと思います。

[コラム] プロダクトオーナーから見たゴール設定の大切さ(岡島幸男さん寄稿)

私は、このチームのプロダクトオーナーとして、約1年間活動してきました。プロダクトオーナーの重要な役割は、バックログアイテム、つまり要件の優先度を明示することであり、例えばAI検温計なら「精度向上のためデータ収集をさらに行う」と「使い勝手向上のためUIを見直す」どちらを次のスプリントでやりますかと問われた際には、その理由を示しつつ、必ずどちらかを選択する必要があります。

その際に重要なよりどころになったのはスプリントゴールでした。スプリントゴールは、文字通り「このスプリントで達成したい姿、得たい効用」であり、このゴールをチームと会話しながら合意することにより、あいまいになりがちで、性能による具体的なコミットが難しける機械学習プロダクトにおける開発を前に進めることができました。

印象に残っているスプリントゴールをいくつか挙げます。

- ・開発協力者に依頼して、昨年同程度のデータ件数を、開発協力者の負担にならないような時間で収集する
- ・機械学習を試して、結果を返せる(精度は求めない)
- ・自分たちの実力/ペースをつかむ

スプリントゴールは、「どんな機能を実装するか」ではなく、「このスプリントで何を達成するか」であり、そのゴールを達成するのに最適なバックログを選んでいくことを意識してください。

機械学習プロダクト開発に限らず、スクラムを利用する場合は、スプリントゴールや、プロダクトレベルでのゴールであるプロダクトゴールを通じて、チームが何に対してコミットできるのかを、全員で確かめながら進めてください。それによりチームの創発性を高め、より良いプロダクト、より良いチームを育むことができます。

その他、プロダクトオーナーとしての役割や、プロダクトゴールについてのエピソードは、以下のブログ記事も参考にしてみてください。

<https://fdp-blog.hatenablog.com/entry/2022/01/11/005159>

<https://fdp-blog.hatenablog.com/entry/2022/01/11/190744>

Step-3: 次の一歩

機械学習のエキスパートチームと共に働く

ここまでくれば、ソフトウェアエンジニアの手でまずは機械学習プロダクト開発を始めることができます。

しかし、本格的なプロダクト開発のフェーズに進む場合には、機械学習エキスパートの力が必要になります。機械学習の分野は非常に奥が深く、ビジネスの領域で勝負していくためには、ソフトウェアエンジニアだけではいずれ力不足になるからです。

そこで、プロダクトのアプリでは引き続きソフトウェアエンジニアによる確実な開発を進めつつ、機械学習システムではエキスパートによる専任チームを構成することをおすすめします。その場合でも、機械学習モデル開発サイクルはこれまで準備してきたMLOpsの仕組みとスクラムのリズムで進めていくことができます。

更に、ソフトウェア開発チームと機械学習エキスパートチームが協力してプロダクトを開発するためには、スクラムの持つ「チームでコラボレーションする」というパワーが強力な武器になります。そこでは、スクラムに慣れ親しんでいるあなたたちソフトウェアエンジニアが、機械学習エキスパートがスクラムに馴染むためのアシストを行うことができるはずです！

参考文献

- [1] 高橋淳一, 野村嗣, 西村隆宏, 水上ひろき, 林田賢二, 森清貴, 越水直人, 露崎博之, 早川敦士, 牧允皓, 黒柳敬一, “データサイエンティスト養成読本 登竜門編”,
<https://gihyo.jp/book/2017/978-4-7741-8877-5>
「データ分析に必要となる関連知識を広く浅く紹介」
- [2] 有賀 康顕, 中山 心太, 西林 孝, “仕事ではじめる機械学習”,
<https://www.oreilly.co.jp/books/9784873118215/>
「ソフトウェアエンジニアが機械学習をはじめる際の実践的なノウハウが満載」
- [3] 濵井 雄介, “AIエンジニアのための機械学習システムデザインパターン”,
<https://www.seshop.com/product/detail/24571>
「機械学習システムのさまざまなアーキテクチャパターンを知ることができる」
- [4] 守田 憲司, “AIとScrumとスケール”,
<https://www.slideshare.net/KenjiMorita1/aiscrum>
「開発チームと機械学習チームがスクラムで協力するためのコツを紹介」
- [5] Andrew Ng, “AI for Everyone”,
<https://www.coursera.org/learn/ai-for-everyone-ja>
「AIの基礎を学ぶときの最初の教材として全員が見ることをおすすめ」
- [6] Jörgen Furuhjelm, Johan Segertoft, Joe Justice and J.J. Sutherland, “Owning the Sky with Agile”,
https://34slpa7u66f159hfp1fh9aur1-wpengine.netdna-ssl.com/wp-content/uploads/2015/09/Release-version_Owning-the-Sky-with-Agile.pdf
「開発効率向上のために戦闘機の開発(ハード/ソフト)にスクラムを導入した話」
- [7] Rini van Solingen, Jeff Sutherland, Denny de Waard, “Scrum in Sales”,
https://34slpa7u66f159hfp1fh9aur1-wpengine.netdna-ssl.com/wp-content/uploads/2014/05/Scrum_in_Sales.pdf
「不確実性に対応するために営業チームにスクラムを導入した話」

使用したライブラリやツール

- [8] scikit-learn <https://scikit-learn.org/stable/>
[9] TensorFlow <https://www.tensorflow.org/>
[10] XGBoost <https://xgboost.readthedocs.io/en/stable/>
[11] Surprise <http://surpriselib.com/>
[12] OpenCV <https://opencv.org/>
[13] Optuna <https://optuna.org/>
[14] MeCab <https://taku910.github.io/mecab/>
[15] Wikipedia2Vec <https://github.com/wikipedia2vec/wikipedia2vec>
[16] mlflow <https://mlflow.org/>
[17] Jupyter Notebook <https://jupyter.org/>
[18] jupytext <https://github.com/mwouts/jupytext>
[19] Visual Studio Code <https://code.visualstudio.com/>
[20] VS Code Python Interactive window
<https://code.visualstudio.com/docs/python/jupyter-support-py>
[21] Google Colaboratory <https://colab.research.google.com/>
[22] AWS (Lambda, DynamoDB, S3, etc.) <https://aws.amazon.com/jp/>

[23] AWS SageMaker <https://aws.amazon.com/jp/sagemaker/>

[24] Serverless Framework <https://www.serverless.com/>

開発リポジトリ

私たちが開発した「書籍レビュー投稿アプリ」のリポジトリを公開しています。

esminc/boat-bee <https://github.com/esminc/boat-bee>

開発メンバより

岡本卓也(おかもとたくや)

ソフトウェア開発に関わって25年目のアジャイル好きエンジニアです。

このプロジェクトでは、初心に戻って機械学習という新しい領域にチャレンジしてみました。バラエティに富んだチームメンバと一緒に右往左往した1年間は楽しかったです。

坂部宏起(さかべひろき)

普段、Webアプリのアジャイル開発をしているエンジニアです。このプロジェクトでは、大学(情報工学)で学んだ機械学習の基礎知識を生かすことができました。

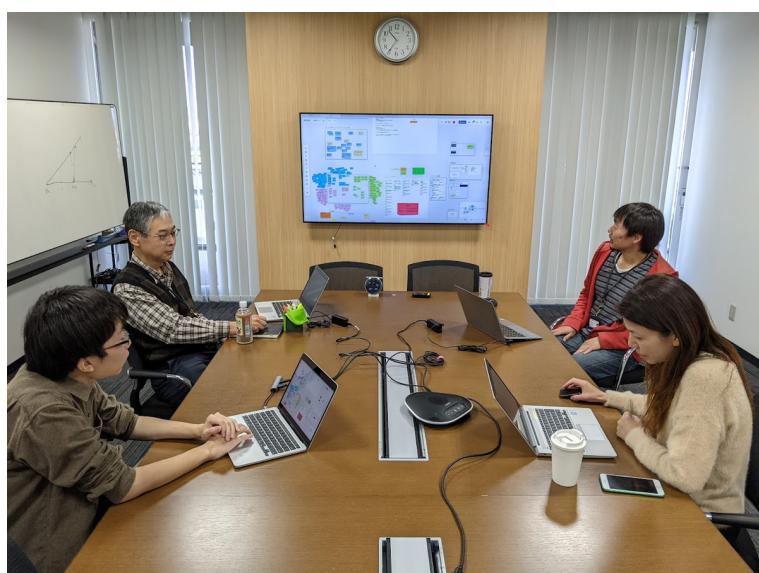
見澤久美子(みさわくみこ)

これまで、ウォーターフォール開発で金融業界のシステム開発をしてきました。

このプロジェクトで、アジャイル開発や機械学習に初めてチャレンジしました。

三田村岳周(みたむらたかし)

金融業務の分野で35年間、レガシーシステムだけを相手にやってきたソフトウェアエンジニアです。このプロジェクトには、大きなチャレンジ(技術転換)として取り組みました。



このチームで1年間頑張りました！