1章 機械学習プロジェクトのはじめ方

本章では、機械学習プロジェクトのはじめ方についてまとめます。

機械学習のプロジェクトは、普通のコンピュータシステムの開発に比べて予測精度を求めるなど試行錯誤をすることが多く、手戻りが発生しやすいため、ポイントを押さえて進めることが重要です。まずは機械学習の概要から、プロジェクトの流れ、機械学習特有の問題、成功させるためのチームづくりについて紹介します。

1.1 機械学習はどのように使われるのか

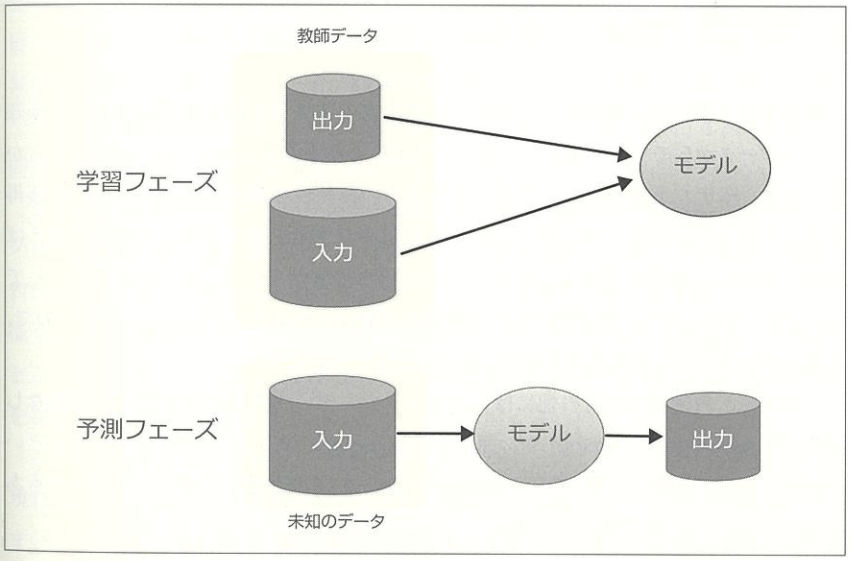


図1-1 機械学習（教師あり学習）の概要

はじめに、機械学習とはどのように使われるものかについて簡単におさらいしておきましょう。

機械学習が良く使われる用途として、未知のデータに対して過去の経験（＝過去のデータ）を元に機械が予測をする、というものがあります。例えばGmailの「迷惑メール」の判定やAmazonの「この商品を買った人はこんな商品も買っています」の推薦など、過去の膨大なデータの傾向を元に、未知のデータに対して予測をします。

このように未知のデータに対して予測を行う場合、ビジネスで頻繁に使われるのが**教師あり学習**（Supervised Learning）というアプローチです。**図1-1**に、教師あり学習の処理の概要をあらわしています。教師あり学習は、大雑把に言うと既知のデータと何かしらのアルゴリズムを用いて、入力データ（画像のRGBや日時、気温などの数値をベクトル化したもの）と出力データ（「犬」「猫」などのカテゴリや降水量などの数値）の関係性を獲得し（これを**モデル**と呼びます）、獲得したモデルによって未知のデータに対する予測ができるようになるプログラムを実現します。

モデルとは入力データと出力データの見えない関係性を、数式やルールなどの簡単な仕組みで近似したものです。学習されたモデルは、どのアルゴリズムを使うかという情報と、データから獲得されたパラメータとで構成されています。

教師あり学習には、既知のデータの入力と出力の関係性を得る**学習フェーズ**、未知の入力データからモデルを通じて予測した出力を得る**予測フェーズ**の2つのフェーズがあります。モデルを獲得する、すなわち入出力の関係性を表現するパラメータを獲得することで、未知のデータに対しても一定の基準で予測を行えます。人間は、例えば徹夜で連続して単純作業をしていると、その判断基準がぶれてくることが往々にしてあります。それに対し、機械学習の予測基準は学習フェーズからブレることはないので、大量のデータに対しても人より安定した予測を行えるのです。

ちなみに、教師あり学習の他には入力データからデータの構造を獲得する**教師なし学習**（Unsupervised Learning）や、囲碁や将棋などでどのように行動をしていくかという戦略を獲得する**強化学習**（Reinforcement Learning）など他の学習のアプローチもあります（詳しくは2章で紹介します）。

1.2 機械学習プロジェクトの流れ

実際の機械学習を含めたプロジェクトを開始する際には、以下のような流れで進めていきます。

1. 問題を定式化する
2. 機械学習をしないで良い方法を考える
3. システム設計を考える
4. アルゴリズムを選定する
5. 特徴量、教師データとログの設計をする
6. 前処理をする
7. 学習・パラメータチューニング
8. システムに組み込む

この流れをおおまかに言うと、「解きたい課題を機械学習で解ける問題設定に落としこむ」（1、2）、「解くための道具選びと前処理」（3～6）、「モデルの作成」（7）、「サービスへの組込み」（8）という4ステップになっています。特に最初の2つの課題設定と前処理が重要です。いくらデータ量があっても、適切に前処理がなされていなければ性能は出ませんし、そもそも解こうと思っている課題が人間にも解けない問題の場合は機械学習で解くのも難しいでしょう。機械学習、特に「教師あり学習」では、何が「正解」なのかを人間が機械に教えなければなりません。つまり、人が「正解」を決められない問題は機械にも解くことができないのです。

まずは、「機械学習でこういう課題を解決した」という事例を見た時に、「どういったアルゴリズムで解決したか」「どのようなデータを特徴量として使っているか」「機械学習部分をどのように組み込んでいるか」ということを意識して調べると良いでしょう（特徴量という単語については後ほど詳しく説明しますが、ここでは入力情報という意味でとらえてください）。こうして引き出しを増やすことで、何ができて何ができなさそうかを判断できるようになります。

こうした引き出しがないと、「機械学習でなにかすごいことをしたい」という上司が現れて、何をすればいいかというところから頭をひねるようなプロジェクトになってしまい、うまくいかない結果に終わる危険性が高まります。大事なことは、機械学習で解ける範囲と解けない範囲を線引きできること、そしてそれを実現するために（泥臭いこ

とも含めて）手を動かせることです。「データ分析業務は前処理が8割」と言われることもあるくらい、フォーマットが崩れているCSVのパースや、Webのログから必要な情報を抽出するなど、分析可能な状態にするまでの時間が大きな割合を占めます。

機械学習をシステムの一部として含むシステム開発は、実際には試行錯誤を繰り返す作業になります。特に、上記の4から7は試してみては変更して、という作業を何度も繰り返すことになると思います。機械学習の理論的な研究では4と7を重点的に取り扱う一方で、ビジネスでは1から8まで全てを行います。そのため、試行錯誤を効率よく行うことが重要です。

では、それぞれの手順について順番に説明をしていきます。

1.2.1 問題を定式化する

次に一般的な課題を解くのと同じように、どのように問題を解くのか定式化をします。その時重要なのが、何を目的にするのか、そして解きたい課題について仮説を立て、何をすればよいのかを明確にすることです。そもそも、何かシステムを作るときには、「売上を改善する」「有料会員を増やす」「製品の生産コストを減らす」など、ビジネス上の目的があるかと思います。こうした大きな目的に対して、例えば「生産コストを減らす」ためには「歩留まり改善」をしなければならず、そのためには「どこで不良が起こっているかを特定する、そのために機械学習の力を使う」というように、より具体的でアクション可能なレベルまでブレイクダウンしていきましょう。

ブレイクダウンをする課程で、売上目標や日毎の有料会員増加数など何かしらビジネス上の指標、いわゆるKPI (Key Performance Indicator) が決まってくるはずです。これは、予測モデルの性能とは別の軸で重要な指標です。KPI自身はフェーズによって変わりうるものではありますが、最初のブレイクダウンをしたときに仮でも良いので1つ決めると良いでしょう。なお、KPIの決め方や目的と課題の考え方や整理の仕方は『Running Lean』[runninglean]や『Lean Analytics』[leananalytics]などの書籍を読むと良いでしょう。

機械学習の問題設定としては、例えば、「ECサイトの売上を上げるために、ユーザー毎におすすめ商品を提示する」とか、「工場の電力消費量を最適化するために、消費電力を予測する」などというように、プロジェクトの目的と解き方をセットで考えます。良くない機械学習の問題設定の例としては、「有料会員を増やしたい」といったアクションがすぐ起こせない曖昧なものや、「深層学習で凄いことをする」といった目的も分からないようなものです。もしかすると、トップダウンでこういった要求が降ってくるかもしれませんが、現場で手を動かす人はもっとブレイクダウンをしなければいけません。

仮説の立て方と検証方法については、クックパッド開発者ブログに掲載されている「仮説検証とサンプルサイズの基礎」[†1]という記事が役に立つでしょう。

1.2.2 機械学習をしなくて良い方法を考える

次に、機械学習を本当に使うべきなのかを考えます。機械学習プロジェクトのはじめ方なのに、なぜ？と思うかもしれませんが、「機械学習は技術的負債の高利貸しのクレジットカード」というタイトルの論文[dsculley]があるほど、機械学習を含んだシステムは通常のシステム以上に技術的負債[†2]が蓄積しやすいのです。

機械学習を用いるシステム構築の難しさには以下のようなものがあります。

- 確率的な処理があるため自動テストがしにくい
- 長期運用しているとトレンドの変化などで入力の傾向が変化する
- 処理のパイプラインが複雑になる
- データの依存関係が複雑になる
- 実験コードやパラメータが残りやすい
- 開発と本番の言語/フレームワークがバラバラになりやすい

これらの難しさの対処方法は、後ほど「1.3 実システムにおける機械学習の問題点への対処方法」で書きたいと思いますが、特に「入力データの傾向が変化する」という点は大きな問題です。例えば、テキストを扱う問題として文のポジネガ判定を行う際に、文章に含まれる「ヤバい」という単語について、昔の文章ではネガティブな意味合いしかなかったのに対し、比較的新しい文章ではポジティブな意味もありうる、といった具合に、テキストの意味が変化する場合があります。これ以外にも、「スイカ」といえば果物のスイカの意味だったのが、電子マネーを指すようになったりと、用法のトレンドが変化したり、新語が登場したりします。

---

†1 http://techlife.cookpad.com/entry/2016/09/26/111601

†2 技術的負債とは、ドキュメントやテストコードがない、行き当たりばったりの設計が残っている、コンパイラの警告が残っているなど、リリースを優先して問題を先送りにすることです。詳細はhttps://ja.wikipedia.org/wiki/%E6%8A%80%E8%A1%93%E7%9A%84%E8%B2%A0%E5%82%B5を参照ください。

このように、同じ予測モデルを使い続けていると、入力データの傾向やトレンドが変化してしまった場合、予測精度が下がったり意図しない挙動をする可能性があります。これを防ぐためには、定期的に新しいデータで予測モデルを更新したり、必要に応じて特徴量の再検討をする必要があります。つまり、「予測モデルをメンテナンス」し続ける必要があります。

また、機械学習アルゴリズム内には乱数を用いた確率的な処理が含まれていることも多く、ルールベースで処理をした時のように挙動が決定論的ではありません。さらには、あらゆるデータに対する挙動を予め確認することは不可能です。それができないような大量の自動処理を期待しているので機械学習を利用しているのだと思います。そのため、思いもよらない予測結果が出力されるリスクが常に存在するという認識が必要です。以前、Googleフォトが写真に写ったアフリカ系の人をゴリラと認識してしまい、差別的であると問題になったことがありました[†3]。この時には「ゴリラ」というタグが出力されないような後処理を追加したようですが、このように思わぬ出力がされる可能性があります。意図しない予測結果が出てしまったときに、後から介入できる仕組み（例えば特定のラベルはブラックリストに登録してはじく）を用意しておくことが重要です。

では、どのようなビジネス上の課題に対して機械学習を利用すれば良いのでしょうか？ 私は以下の条件を満たす必要があると考えます。

- 大量のデータに対して、高速に安定して判断を求める必要がある
- 予測結果には、一定数の間違いが含まれることが許容できる

機械学習による予測は、人間が疲れによって判断がぶれてしまうのと異なり、大量のデータに対しても同じ基準で常に判断し続けることができます。また、予測結果が100%ということはまずありえません。そのため運用上、誤りをカバーできる仕組みがなければなりません。

条件が整った上で、まずMVP (Minimum Viable Product) を作りましょう。MVPとは、『リーンスタートアップ』[leanstartup]の文脈で良く話題にされるもので、最低限の顧客価値を生み出す最小のプロダクトのことです。では、機械学習でのMVPとは何でしょうか。例えば、男女や年齢などのユーザーの属性によるクロス集計でセグメント分けをして、そのセグメント毎にルールベースでレコメンドをしてはどうか？ Apache

†3 https://www.theguardian.com/technology/2015/jul/01/google-sorry-racist-auto-tag-photo-app

SolrやElasticsearchなど、既存のモジュールにあるMore Like Thisなどの機能の組み合わせではダメなのか？といったように、集計ベースや既存のモジュールの機能で簡単に実装できるものがMVPとなりえます。もちろん、MVPの原則に則って人手で決め打ちのコンテンツを用意をして、簡単なルールで振り分けるのでも十分でしょう。筆者も経験がありますが、場合にもよりますがこのMVPでも十分機能するということが往々にして起きるのです。

MVPを検証することで、そもそも自分が立てた仮説の筋が良いか悪いかを判断できます。こういった機械学習の問題設定から仮説検証までのサイクルは、多くの場合で一般的なコンピュータシステムのプロダクトよりも長くなるようです。そもそも狙い自体が間違っている場合には、システム構築と実験まで終えた後で、最初の問題設定まで大きく手戻りすることさえあります。本当に顧客が必要だったものは何か、コンセプトは正しいのかを事前に検証することが、通常のプロダクトより重要になってきます。

プロジェクトのはじめに機械学習をやろう、と始まったプロジェクトでも、必要がないのであれば機械学習を使わない方向にかじを切ることを恐れないでください。

このように、機械学習に向いている課題であることを確認し、MVPを作って最低限のコンセプトを検証することで、システムの設計へと進みます。

1.2.3 システム設計を考える

問題の定式化とMVPの検証が済んだら、機械学習を含めたシステムを設計しましょう。設計の上で重要なポイントは2つあります。

- 予測結果をどういう形で利用するのか
- 予測誤りをどこで吸収するのか

1つ目は、例えばバッチで予測処理をしてその結果をRDB (Relational Database) で配布するのか、Webサービスやアプリケーションを用いてユーザーのアクション毎に非同期で予測するのかなど、方法の違いがあります。予測結果をどう渡すかについて、詳しくは「4章 システムに機械学習を組み込む」を参照してください。

機械学習に100%正解を出力するアルゴリズムは存在しません。システム全体でどのように誤りをカバーするのか、人手による確認や修正は必要なのか、必要だとしたらどこでカバーするのかを考えることも機械学習のシステム設計に含まれます。それを理解した上で、どのようにシステム全体でリスクをコントロールするかが重要です。例えば

予測結果を人手で確認するフェーズを用意したり、予測結果が重大な悪影響を与えないと分かっていたりする場所でアプリケーションに利用するなどの方法を用います。

また、このフェーズを過ぎると実際に手を動かしてデータ収集や予測モデルの作成、システム構築を進めるフェーズに入るので、ここまでに目標性能と撤退ラインを決めましょう。機械学習の予測モデル開発は、往々にして「あと少し性能が良くなったら…」と、予測モデル自身を改善する泥沼にハマりこみます。そこまでに、学習データの収集や正解データの作成などといった作業をしてしまうと、一定のドメイン知識が付いてくるため、根拠のない自信で改善ができると思いこんでしまいます。場合によっては、問題設定の見直しまで戻って繰り返し改善し続ける可能性もあります。サンクコストによるバイアスがかかる前に、「2ヶ月で90%の予測性能を達成する」といった、具体的な目標性能と撤退ラインを決めておきましょう。なお、予測性能の決め方については「3章 学習結果を評価しよう」で解説します。

1.2.4 アルゴリズムを選定する

機械学習を使うときに考えなければならない、採用するアルゴリズムについて考えます。

アルゴリズムの選定の指針は「2章 機械学習で何ができる？」も参考にしてください。

過去に類似の問題がどのように解かれているかを調べると、おおよそのあたりは付けられます。データの特性が分からなければ、クラスタリングなどの教師なし学習（2章で説明します）や散布図行列（図1-2）などを使って事前に可視化してあたりをつけ、どういった方法で解けるのかを考えます。

また、想定されるデータ量を含めてオンライン学習が良いのかバッチ学習（「4.2.1 混乱しやすい「バッチ処理」と「バッチ学習」」で説明します）でも十分なのかを見積もります。

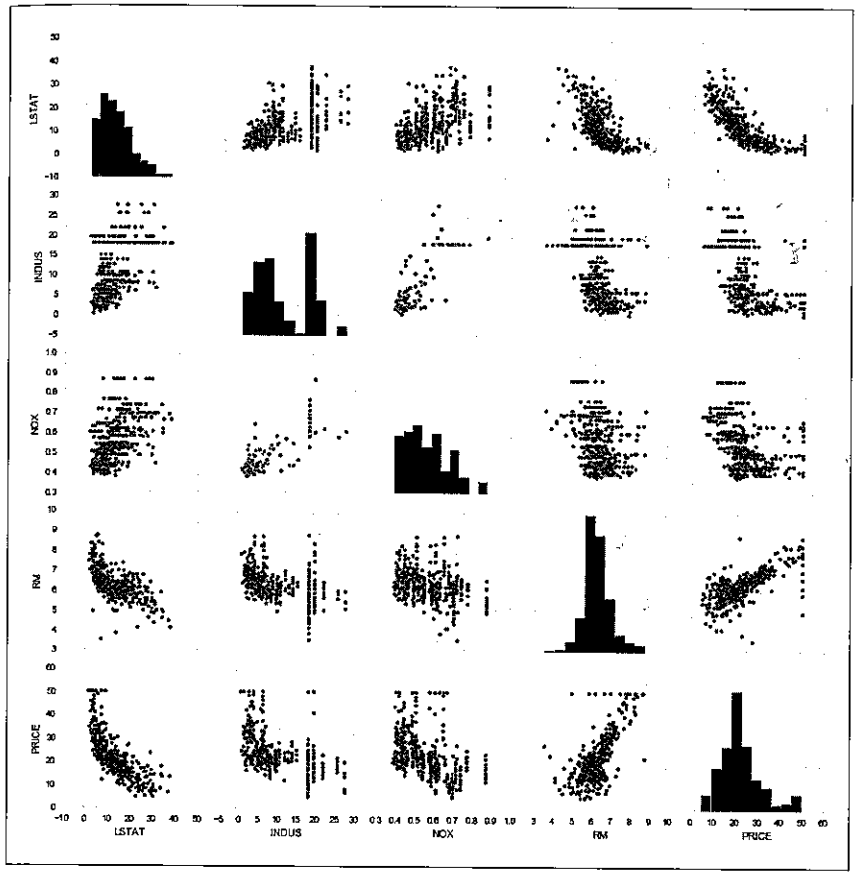


図1-2 散布図行列の例

1.2.5 特徴量、教師データとログの設計をする

アルゴリズムの候補を見繕ったら、どういった情報が使えるかについて設計をします。

特徴量（Feature）[†4]とは、機械学習の予測モデルに入力をする情報のことを指します。機械学習では、入力の情報をまず数値ベクトルにします。例えば、明日の積雪の有無を予測するために、今日の気温（1.0℃）、降水量（0.8mm）、風速（0m）、積雪量（2cm）、

†4 「feature」という単語の訳について、自然言語処理の分野では「素性（そせい）」という表現が好まれることが多いですが、ここでは特徴量と訳します。

天気（曇り）を使うとすると、それぞれを数値化したもののリスト（例：[1.0, 0.8, 0.0, 2.0, 1]）が特徴ベクトルになります。

ここで、「曇り」のような特徴量を**カテゴリカル変数**（Categorical Variable）と言い、晴れは0、曇りは1というような数値データに変換して処理をします。このような数値データのことを**ダミー変数**（Dummy Variable）と呼びます。scikit-learnではLabelEncoderクラスやOneHotEncoderクラスでカテゴリカル変数をダミー変数に変換できます。

古典的な機械学習では特徴量が肝となります[†5]。ここはビジネスドメインの知識がある人と、ユーザーの行動ログや購買履歴、工場のセンサーデータなど、機械学習の入力となる特徴量が、予測に必要な情報を含んでいることを予め確認しておきましょう。例えば、タービンの不具合を検出するために、過去の経験を元にハンマーで叩いた音の情報を集めて不良検出をしたという事例もあります。ビジネスのドメイン知識を持った人と協力をして、何がその現象に影響を与えそうかということを確認します。後から不要なデータを削ることはできますが、必要なデータを遡って取得することはできません。

特徴量をいったん決めたら、入力データとなる正解データを用意します。**教師データ**とは、**教師あり学習**と呼ばれる複数のカテゴリを予測する問題を解く際に必要な、正解カテゴリのラベル（正解ラベル）と元となるデータのセットことです。例えば画像を使った物体認識では、写真に移っている「車」や「犬」といったカテゴリの正解を、あらかじめ人手でつけておく必要があります。なお、こうした分類対象のカテゴリのことを機械学習では**クラス**（class）と呼びます。プログラムのクラスとは違うので注意してください。ビジネスの多くでは、教師あり学習を使い何かしらを分類することが多いです。

教師あり学習では、質の良い正解ラベルをどのように取得するかが重要になります。この正解ラベルのクオリティ次第で問題がうまく解決できるかが大きく変わります。教師データの正解ラベル収集については5章を参考にしてください。

また、教師データのもととなるデータは、Webアプリケーションのログなどから取得することも良く行われます。特徴量を抽出するためのログの設計については、「4.3 ログ設計」で詳しく説明します。ログを設計するタイミングに合わせて、思いつく特徴量も列挙すると良いでしょう。というのも、ログの収集はいったん開始してしまうと形式を変えるコストが大きく、変えられたとしても使えないデータが増えてしまうからです。

†5 深層学習（Deep Learning）では、例えば画像の物体認識ではRGBの値を使うなど、特徴量の設計よりもどういったネットワーク構造を使うかといったことのほうが重要です。

1.2.6 前処理をする

前処理はタスクに依存するためここでは詳しくは書きませんが、不要な情報を削ぎ落とすなどデータを機械学習に使える形にする重要なプロセスです。機械学習の入力とするデータは、特徴量のところでも説明をしましたが、RDBで表現できるような表形式のデータの形になっています。しかし、実際のWebのログなどの生データはテキスト形式だったり、すぐにそのまま使えるデータではありません。数値データも、センサーデータが一部取得できていないことによる欠損値と呼ばれるデータの処理や、異常な値を除外したり、値の取りうる幅の影響を受けないように正規化したりといった作業が必要です。テキストデータの場合には、単語に分割して頻度を数えたり、低頻度語を除去したりという調整を行います。先ほど紹介したカテゴリ変数をダミー変数にする操作もここに含まれます。こうしたデータ変換が、前処理の最初に重要なステップとなります。

いくら優れたアルゴリズムを使っても、データを適切に整形・加工することに勝るものはありません。現実の問題では多くの時間をここに取られます。

1.2.7 学習・パラメータチューニング

いよいよ学習を行います。学習のアルゴリズムが決まっているので、機械学習のアルゴリズムを調整するパラメータを試行錯誤しながら変えてみて、より良い結果ができるパラメータを探索します。まず、人力で付与した正解やルールベースで決めた正解など、ベースラインの予測性能を決めてそれを超えることを目指します。

最初のステップとしては、ロジスティック回帰など比較的シンプルなアルゴリズムと既存のライブラリ・フレームワークを用いることで、シンプルな予測モデルを作ることをゴールとしましょう。多くの場合、一部データが正常に取れていないなどデータ自体にバグが潜んでいます。そのため問題を切り分けるために、まずはシンプルな手法で良いのでいったん予測モデルを作るようにします。

はじめての予測モデルでいきなり予測性能99.9%のように高い性能が出た場合には、何かミスがあると疑ってかかりましょう（筆者は、感覚的には初めて書いたコードのテストが全て通ってしまったときに近いものを覚えます）。多くの場合、学習時のデータに対して過剰に適合してしまい、未知のデータを適切に予測できない**過学習**

(Overfitting) や、本来知り得るはずのない正解データの情報が教師データに紛れてモデルの予測性能が不当に高くなってしまうData Leakageが起きています。

過学習、Data Leakage

過学習したモデルは、平たく言うと「学習に使ったデータに対してはきちんと正解できるけど、知らないデータに対しては全然当たらない」というモデルになります。これは、既知のデータに対して過剰に最適化をしてしまい、未知のデータには対応できないという状態になります。昔、私がセンター試験を受けた年に、英語の突如出題傾向が変わりました。塾でバッチリ対策をしていた人が「うわー、今年傾向変わって全然解けなかったー。きっと他の人も解けなかったよね」という話をしていたのですが、今思うと、これもある意味過学習かもしれません。逆に、未知のデータにも対応できるモデルを**汎化** (Generalization) 性能があるモデルといいます。

Data Leakageの分かりやすい例として、Kaggleの癌予測のためのコンテストのデータに前立腺手術をしたかどうかというフラグが含まれていたことがありました。[†6]この情報を使った予測モデルは非常に高い予測性能を達成しましたが、前立腺がんの人がガンとわかった後に手術を受けているというだけの情報で、未知のデータに対しては意味のない予測モデルとなってしまったのです。他にも、よくある例として、時系列データの予測を行う際に学習に使うデータと検証に使うデータをランダムで分割してしまったために、未来のデータに対する予測をするモデルを作りたかったのに、未来のデータも含めて学習してしまったという話があります。

これらのポイントを意識しながら、学習とパラメータチューニングを行います。性能改善を行う場合は、誤判定をした予測結果を実際に見て、何が誤りの原因になっているか、共通項はないかとエラー分析をすることも忘れないようにしましょう。ここでうまく行かなかった場合には4に戻って、アルゴリズムの検討からやりなおします。

†6 https://www.kaggle.com/wiki/Leakage

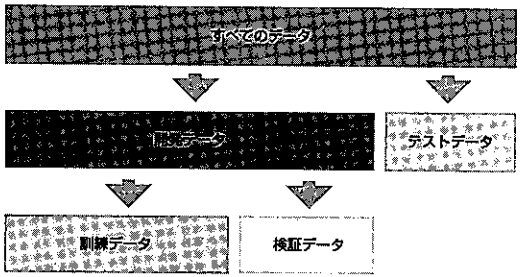
過学習を防ぐには

過学習を防ぐにはいくつかの方法があります。アルゴリズムによらない方法としては以下のような方法があります。

1. **交差検証** (Cross Validation) を使ってパラメータチューニングをする
2. **正則化** (Regularization) を行う
3. **学習曲線** (Learning Curve) を見る

交差検証とは、学習の際に学習用の**訓練データ** (Training Data) と評価用の**検証データ** (Validation Data) を分割して性能を計測することで、特定のデータによらない汎化性能のあるモデルを得る方法です。例えば、データを10分割して9割の訓練データで学習、残り1割の検証データ評価を行う、というプロセスを10回繰り返すことで、平均的に性能の良い**ハイパーパラメータ** (Hyper-parameter、ニューラルネットワークの隠れ層の数やロジスティック回帰のしきい値など、モデルの性能を左右するパラメータ) を選びます (scikit-learnには`cross_val_score()`という関数や`GridSearchCV`というクラスなど交差検証を簡単に行う便利な道具があります)。

実際には、最初に例えば1割のデータを抜き取って最後の性能評価にのみ使うことで、ハイパーパラメータのチューニングとは独立した性能評価をできるようにします。この抜き取ったデータを**テストデータ** (Test Data) と呼びます。なお、本書では訓練データと検証データを合わせたデータを**開発データ** (Development Data) と呼びます。



過剰適応を防いで汎化性能を上げるために正則化というテクニックを用います。簡単に言うと、2つのクラスを分離する境界を、全データ正しく分かれるようにきっちり設定するのではなく、多少違うクラスのデータが混ざって誤ってもいいから未知のデータに対する対応力を付けるというものです。詳細は「2.2.2 ロジスティック回帰」で説明します。

学習曲線とは、データサイズや学習の繰り返し回数に対して、訓練データと検証データでの損失（あるいは精度）の推移をグラフに書いたものです。損失についての詳細は2章を、学習曲線の詳細は筆者の記事「そのモデル、過学習しているの？未学習なの？と困ったら」[†7]を参照ください。

1.2.8 システムに組み込む

おめでとうございます、良い性能の予測モデルを得ることができましたね。それでは機械学習のロジックをシステムに組込みましょう。

この時気をつけたいのが、予測性能とそれに伴うビジネスインパクト（例えば商品購入のコンバージョン率など）をモニタリングすることです。システムに組み込むことは、ビジネス的には仮説検証のための1つのステップにすぎません。予測性能のモニタリングには、あらかじめ人手で用意をしたデータと正解ラベルのセットを使って予測性能を計測します。

このようなデータのセットをゴールドスタンダード（Gold Standard）と呼びます。

予測モデルの開発に注力していると往々にして忘れがちなのがですが、本来は予測モデルを作ることで、売上目標や日毎の有料会員増加数など何かしらビジネス上の指標、いわゆるKPIの改善をしたいというような要求があるはずです。そちらの推移を見守り、必要に応じて性能改善を続けます。

機械学習は「1.2.2 機械学習をしなくて良い方法を考える」でも書いたように、長期

†7 http://chezou.hatenablog.com/entry/2016/05/29/215739

運用をしていると入力の傾向が変わります。これにより、予測性能がじわじわと、あるいは急激に劣化する場合がよくあります。KPIが悪化した場合は、5～7に立ち戻って改善をする必要があります。これに立ち向かうためにも「システムに組み込んで終わり」ではなく、継続してビジネスに貢献しているのかを確認し、改善し続けましょう。

また、改善し続けることができる持続的な組織づくりも重要となります。KPIをきちんとトラックできるようにダッシュボードを作ったり、異常時にはアラートを飛ばすようにしたり、アクションをいつでも取れるような体制が重要です。

1.3 実システムにおける機械学習の問題点への対処方法

「1.2.2 機械学習をしなくて良い方法を考える」でも書きましたが、実システムにおける機械学習の問題点には以下のようなものがあります。

1. 確率的な処理があるため自動テストがしにくい
2. 長期運用しているとトレンドの変化などで入力の傾向が変化する
3. 処理のパイプラインが複雑になる
4. データの依存関係が複雑になる
5. 実験コードやパラメータが残りやすい
6. 開発と本番の言語/フレームワークがバラバラになりやすい

まとめると、予測性能のみを追い求めてモデルの更新が難しくなり易く、システムの複雑性が上がってメンテナンス性が低下しがちであり、変化に追従しづらいと言えます。

こうした問題に対処するためには、変化を前提とした設計をするために、以下のポイントが重要となります（カッコ内の数字は対応している課題です）。

- 人手でゴールドスタンダードを用意して、予測性能のモニタリングをする（1、2、4）
- 予測モデルをモジュール化をしてアルゴリズムのA/Bテストができるようにする（2）
- モデルのバージョン管理をして、いつでも切り戻し可能にする（4、5）
- データ処理のパイプラインごと保存する（3、5）
- 開発/本番環境の言語/フレームワークは揃える（6）

以下では5つのポイントについて順番に説明をしていきます。

1.3.1 人手でゴールドスタンダードを用意して、予測性能のモニタリングをする

ゴールドスタンダードを用意することについては、「1.2.8 システムに組み込む」に書きましたのでそちらを参照してください。機械学習の予測結果は確率的な処理を含むため、個別の予測結果を決定論的に自動テストで検証するのは難しいです。ですので、予め用意しておいたデータと正解に対して予測性能を測定し、その推移を監視します。そうすることで、1の自動テストがしづらい問題を、予測性能のモニタリングでカバーします。また、2の入力の傾向の変化についても、予測性能をダッシュボードでモニタリングし、閾値を設けてアラートを飛ばすことで、長期運用時に傾向の変化に気づきやすくなります。4については、例えば予測モデルの更新と単語分割用の処理の辞書の更新とを同時に行ったときに、本番環境で予測モデルだけを更新してしまい辞書の更新を忘れてしまうと言ったデータの不整合が起こりえます。こうした問題も、予測性能のモニタリングによりカバーできます。

1.3.2 予測モデルをモジュール化をしてアルゴリズムのA/Bテストができるようにする

予測モデルのモジュール化について説明をします。性能向上を継続するためには、1つのアルゴリズムだけでは天井が見えてしまうことがあります。そのようなときのために、複数の予測モデルを並列で並べてA/Bテストできるようにモジュール化して交換が容易な設計をしておくことが重要です。モジュール化によってモデルの比較をしやすいシステムにしておくことで、特徴量を変更したりアルゴリズムを変更したモデルを並行させて性能検証ができるのです。そうすることで、2の長期運用した時の入力の傾向の変化に対しても、現行の予測モデルを動かしながら新しい取り組みが容易になるのです。

1.3.3 モデルのバージョン管理をして、いつでも切り戻し可能にする

モデルのバージョン管理については、いつ、何の影響によって本番環境の予測モデルが性能劣化を起こすか分かりません。入力データの形式が変化したかもしれませんし、途中の処理が変わったのかもしれません。モデルの更新が原因かどうかを切り分けるためにも、過去のモデルに切り戻せるようにしておくことが大切です。当然、ソースコードはバージョン管理をすると思いますが、可能であれば過去のモデルに切り戻ししたときにきちんと比較できるよう、データもバージョニングをしておくのが望ましいでしょう。ソースコード、モデル、データの3つの変更が管理されていること理想的な状態です。

モデルのバージョン管理をし、それに加えてどういうデータでモデルを生成したかをドキュメント化することで、4のデータの依存関係の問題は緩和されます。また、5の実験コードとアルゴリズムのパラメータがコード上に散逸する問題も、モデルのバージョン管理とドキュメント化をすることで取り組みやすくなるでしょう。

1.3.4 データ処理のパイプラインごと保存する

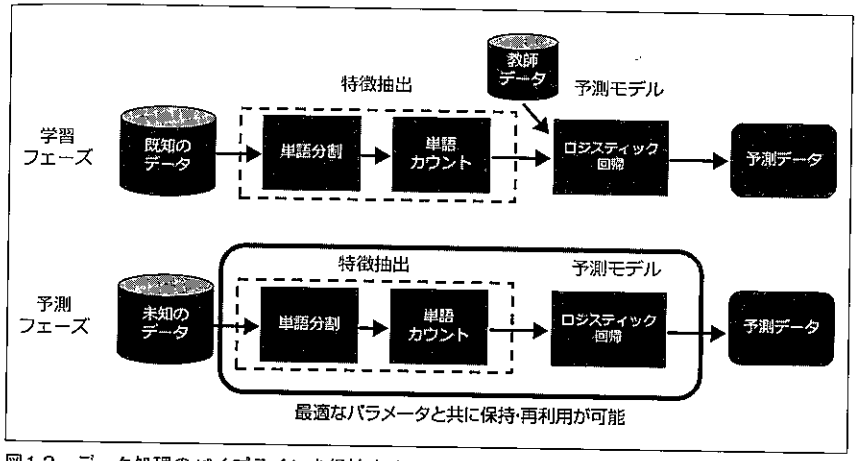


図1-3 データ処理のパイプラインを保持する

続いて、データ処理のパイプラインの保持について説明します（図1-3）。予測モデルを作成する際には、予測モデル自身のハイパーパラメータもチューニングしますが、それ以前の前処理にもパラメータが含まれることが多いです。例えば、テキスト処理では単語に分割をし、その頻度を数えた後に低頻度な単語や高頻度な単語を除外することが多くあります。こうしたときの、低頻度と足切りするしきい値は何が良いのか、といった点もチューニングポイントに含まれます。

だんだんとパラメータの数が増えデータが複雑になると、開発と本番でのパラメータ

がずれてしまって想定していた性能が出ないという事態が発生します。これを防ぐためにも、前処理から予測モデルの構築までを含めたデータ処理のパイプライン全体を保存することが重要になります。

データ処理をパイプラインとして扱いやすくするという点においては、scikit-learnの抽象化はよくできています。それに影響を受けて、Sparkなどの機械学習ライブラリも、パイプラインで処理の再利用ができるようになっています。

パイプラインごと保存をすることで対応するコードがまとまるため、3の処理のパイプラインが複雑になったときにも見通しやすくなります。また、5の実験コードとアルゴルズムのパラメータが散らかる問題も、パイプラインでまとめることで一箇所で管理しやすくなります。

1.3.5 開発/本番環境の言語/フレームワークは揃える

最後に、開発と本番環境の言語/フレームワークはできる限り揃えましょう。例えば、予測モデルの開発はRで行い、アプリケーション側の予測処理はJavaで再実装をすると考えます。この場合、両言語での実装が必要になるため、アルゴリズムの変更コストが跳ね上がってしまいます。せっかくRで高速なプロトタイピングをしたのに、本番に反映するのに時間がかかってしまったり、場合によっては断念したりしてしまうことが起こりえます。

予測モデル開発は実験を何度も繰り返すことが多く、整理されたコードではない場合も多いです。実験用のコードから本番環境のアプリケーションコードへの移行も、共通のフレームワークを利用していれば移植のコストが低くなり意思疎通もしやすくなるでしょう。

6の課題についても、言語・フレームワークを揃えることがシステムの複雑性を抑える取り組みになります。

ただし、昨今ではmicroservices[†8]と呼ばれる機能やサービスごとにAPIやメッセージキューを通じてやりとりをするアーキテクチャも増えてきています。このアーキテクチャは、大きな1つのアプリケーションを作るmonolithicなアーキテクチャと比較すると、システムを細かく分割しAPIなどでやりとりをするため、機械学習の処理部分を切り出しやすくなります。この考え方にのっとり、機械学習用のRESTやgRPCなどのAPIサーバを立てたりすることも増えてきています。Dockerなどのコンテナ技術とそれをデプロイしやすいクラウドサービスの登場により、機械学習の学習や予測の機能を切り出しAPIサーバを構築にも取り組みやすくなっています。特に、言語の選択は開発チームのスキルセットにも影響を与えてきますので、システム全体の設計やメンバーのスキルセットと照らし合わせて判断してください。

†8 https://martinfowler.com/articles/microservices.html

このように、実システムの機械学習プロダクトでは変化に耐えるための工夫をすることが重要です。より詳細なベストプラクティスを知りたい場合は、「Rules of Machine Learning: Best Practices for ML Engineering」[mlbestpractice]を参考にしてください。

scikit-learnはなぜ古典的機械学習のデファクトスタンダードになったのか？

scikit-learnは、深層学習以前の古典的機械学習のデファクトスタンダードとなったと言っても過言ではないでしょう。その一番の理由は、学習には`fit()`関数、予測には`predict()`関数といったように統一したAPIが提供されていることだと思います。統一したAPIのおかげで、scikit-learnでは交差検証用の関数`cross_val_score()`などの補助的な関数やクラスを様々なアルゴリズムに対して実施できます。

また、複数のアルゴリズムの切り替えも容易です。昔のライブラリは、単一か限られた数のアルゴリズムしか実装されておらず、アルゴリズムごとの性能比較をするには多くのコストが必要でした。それが、きれいな抽象化されたAPIのおかげで、前処理のしきい値のリストを作成することで容易にパラメータ探索を実行できたり、学習用のアルゴリズムのリストを作成することで各アルゴリズムのモデルを一度に作成できたりと、パラメータやアルゴリズムの探索を自動化するのも簡単です。

更に前処理やアルゴリズムのパイプラインを作成し、性能が最も良い組み合わせを保存することができます。これにより、ベストなパイプラインを予測時に再利用するといったことも簡単に行えます。

これらを統一的に操作できるメリットのため、Sparkなど他のライブラリでも同様のデザインを採用することが増えています。

1.4 機械学習を含めたシステムを成功させるには

機械学習を用いたシステムづくりには、ある種ギャンブルの要素があります。通常のコンピュータシステムでは、適切な設計がされていれば、何かしら動くものはできます(もちろん、ただ動くだけでは意味がなく、きちんとユーザーに対して価値のあるものを届けることは難しいです)。一方で、機械学習を含めたシステムの場合は、数週間から数ヶ月かけても、意味のあるアウトプットが何もできないという最悪のケースもありえます。

分類のモデルを作ったけれどランダム出力のほうが良い性能となることも頻繁に起こります。機械学習を含めたシステムの開発は、通常のWebシステムの開発で期待されるような1,2週間という短いサイクルでイテレーションを回すのは難しく、数ヶ月かかることも珍しくありません。

では、機械学習を含めたプロダクトをビジネスとして成功させる上で重要になるのは、どのようなチームでしょうか?私は以下の4者が重要だと考えます。[†9]

1. プロダクトに関するドメイン知識を持った人
2. 統計や機械学習に明るい人
3. データ分析基盤を作れるエンジニアリング能力のある人
4. 失敗しても構わないとリスクを取ってくれる責任者

ドメイン知識を持った人はとても重要です。解くべき課題は何か?プロダクトのどこに機械学習の手法を使えばいいのか?を考える際に、ドメイン知識がないと全く見当違いの手法を選んでしまいかねません。また特徴量を決めたりデータを収集したりするときにも、ドメイン特有の知識が重要なカギとなります。

機械学習の知識がある人は、おそらく読者の皆さんが目指しているポジションだと思います。実装をすることに加えて、問題設定をするために他の人達とのコミュニケーションを深めていくことも求められるでしょう。

昨今ではデータエンジニアと呼ばれる、データを活用できるようにする分析基盤を作るエンジニアが重要視されてきています。こうした人達と協力しながら、機械学習の基盤はどうあるべきかを考えていきましょう。

最後に、リスクを取ってくれる責任者が重要です。これは、機械学習がリスクの大きい投資だということを認識した上で、それでも機械学習を使わないとできない価値を生み出すことに背中を押してくれる存在です。できれば、機械学習やデータ分析の経験がある人の方が、詳細なリスク説明を省くことができ、意思決定までの時間が短くなって、現場は楽になります。そうでない場合は、プロダクトに関わる人がリスクを取ってでもやる必要性を説明するしかありません。時にはこの責任者に、不確実性に対する強権を発動してでも機械学習に投資をしてもらう必要があります。プロダクトを進める上で、強力な味方になってくれそうな人を探しましょう。

機械学習のプロダクトを作っていくうえでの進め方は、以下に示す2つの資料が役に立つと思います。通常のプロジェクトマネジメントと似ている所、違う所を含めて参考にして下さい。
http://www.slideshare.net/shakezo/mlct4
http://www.slideshare.net/TokorotenNakayama/2016-devsumi

1.5 この章のまとめ

本章では、機械学習プロジェクトの進め方の流れとそのポイントについて説明しました。

- 解くべき問題の仮説を立て、MVPを作りコンセプトの検証を最優先する
- 機械学習をしないことを恐れない
- 機械学習に適している問題設定かを見極める
- 予測性能とKPIの両方のモニタリングし、継続して改善を続ける

機械学習のプロジェクトは、どういう予測結果になるか特性が分からない未知のデータに対して探索的に試行錯誤をするため、通常のプロジェクトと比較して手戻りが発生しやすいです。ビジネスの目的明確にし、仮説をきちんと立てて、価値を出すためにはどうしたらよいかを考えながらプロジェクトを進めていきましょう。

†9 4人は別々の人格の場合もありますし、一人で複数役を行う場合もあります。ですが、一人でなんでもできる人を集めると属人性が高くなり、プロジェクトの持続可能性が下がってしまうことに気をつけてください。