scikit-learnとTensorFlowによる実践機械学習 2章

阿部 泰之

タイムスケジュール

17:30 ～　開場　受付開始
17:30 ～ 17:45　発表準備
17:45 ～ 18:45　1章　機械学習の現状
18:45 ～ 18:55　休憩
18:55 ～ 19:55　2章　エンドツーエンドの機械学習プロジェクト
20:00 ～ 20:30　今後のスケジュール、担当について
20:30 ～ 21:00　片づけ撤収

自己紹介

・阿部 泰之　/ Hiroyuki Abe
・「scikit-learnとTensorFlowによる実践機械学習」の輪読会をやると決めた人で、場所の用意もしています。

・業務エンジニア
　（生命保険　主に保険金支払）

・twitter　　/ @taki_tflare
・https://tflare.com

グループ紹介

AI&機械学習しよう！（Do2dle）
・https://www.facebook.com/groups/do2dle/

AI&機械学習関連の勉強会を実施しています。
非公開グループですので、上記から参加をお願いします。

最近のニュース

個人的な補足

- TensorFlow 2.0 is coming

https://groups.google.com/a/tensorflow.org/forum/#!topic/announce/qXfsxr2sF-0/discussion

Eager execution will be a central feature of 2.0. It aligns users' expectations about the programming model better with TensorFlow practice and should make TensorFlow easier to learn and apply.

最近のニュース

個人的な補足

'Define and Run'(Tensorflow, Keras): ネットワークを固定してから学習

・利点：最適化が容易

・欠点：データ構造によってモデルを変えるのが難しい

'Define by Run'(Chainer, PyTorch, DyNet)：　ネットワークは順伝搬後に確定し学習

・利点：データ構造によってモデルを変えるのが簡単、デバッグが容易

・欠点：最適化が難しい

'Define by Run’型の深層学習フレームワークが自然言語処理に向いている理由

https://qiita.com/GushiSnow/items/aa660c7228b7024076a8　より引用

アジェンダ

下記の2章について輪読を行います。

はじめに

随時コメント下さい。

本の内容に沿っていない補足の箇所は、その部分の右上に下記をつけています。（すべて補足の場合はページ右上につけます。）

個人的な補足

ついていない場合は、本の内容のままです。

2章 エンドツーエンドの機械学習プロジェクト

- この章では、最近、不動産会社に採用されたデータサイエンティストになったつもりで、プロジェクト†1を最初から最後まで体験していただこう。主要なステップは次に示す通りだ。[†1]プロジェクト例は、まったくのフィクションである。目標は、不動産取引の実際について学ぶことではなく、機械学習プロジェクトの主要なステップを具体的に説明することだ。

2章 エンドツーエンドの機械学習プロジェクト

1. 全体の構図をつかむ。

2. データを手に入れる。

3. 洞察を得るためにデータを見つけ出し、可視化する。

4. 機械学習アルゴリズムが処理しやすいようにデータを準備する。

5. モデルを選択して訓練する。

6. モデルを微調整する。

7. ソリューションをプレゼンテーションする。

8. システムを本番稼働、モニタリング、メンテナンスする。

2.1　実際のデータの操作

この章では、StatLibリポジトリ†2（図2-1）のカリフォルニアの住宅価格のデータセットを使うことにした。

2.2 全体像をつかむ

機械学習ハウジング株式会社にようこそ。あなたが最初に与えられた仕事は、カリフォルニア州の国勢調査データを使ってカリフォルニアの住宅価格のモデルを作ることである。

あなたのモデルは、このデータを使って学習し、ほかのすべての指標から任意の区域の住宅価格の中央値を予測できなければならない。

2.2.1　問題の枠組みを明らかにする

上司には、まずビジネスサイドの目的が何なのかを尋ねよう。モデルを構築することは、たぶん最終的な目標ではない。会社はこのモデルをどのように使うつもりで、何を得たいのだろうか。これが重要なのは、問題をどのように組み立てていくか、どのアルゴリズムを選択するか、モデルの評価のためにどのような性能指標を使うか、どれくらいの労力をかけるべきかといったことがこれによって左右されるからだ。

2.2.1　問題の枠組みを明らかにする

上司は、ほかの多くの信号（signal）†3とともに、モデルの出力（区域の住宅価格の中央値の予測値）をほかの機械学習システムに与えるのだと答える（図2-2）。この下流のシステムは、その地域に投資する価値があるかどうかを判断する。収益に直接影響を与えるので、これを正しく判断することはきわめて重要である。

図2-2　不動産投資の機械学習パイプライン参照

2.3 データを手に入れる

個人的な補足

- Google Colab上にコードを置いています。

下記共有しているので、セルごとに実行可能です。

https://colab.research.google.com/drive/1g3o7mCbMY4N9U_VnMJnIzoE_TqxYseAx

すべてのセルをまとめて実行したい場合、コピーしてから実行して下さい。

2.3.3 データの構造をざっと見てみる

- 各行がひとつの区域を表している。
- 属性は、longitude（経度）、latitude（緯度）、housing_median_age（築年数の中央値）、total_rooms（部屋数）、total_bedrooms（寝室数）、population（人口）、households（世帯数）、median_income（収入の中央値）、median_house_value（住宅価格の中央値）、ocean_proximity（海との位置関係）の10個である

2.3.3 データの構造をざっと見てみる

- データセットのインスタンス数は20,640で、機械学習の常識からするとかなり小さいが、最初に扱うものとしてはまったく問題ない。total_bedrooms属性には、nullではない値が20,433個しかないことに注意しよう。これは、この特徴量を持たない区域が207あるということである。このことにはあとで注意を払う必要がある。

2.3.3 データの構造をざっと見てみる

- 「ヒストグラムから気付くことがいくつかある」から読んでいきます。

2.3.4 テストセットを作る

- データセットのインスタンス数は20,640で、機械学習の常識からするとかなり小さいが、最初に扱うものとしてはまったく問題ない。total_bedrooms属性には、nullではない値が20,433個しかないことに注意しよう。これは、この特徴量を持たない区域が207あるということである。このことにはあとで注意を払う必要がある。

2.3.4 テストセットを作る

- さて、今まで説明してきたのは、純粋に無作為なサンプリング方法である。データセットが十分大規模ならそれでよいのだが（特に属性数との相対的な割合で）、そうでなければ、大きなサンプリングバイアスを持ち込む危険がある。たとえば、調査会社が1,000人の人に電話をかけて質問をするときには、電話帳で無作為に1,000人の人々を拾い出すわけではない。人口全体を代表するような1,000人になるように努力する。たとえば、米国の人口は女性が51.3%、男性が48.7%なので、ていねいに実施されている調査では、サンプルでも同じ比率を守ろうとする。つまり、513人の女性と487人の男性に尋ねるのである。これは、層化抽出法（stratifiedsampling）と呼ばれている。

2.4 洞察を得るためにデータを研究、可視化する

- 2.4.1 地理 データの可視化

- Google Colab参照

https://colab.research.google.com/drive/1g3o7mCbMY4N9U_VnMJnIzoE_TqxYseAx

これでだいぶよくなった。高密度の地域、すなわちベイエリアとロスアンゼルス、サンディエゴ、特にサクラメントとフレズノを中心とするセントラルバレーがはっきりとわかる。

2.4 洞察を得るためにデータを研究、可視化する

- 2.4.1 地理 データの可視化

- 住宅データ　Google Colab参照

https://colab.research.google.com/drive/1g3o7mCbMY4N9U_VnMJnIzoE_TqxYseAx

このイメージからは、住宅価格が位置（たとえば、海の近く）、人口密度と密接な関係を持っていることがわかる。これはすでにわかっていたことだろう。おそらく、クラスタリングアルゴリズムを使って主要なクラスタを見つけ出し、クラスタの中心との距離を表す新しい特徴量を追加すると役に立つ。太平洋との距離の属性も役に立つかもしれない。ただし、北カリフォルニアでは、海岸沿いの住宅価格はそれほど高くないので、これは単純なルールではない。

2.4.2 相関を探す

- 相関係数は、−1から1までの範囲である。1に近ければ、強い正の相関があるという意味になる。たとえば、収入の中央値が高くなると、住宅価格の中央値も高くなりやすい。それに対し、係数が−1に近くなると、強い負の相関がある。

2.4.2 相関を探す

- median_house_value 1.000000
- median_income 0.687170
- total_rooms 0.135231
- housing_median_age 0.114220
- households 0.064702
- total_bedrooms 0.047865
- population -0.026699
- longitude -0.047279
- latitude -0.142826

個人的な補足

- −1.0から−0.7 強い負の相関
- −0.7から−0.4 負の相関
- −0.4から−0.2 弱い負の相関
- −0.2から+0.2 ほとんど相関がない
- +0.2から+0.4 弱い正の相関
- +0.4から+0.7 正の相関
- +0.7から+1.0 強い正の相関

2.4.2 相関を探す

個人的な補足

住宅価格の中央値と相関があるのは、収入の中央値

他は殆ど相関がない。

2.4.2 相関を探す

- 住宅価格の中央値を予測するためにもっとも使える値は収入の中央値なので、その相関を表す散布図を大きく表示しよう（図2-16）。

  housing.plot(kind="scatter",x="median_income",y="median_house_value",alpha=0.1)この図からはいくつかのことがわかる。まず第1に、相関が本当に非常に強いことである。上向きの傾向がはっきりと現れ、点はあまり散らばっていない。第2に、以前触れた価格の上限の設定が、50万ドル近辺の横線という形ではっきりと現れている。しかし、この図には、これよりも少し目立たない直線もある。45万ドル近辺の横線、35万ドル近辺の横線、そして28万ドル近辺にもおそらく横線があり、それよりも低いところにも横線がある。アルゴリズムがこのようなデータの癖を再現しないように、対応する区域を訓練セットから取り除くようにしたい。

2.4.3 属性の組み合わせを試してみる

- 機械学習アルゴリズムに渡せるようにデータを実際に準備する前に、最後にしておきたいことがもうひとつある。さまざまな属性を結合してみることだ。たとえば、区域の部屋数の合計がわかっても、区域の世帯数がいくつかがわからなければあまり意味はない。本当に知りたいのは、世帯あたりの部屋数である。同様に、寝室の総数もそれ自体では意味がない。部屋数と比較してみたいはずだ。そして、世帯あたりの人数も、面白そうな属性の組み合わせ方である。こういった新属性を作ってみよう。

2.4.3 属性の組み合わせを試してみる

- bedrooms_per_room -0.259984

寝室数／部屋数の割合が低い家の方が値段が高くなる傾向があるのは明らかだ。

-0.25は、弱い負の相関　個人的な補足

相関がある組み合わせを発見できました。

2.4.3 属性の組み合わせを試してみる

- データ探索のこの部分は、徹底的なものである必要はない。ポイントは、よい出発点を見つけて、早く洞察をつかみ、最初のプロトタイプとして十分によいものを手に入れることだ。しかし、この部分は反復的なプロセスになる。プロトタイプを動かしてその出力を分析すると、さらに洞察が得られ、そこからこの探索ステップに戻ってくることがある。

2.5 機械学習アルゴリズムに渡せるようにデータを準備する

- ### 2.5.1　　データをクリーニングする

ほとんどの機械学習アルゴリズムは欠損特徴量を処理できないので、それに対応するための関数を作っておこう。先ほど気付いたように、total_bedrooms属性には欠損値があるので、それに対処するのである。

2.5.1　データをクリーニングする

オプション1：対応する区域を取り除く。

オプション2：属性全体を取り除く。

オプション3：何らかの値を設定する（0、平均、中央値など）

DataFrameのdropna()、drop()、fillna()メソッドを使えば、これらは簡単に実現できる。

```
#オプション1

housing.dropna(subset=["total_bedrooms"])
#オプション2

housing.drop("total_bedrooms",axis=1)
#オプション3

median=housing["total_bedrooms"].median()
```

2.5 scikit-learnのAPI

- scikit-learnのAPIは、非常に見事に設計されている。その主要な設計原則（http://goo.gl/wL10sI）は、次の通りである†15。

- 一貫性：すべてのオブジェクトが首尾一貫した単純なインターフェイスを持っている。

2.5 scikit-learnのAPI

個人的な補足

- 一貫したインターフェイスを持つことによって、モデル間の比較が容易に行える。
- TensorFlowにもscikit-learnに影響を受けた高レベルAPI tf.learnがあります。
- 例えば、scikit-learnでは、次ページ以降のコードでモデル間の比較ができる。データは前処理済み

個人的な補足

モデルの import

```
from sklearn.linear_model import
LogisticRegression

from sklearn.svm import SVC, LinearSVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import
RandomForestClassifier

from sklearn.naive_bayes import GaussianNB

from sklearn.neural_network import MLPClassifier


from sklearn.model_selection import GridSearchCV,
cross_val_score
```

個人的な補足

モデルの定義

```
predictors = ["Pclass", "Sex", "Age", "SibSp",
"Parch", "Fare", "Embarked"]



models = []



models.append(("LogisticRegression",LogisticRegress
ion()))

models.append(("SVC",SVC()))

models.append(("LinearSVC",LinearSVC()))

models.append(("KNeighbors",KNeighborsClassifier()
))

models.append(("DecisionTree",DecisionTreeClassifier
()))

models.append(("RandomForest",RandomForestClass
ifier()))
```

個人的な補足

```
rf2 = RandomForestClassifier(n_estimators=100, criterion='gini',
                             max_depth=10, random_state=0, max_features=None)
rf3 = RandomForestClassifier(max_depth=20, max_features=None,
                             min_samples_split=10, n_estimators=100, n_jobs=1, random_state=0)
models.append(("RandomForest2",rf2))
models.append(("RandomForest3",rf3 ))
models.append(("MLPClassifier",MLPClassifier(solver='lbfgs', random_state=0)))
```

モデルのパラメータ定義

```
results = []
names = []
for name,model in models:
    result = cross_val_score(model, train_data[predictors], train_data["Survived"],  cv=3)
    names.append(name)
    results.append(result)
```

交差検証による評価

```
for i in range(len(names)):
    print(names[i],results[i].mean())
```

出力

個人的な補足

結果

LogisticRegression 0.785634118967

SVC 0.68911335578

LinearSVC 0.739618406285

KNeighbors 0.703703703704

DecisionTree 0.772166105499

RandomForest 0.780022446689

RandomForest2 0.820426487093

RandomForest3 0.813692480359

MLPClassifier 0.778900112233

ソースコード全文は以下参照

https://www.kaggle.com/tflare/testing-multiple-models-with-scikit-learn-0-79425

2.5.4　特徴量のスケーリング

- データに対して実行しなければならない変換のなかでも特に重要なもののひとつが特徴量のスケーリング（featurescaling）である。ごく一部の例外を除き、機械学習アルゴリズムは、入力の数値属性のスケールが大きく異なると性能を発揮できない。住宅価格データにもこれは当てはまる。総部屋数は6から39,320までの大きな範囲になっているのに、収入の中央値は0から15までの範囲である。なお、ターゲット値のスケーリングは一般に不要だということに注意していただきたい。

2.5.4　特徴量のスケーリング

- すべての属性のスケールを統一するためによく使われている方法としては、最小最大スケーリング（min-maxscaling）と標準化（standarization）のふたつがある。最小最大スケーリング（多くの人々はこれを正規化:normalizationと呼んでいる）は、ごく単純な方法で、0から1までに収まるように値をスケーリングし直すだけである。値から最小値を引き、最大値と最小値の差で割ればよい。scikit-learnは、この目的のためにMinMaxScalerという変換器を提供している。また、何らかの理由で範囲を0から1までにしたくないときに範囲を変えられるfeature_rangeハイパーパラメータもある。

2.5.4　特徴量のスケーリング

- 標準化はこれとは大きく異なる。まず、値から平均値を引き（そのため、標準化された値の平均はかならず0になる）、その値を分散で割って得られる分布が単位分散になるようにする。最小最大スケーリングとは異なり、標準化には上下限がなく、特定の範囲には収まらないので、一部のアルゴリズムではそれが問題になる（たとえば、ニューラルネットワークは、入力値が0から1までの範囲に収まっていることを前提とすることが多い）。しかし、標準化は最小最大スケーリングよりも外れ値の影響が小さくなる。たとえば、ある区域の収入の中央値が100だとする（何かの間違いにより）。この場合、最小最大スケーリングでは、0から15までの範囲のほかの値は0から0.15までの範囲に押し込まれてしまうが、標準化ならそのような大きな影響は出ない。scikit-learnは、標準化のためにStandardScalerという変換器を用意している。

2.5.5　変換パイプライン

- 今までの説明からもわかるように、データ変換のステップはいくつもあり、それを正しい順序で実行しなければならない。幸い、scikit-learnには、そのような変換シーケンスを実行しやすくするPipelineクラスがある。

2.6　モデルを選択して訓練する

- ついにここまで来た。今までに、問題を構成し、データを入手して探索し、機械学習アルゴリズムのためにデータを自動的にクリーンアップ、準備する変換パイプラインを書いてきて、ついに機械学習モデルを選択、訓練する準備が整ったのである。

2.6.1　訓練セットを訓練、評価する

- 今までのステップのおかげで、みなさんが思っているのと比べて仕事ははるかに単純になっている。まず、前章で行ったように、線形回帰モデルを訓練してみよう。

from sklearn.linear_model

import LinearRegressionlin_reg=LinearRegression()

lin_reg.fit(housing_prepared,housing_labels)

これで終わりだ。使える線形回帰モデルがもう作られている。

2.6.2　交差検証を使ったよりよい評価

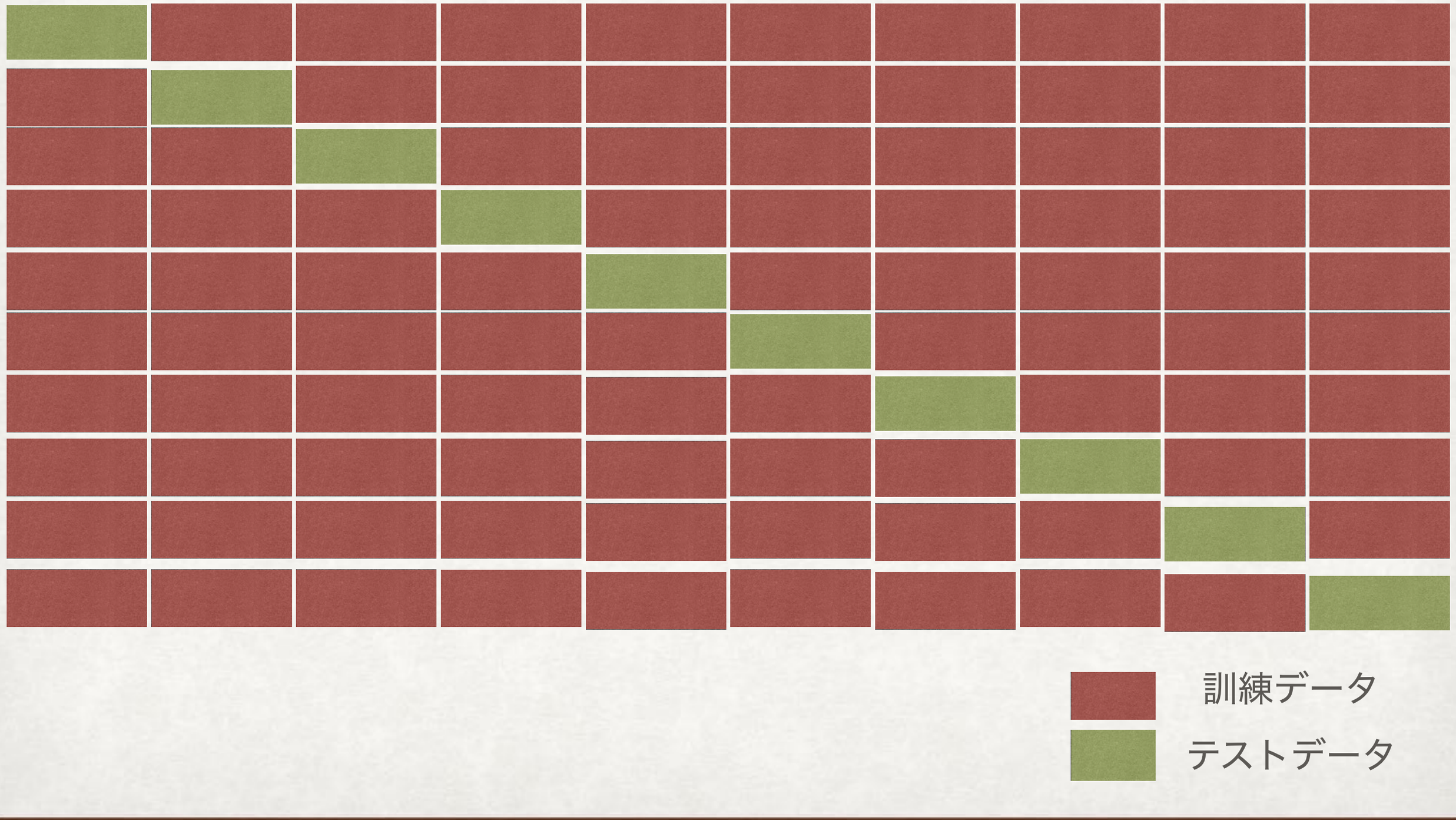
- 決定木モデルを評価するためのひとつの方法は、train_test_split関数を使って訓練セットを小さな訓練セットと検証セットに分割し、小さい方の訓練セットでモデルを訓練して検証セットで評価するものだ。少し手間だが、難しすぎるようなことはいっさいなく、うまく機能する。もうひとつの優れた方法は、scikit-learnの交差検証（cross-validation）である。次のコードはK分割交差検証（K-foldcross-validation）を行う。つまり、訓練セットをフォールド（fold）と呼ばれる10個の別々のサブセットに無作為に分割し、1個のフォールドを評価用に残し、その他9個のフォールドで訓練して、決定木モデルを10回訓練、評価するのである。結果は、10個の評価スコアから構成されたベクトルになる。

2.6.2　交差検証を使ったよりよい評価

個人的な補足

10分割ということはこういうことです。

10回実施、結果を平均して評価を出します。



2.7　モデルを微調整する

- 2.7.1 グリッドサーチ

ひとつの方法は、最適なハイパーパラメータ値の組み合わせを見つけるまで、マニュアルでハイパーパラメータを操作するものである。これはかなり面倒な作業であり、多くの組み合わせを試す時間はないかもしれない。そこで、scikit-learnのGridSearchCVにサーチをさせればよい。どのハイパーパラメータを操作するか、その値として何を試すかを指定すると、GridSearchCVは、指定から得られるハイパーパラメータ値のすべての組み合わせを交差検証で評価する。

コード

個人的な補足

https://colab.research.google.com/drive/1g3o7mCbMY4N9U_VnMJnIzoE_TqxYseAx

2.7.2　ランダムサーチ

- ランダムサーチをたとえば1,000回繰り返すと、個々のハイパーパラメータについて1,000種類の異なる値を試せる（グリッドサーチのようにハイパーパラメータごとに数個の値だけを試すのではなく）。

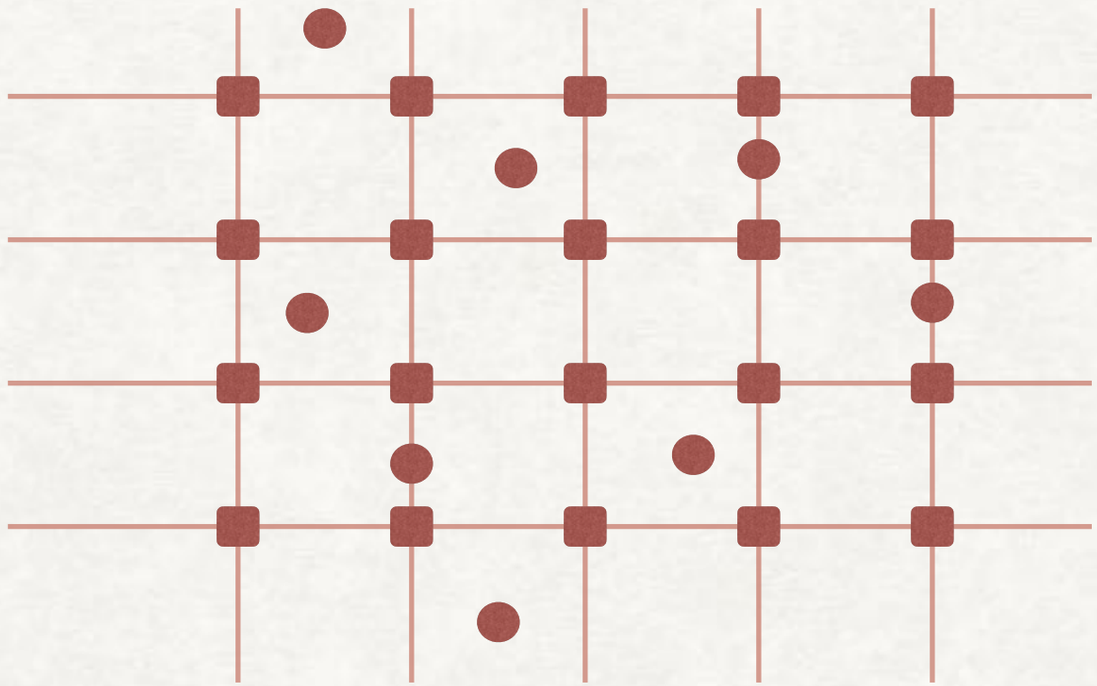
- ハイパーパラメータサーチのための計算資源の予算を管理しやすくなる。単純にイテレーションの回数を設定するだけでよい。

グリッドサーチとランダムサーチのイメージ

個人的な補足

グリッドサーチは順番に実行していく

ランダムサーチはランダムで実施のため、早めにいろいろな値で試すことができる。



■グリッドサーチ
●ランダムサーチ

機械学習モデルのハイパパラメータ最適化

個人的な補足

- 以下がまとまっていてわかりやすかったです。

機械学習モデルのハイパパラメータ最適化

https://www.slideshare.net/greetech/ss-110811527/

2.7.5　テストセットでシステムを評価する

- ハイパーパラメータの微調整をしっかり行った場合、評価から得られる性能は、交差検証で測定した性能よりもわずかに低くなるのが普通だ（システムは検証データに対して性能が高くなるように微調整されているので、未知のデータセットではそこまでの性能が出ないことが多い）。この例ではそのようなことにはなっていないが、もしそうであっても、テストセットでの数値を上げるためにハイパーパラメータをいじりたくなる気持ちを抑えなければならない。

2.8 システムを本番稼働、モニタリング、メンテナンスする

- システムの性能を評価する
- システムの入力データの品質を評価する
- モデルは、新鮮なデータを使って定期的に訓練する

2.9　試してみよう

- ご覧になったように、仕事の大半はデータの準備のステップにある。モニタリングツールを作り、人間による評価パイプラインを準備し、定期的なモデルの訓練を自動化することだ。もちろん、機械学習アルゴリズムも大切だが、高度なアルゴリズムの探索にばかり時間を使ってプロセス全体のことを考える時間が足りなくなるよりも、プロセス全体をしっかりと把握し、3〜4種類のアルゴリズムを知っている方がおそらくよい。

2.9　試してみよう

- あなたがまだそうでなければ、今すぐラップトップを開き、興味のあるデータセットを選んで、プロセス全体のAからZまでを実際にしてみよう。出発点としては、http://kaggle.com/のようなコンテストのサイトがよい。遊べるデータセットが手に入り、明確な目標を持つことができる。そして、人々が経験をシェアしてくれる。