**Azure Machine Learning を使用したスマート アプリの構築**

**概要**

機械学習は、コンピューター科学の中で最も急速に成長している分野の 1 つです。大量のデータを使用する予測分析を、それらのデータから繰り返し学習するアルゴリズムを採用することによって簡易化します。機械学習の使用範囲は、クレジットカード不正使用の検出や自動運転車から、光学文字認識 (OCR) やオンライン ショッピングでの商品のお勧めまで、多岐にわたります。機械学習は、コンピューターをよりスマートにすることによって、ユーザーをよりスマートにします。また、より多くのデータが使用可能になるにつれ、機械学習の有用性は増すばかりで、そのデータから予測分析を行いたいという要望も大きくなってきています。

[Azure Machine Learning](https://azure.microsoft.com/ja-jp/services/machine-learning/) は、クラウドベースの予測分析サービスであり、あらゆるスキル レベルのデータ科学者に、効率化されたエクスペリエンスを提供します。Azure Machine Learning と共に、Azure Machine Learning Studio (ML Studio) が提供されています。これは、ブラウザーベースのツールであり、機械学習モデルを構築する際に、使いやすいドラッグ アンド ドロップ インターフェイスを提供します。これには、時間を節約できる実験のライブラリが搭載されており、Bing などの Microsoft ビジネス ソリューションによって実世界で開発およびテストされたクラス最高のアルゴリズムが提供されます。また、[R](https://www.r-project.org/) および [Python](https://www.python.org/) のサポートが組み込まれているので、カスタム スクリプトを作成して、モデルをカスタマイズすることができます。ML Studio でモデルを構築し、トレーニングを行った後は、そのモデルをさまざまなプログラミング言語で使用可能な Web サービスとして簡単に公開することも、[Cortana Intelligence ギャラリー](https://gallery.cortanaintelligence.com/)に置いて、コミュニティで共有することもできます。

このラボでは、Azure Machine Learning を使用して、手書きの数字を認識するモデルの構築、トレーニング、およびスコア付けを行います。学術調査用に公開されている実際の OCR データ セットを使用します。モデルを Web サービスとして展開した後は、画面に数字を描けるように [Electron](http://electron.atom.io/) クライアントを作成します。その後、描いた数字を Azure Machine Learning が識別できるかどうか確認します。モデルを構築してトレーニングを行う方法、およびそのモデルを利用するコードを記述する方法を学びます。

**目的**

このハンズオン ラボでは、以下の方法について学習します。

* Azure Machine Learning Studio を使用して、モデルの構築、トレーニング、およびスコア付けを行う。
* モデルを Web サービスとして展開し、コードまたはスクリプトからアクセスできるようにする。
* 作成したアプリから ML Web サービスを呼び出す。

**前提条件**

このハンズオン ラボを完了するには、以下が必要です。

* アクティブな Microsoft Azure サブスクリプション。これがない場合は、[無料試用版](https://azure.microsoft.com/ja-jp/free/?WT.mc_id=AEDA13C29)にサインアップしてください。
* [Visual Studio Code](http://code.visualstudio.com/)。
* [Node.js](https://nodejs.org/en/)。

**演習**

このハンズオン ラボでは次の演習を行います。

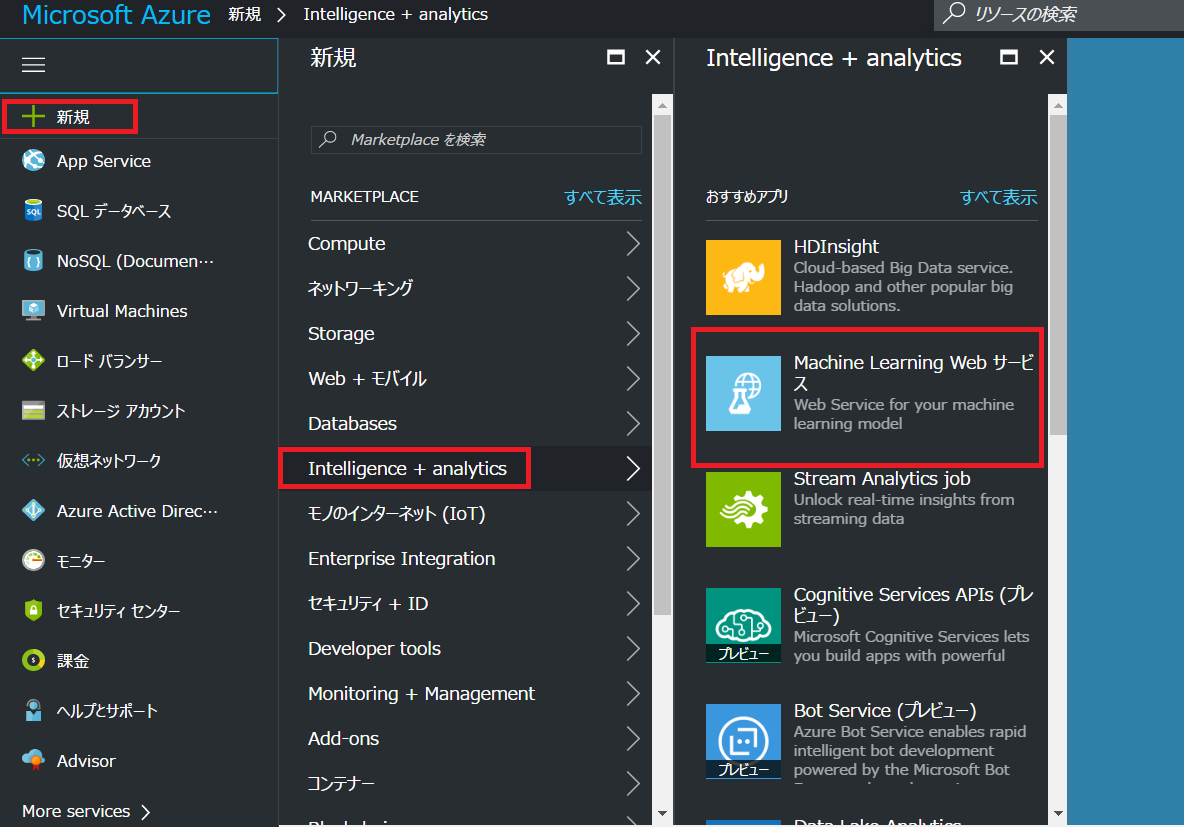
* [演習 1:](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Azure%20Machine%20Learning%20HOL%20(Node).md#Exercise1)  [Machine Learning の実験を作成する](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Azure%20Machine%20Learning%20HOL%20(Node).md#Exercise1)
* [演習 2:](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Azure%20Machine%20Learning%20HOL%20(Node).md#Exercise2)  [データセットをアップロードする](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Azure%20Machine%20Learning%20HOL%20(Node).md#Exercise2)
* [演習 3:](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Azure%20Machine%20Learning%20HOL%20(Node).md#Exercise3)  [分類モデルのトレーニングを行う](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Azure%20Machine%20Learning%20HOL%20(Node).md#Exercise3)
* [演習 4:](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Azure%20Machine%20Learning%20HOL%20(Node).md#Exercise4)  [モデルにスコアを付ける](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Azure%20Machine%20Learning%20HOL%20(Node).md#Exercise4)
* [演習 5:](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Azure%20Machine%20Learning%20HOL%20(Node).md#Exercise5)  [モデルを Web サービスとして展開する](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Azure%20Machine%20Learning%20HOL%20(Node).md#Exercise5)
* [演習 6:](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Azure%20Machine%20Learning%20HOL%20(Node).md#Exercise6)  [Electron クライアントを構築する](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Azure%20Machine%20Learning%20HOL%20(Node).md#Exercise6)
* [演習 7:](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Azure%20Machine%20Learning%20HOL%20(Node).md#Exercise7)  [モデルをテストする](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Azure%20Machine%20Learning%20HOL%20(Node).md#Exercise7)

このラボの推定所要時間: 60 分

**演習 1: Machine Learning の実験を作成する**

Azure Machine Learning を導入する最初のステップは、ML ワークスペースと、そこで行う実験の作成です。この演習では、Azure ML Studio で Machine Learning の実験を稼働させます。

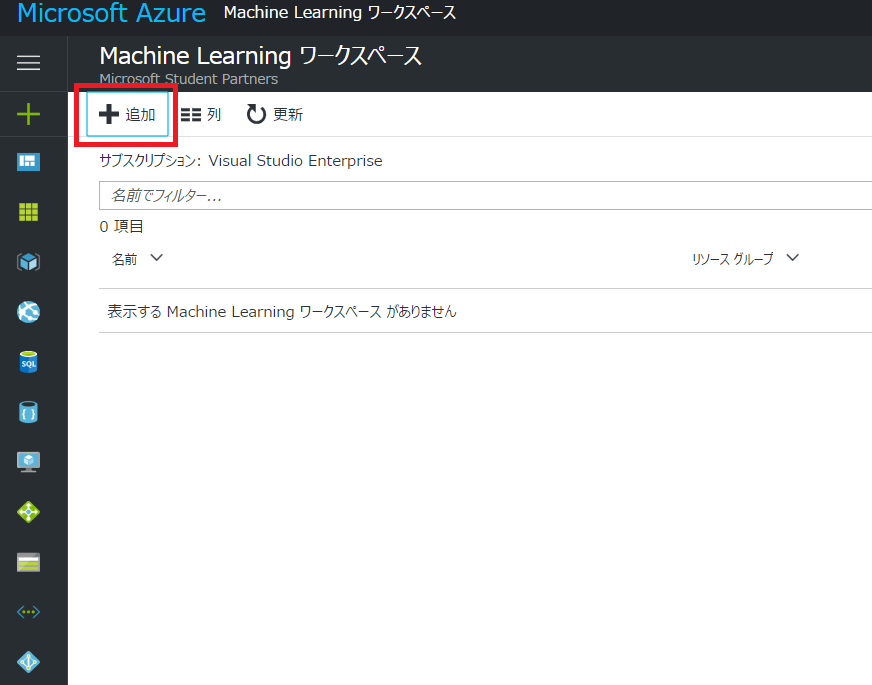
1. ブラウザーで [Azure ポータル](https://portal.azure.com)を開きます。ログインするよう求められたら、Microsoft アカウントを使用してログインします。
2. [+ New]、[Intelligence + analytics]、[Machine Learning Webサービス],[ワークスペースの表示].[追加] の順にクリックします。



*ML ワークスペースの作成*



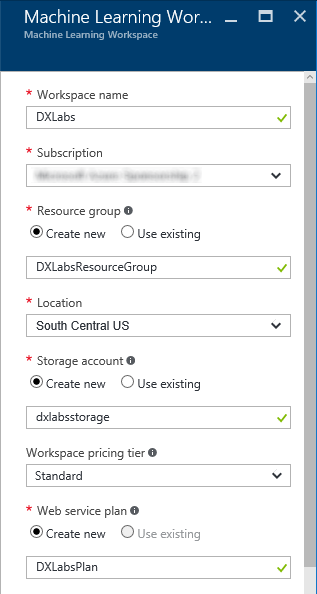
ワークスペースの表示



追加画面

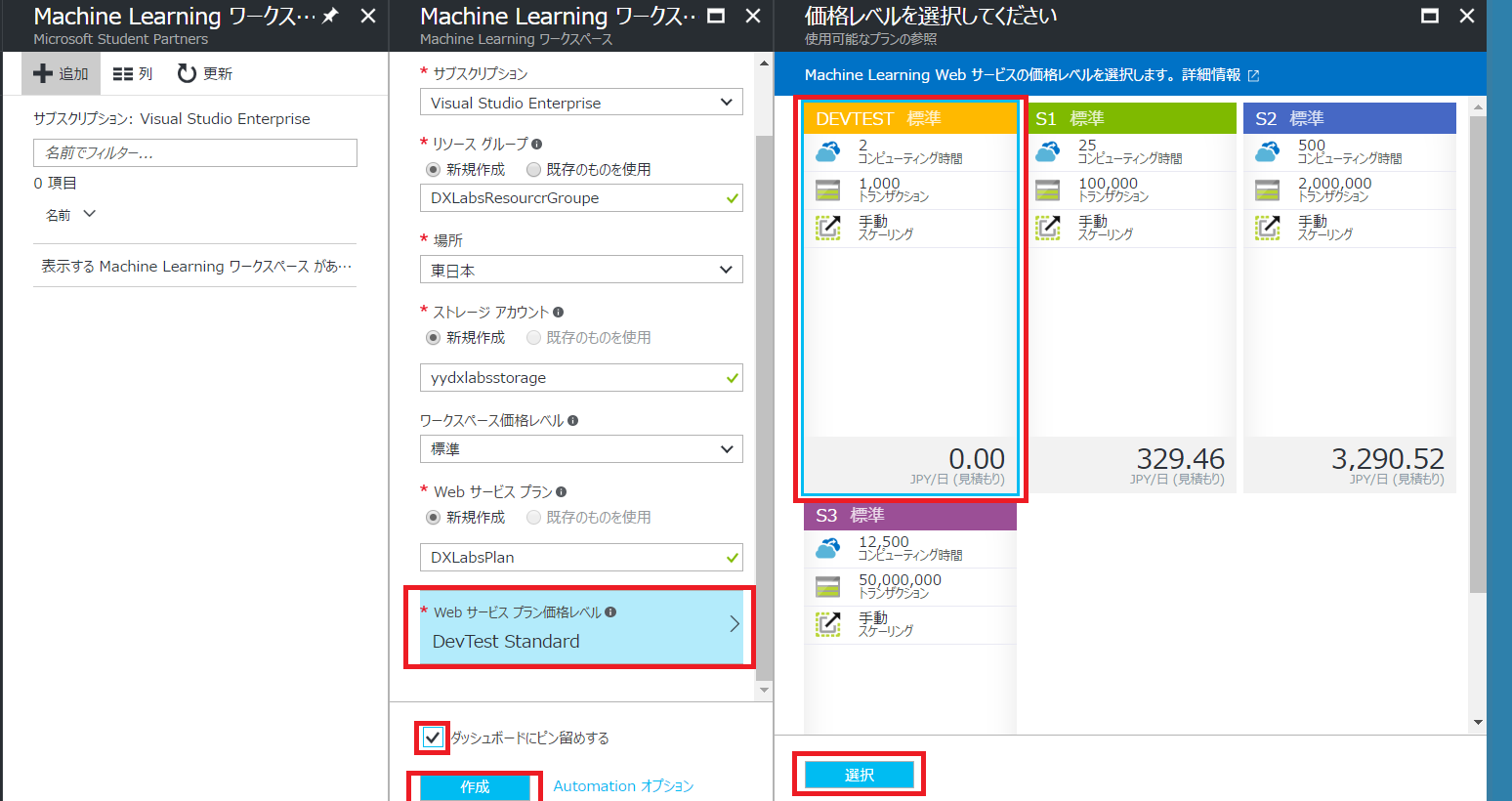
1. [Machine Learning ワークスペース] ブレードで、"DXLabs" などのワークスペース名を入力し、緑色のチェック マークがその横に表示されることを確認します。[Resource group] の [Create new] をクリックして、"DXLabsResourceGroup" などのリソース グループ名を入力します (二重引用符は付けません)。[Location] で、最も近くの場所を選択します。次に、ストレージ アカウント名を入力して、ここでもその横に緑色のチェック マークが表示されることを確認します。

ストレージ アカウント名は、3 ～ 24 文字の長さで、数字と小文字のみを使用でき、Azure 内で一意である必要があります。名前の横の緑色のチェック マークは、これらの条件を満たしていることを表します。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/workspace-parameters.png)

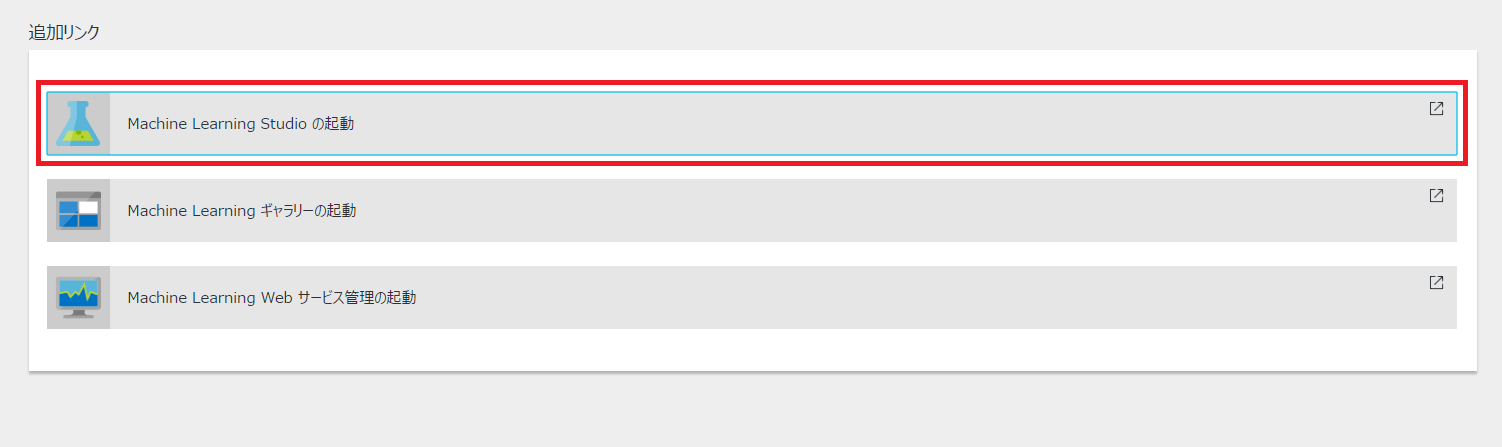
*ML ワークスペースの構成*

1. [Web サービス プラン価格レベル] をクリックして、[DEVTEST 標準] を選択します。 [Machine Learningワークスペース] ブレードの下部にある [ダッシュボードにピン留めする] をオンにし、[Create] ボタンをクリックして完了します。



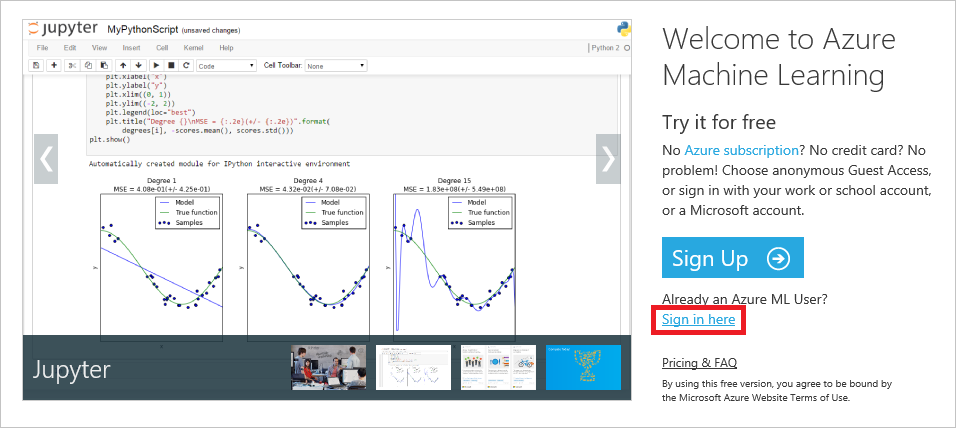
*価格レベルの選択*

1. 通常、新しい ML ワークスペースの作成には 30 秒ほどかかります。展開が完了したら、[Launch Machine Learning Studio の起動] をクリックします (次に示した画面が自動的に表示されない場合は、ダッシュボードで ML ワークスペース用に作成されているタイルをクリックして、表示します)。



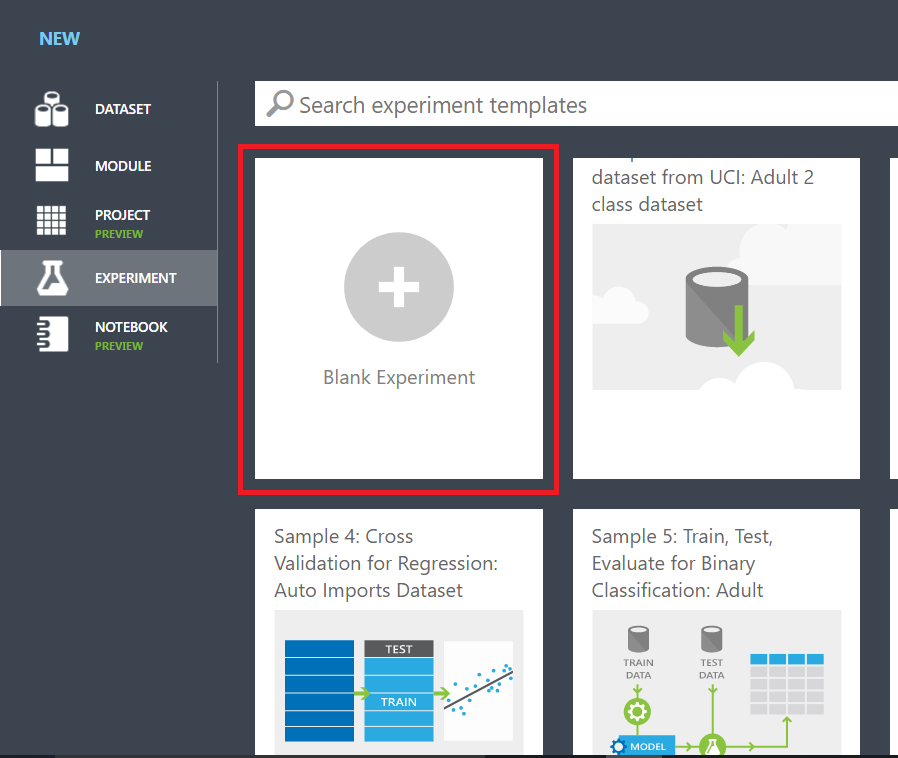
*ML Studio の起動*

1. [Sign in here] をクリックし、Microsoft アカウントを使用して ML Studio にサインインします。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/sign-in.png)

*ML Studio へのサインイン*

1. ML Studio で、左下隅にある [+ NEW] をクリックします。[Blank Experiment] をクリックして、新しい実験を開始します。



*空の実験の作成*

1. ページ上部にある実験のタイトル ("Experiment created on...") をクリックし、"Digit Recognition" などの新しい実験名を入力します。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/new-experiment-title.png)

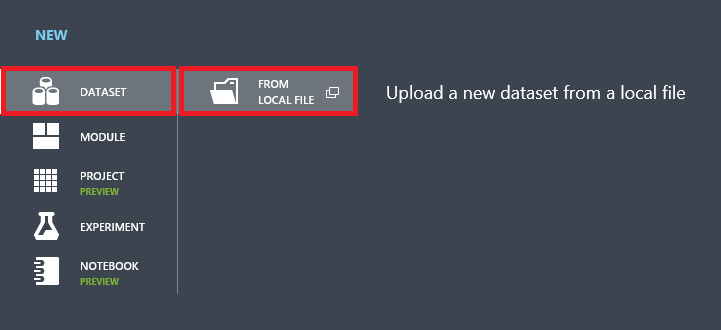
*実験名の変更*

これで、実験が作成されました。次のステップでは、データをインポートし、それに基づいてモデルを構築します。

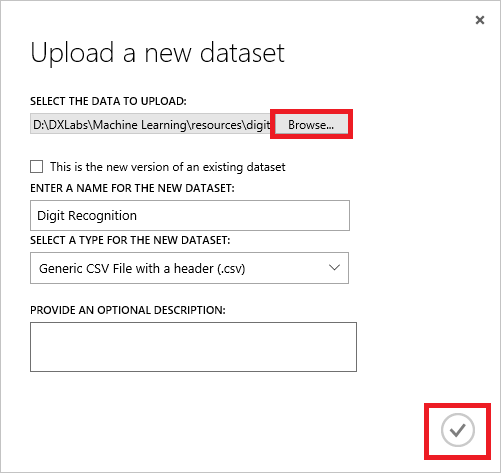
**演習 2: データセットをアップロードする**

Azure Machine Learning Studio には、サンプルのデータセットがいくつか同梱されています。追加のデータセットは、[Data.gov](http://www.data.gov/)、[Kaggle](http://www.kaggle.com/)、カリフォルニア大学アーバイン校の [Machine Learning Repository](http://archive.ics.uci.edu/ml/) など、さまざまなソースから入手可能です。この演習では、43 名分の手書きの数字 (0 ～ 9) をスキャンしてデジタル化することにより生成したデータが含まれるパブリック データセットをアップロードします。後でこのデータセットを使用して、高い精度で手書きの数字を認識できる ML モデルのトレーニングを行います。

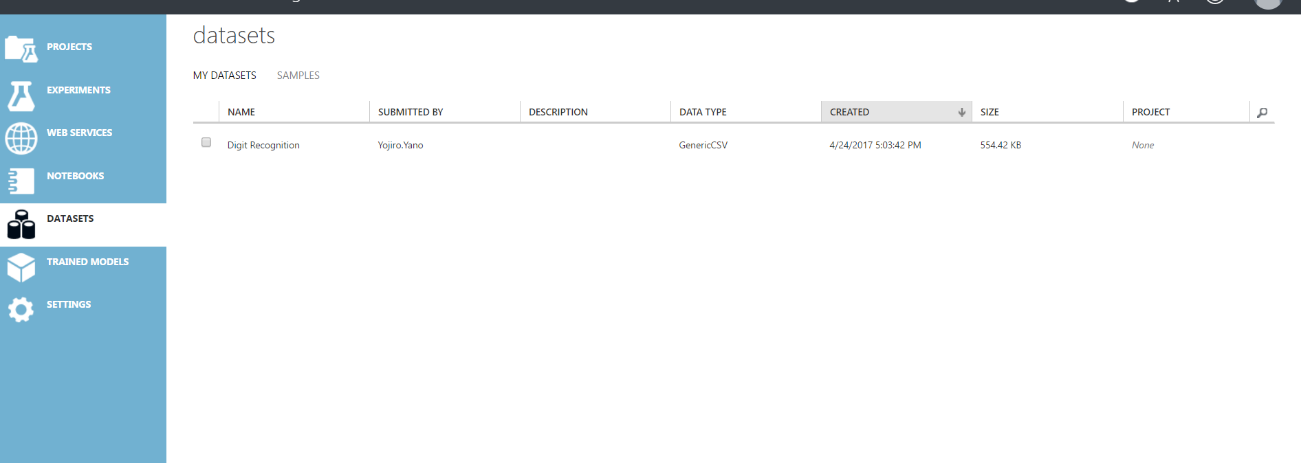
1. ML Studio の左下隅にある [+ NEW] をクリックします。[DATASET]、[FROM LOCAL FILE] の順にクリックします。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/new-dataset.png)

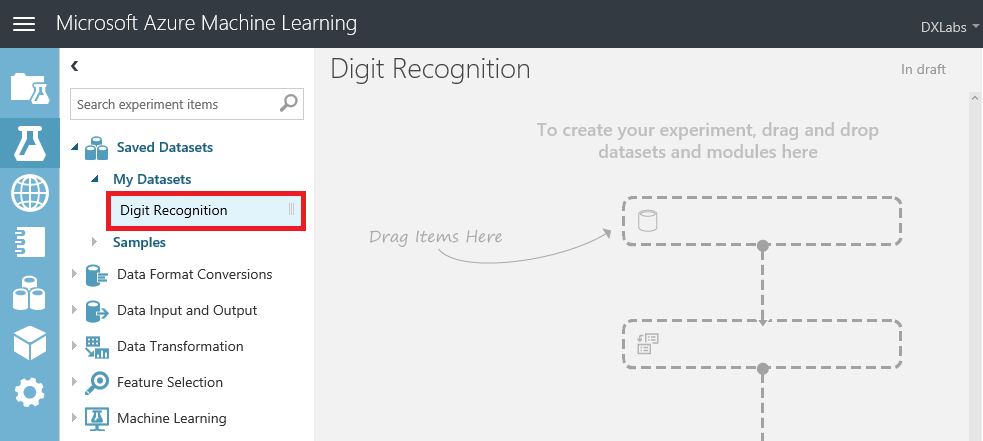
*データセットの新規作成*

1. [ファイルを選択] ボタンをクリックします。このラボの resources サブディレクトリに移動して、**digit-recognition.csv** という名前のファイルを選択します。[SELECT A TYPE FOR THE NEW DATASET] で "**Generic CSV File with a header (.csv)**" が選択されていることを確認します。オプションで、[ENTER A NAME FOR THE NEW DATASET] ボックスに、データセットのフレンドリ名を入力し、右下隅にあるチェック マークをクリックして、データセットのアップロードを開始します。[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/upload-dataset.png)

*データセットのアップロード*

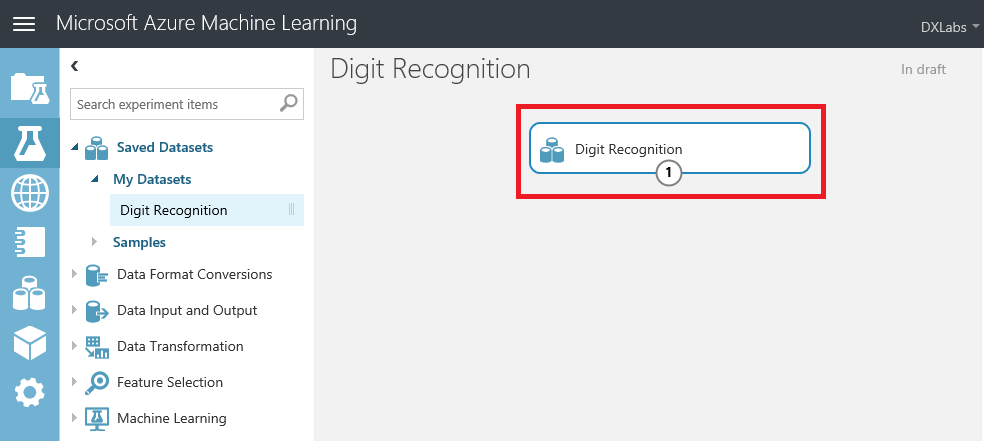
1. アップロードが完了するのを待ちます。左側にあるモジュール パレットに移動し、[Saved Datasets] の [My Datasets] の下に、アップロードしたデータセットが表示されていることを確認します.
2. 

Datasetsの確認画面

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/select-dataset.png)

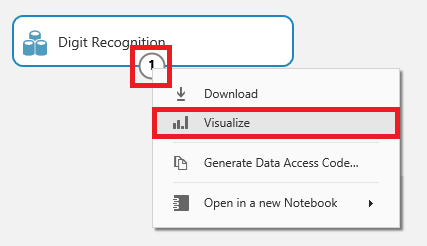
*アップロードしたデータセット*

1. モジュール パレットからキャンバス (右側の灰色の領域) に、データセットをドラッグ アンド ドロップします。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/add-dataset.png)

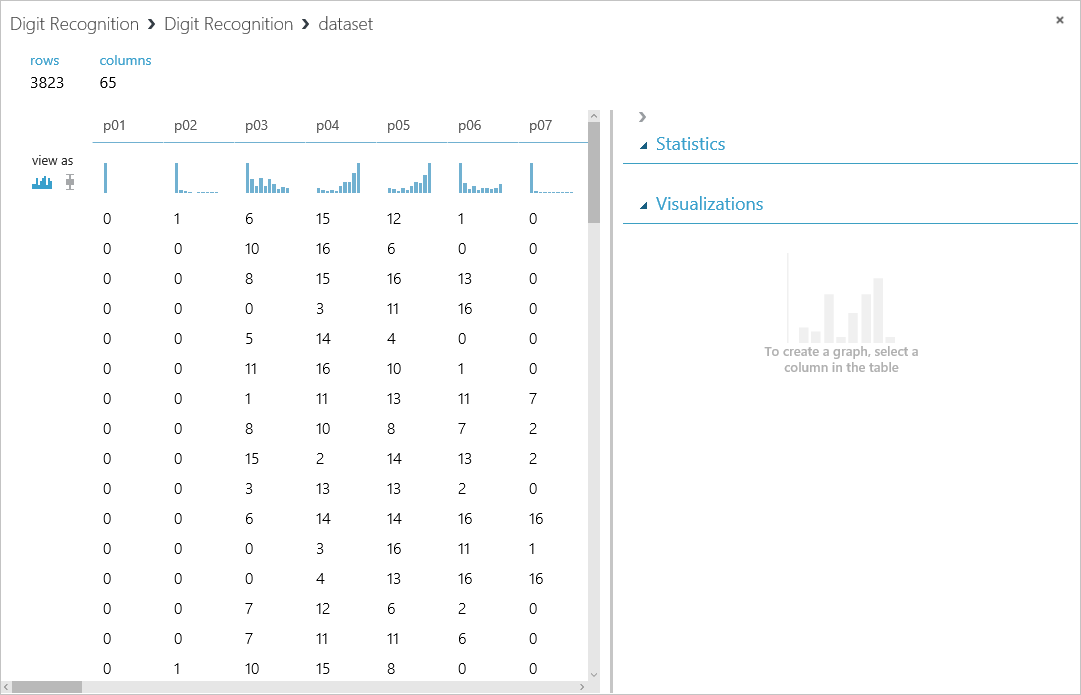
*モデルへのデータセットの追加*

1. データセットがどのようになっているかを確認するために、データセットの下部にある出力ポート (丸の中に 1 があるもの) をクリックして、[Visualize] を選択します。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/visualize-dataset.png)

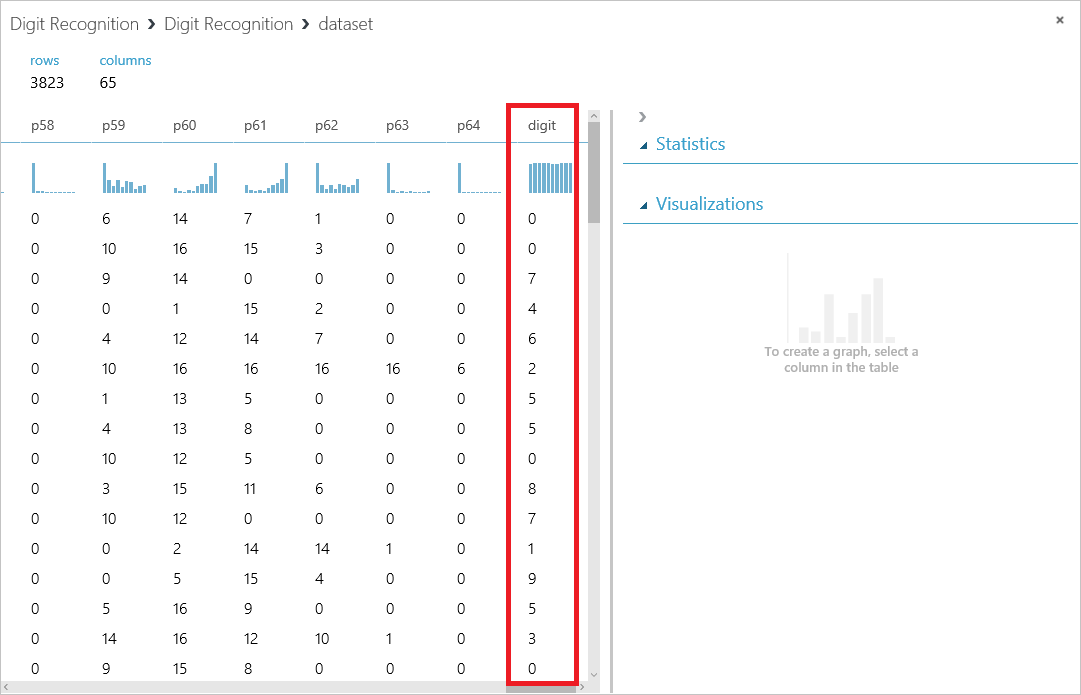
*データセットの視覚化*

1. データセットでは、変数、つまり "フィーチャー" は、列として表示され、各行が、0 ～ 9 の数字を表します。データセットには、3,823 行と 65 列が含まれています。最初の 64 列には、4 × 4 ブロックのピクセルを表す 0 ～ 16 の値が含まれています。データセットに表されている各数字は、32 × 32 配列にスキャンされたもので、1 つの数字あたり、合計 1,024 個のピクセルになります。データセットでは各スキャンは 64 個の値で表され、それぞれの値が 16 ピクセルを表します。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/raw-dataset.png)

*数字認識データセット*

1. "digit" という名前の最後の列が表示されるまで、右にスクロールします。この列内のそれぞれの値が、同じ行内のピクセル データに対応する数字です。これが、ほかの 64 列の値を考慮して、モデルによって予測されるターゲット値です。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/digit-column.png)

*ターゲットの変数*

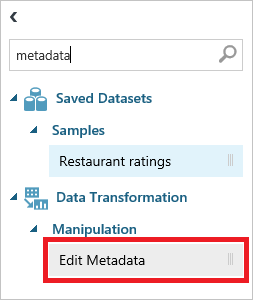
右上隅にある [x] をクリックして、視覚化ウィンドウを終了します。データがロードされます。これを使用して、次の演習を行います。

**演習 3: 分類モデルのトレーニングを行う**

この演習では、ML Studio のドラッグ アンド ドロップ ユーザー インターフェイスを使用して、ML モデルのトレーニングを行います。トレーニングでは、いずれかの機械学習アルゴリズムを選択して、モデルにデータを取り込みます。トレーニングの間に、コンピューターでは、今後の入力からの値を予測するときに使用できるデータのパターンが検索されます。

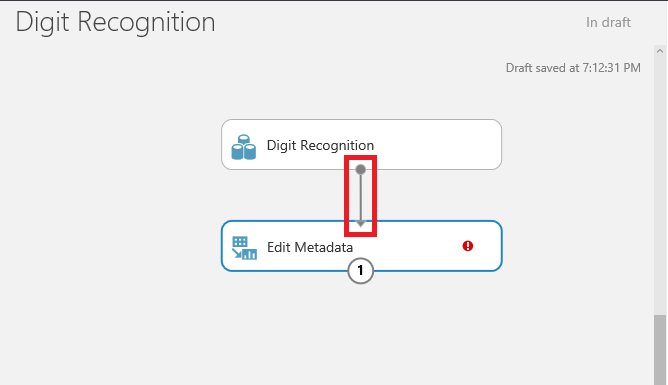
機械学習モデルには、いくつかの種類があります。最も一般的な種類の 1 つが、回帰モデルです。このモデルでは、多数ある回帰アルゴリズムの 1 つを使用して、数値を生成します。たとえば、人の年齢や、クレジットカード トランザクションが不正である確率などです。ここでは、分類モデルのトレーニングを行います。この種類のモデルでは、一連の入力を一連の既知の出力の 1 つに解決できるようにします。分類モデルの典型的な例として、電子メールを調査して、"スパム" または "非スパム" に分類する処理があります。ここでは、モデルを使用して、ピクセル パターンを表す一連の入力を調査し、それぞれを 0 ～ 9 までの数字に分類してみます。

1. モジュール パレットの上部にある検索ボックスに「metadata」と入力して (かぎかっこは付けません)、[[Edit Metadata](https://msdn.microsoft.com/en-us/library/azure/dn905986.aspx)] モジュールを検索します。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/find-metadata-editor.png)

*[Edit Metadata] モジュールの検索*

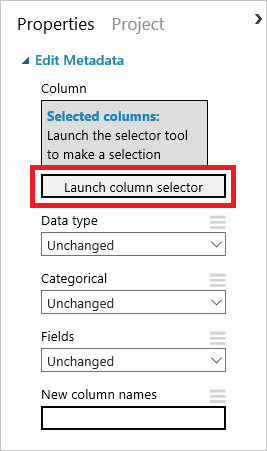
1. このモジュールを実験キャンバスにドラッグして、データセットの下にドロップします。次に、データセットの出力ポートから出ている下向き矢印をドラッグして、出力ポートを [Edit Metadata] モジュールの入力ポートに接続します。[Edit Metadata] モジュールを使用すると、特に、データセット内の列のデータの種類を指定できるようになります。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/metadata-editor.png)

*[Edit Metadata] モジュールの追加*

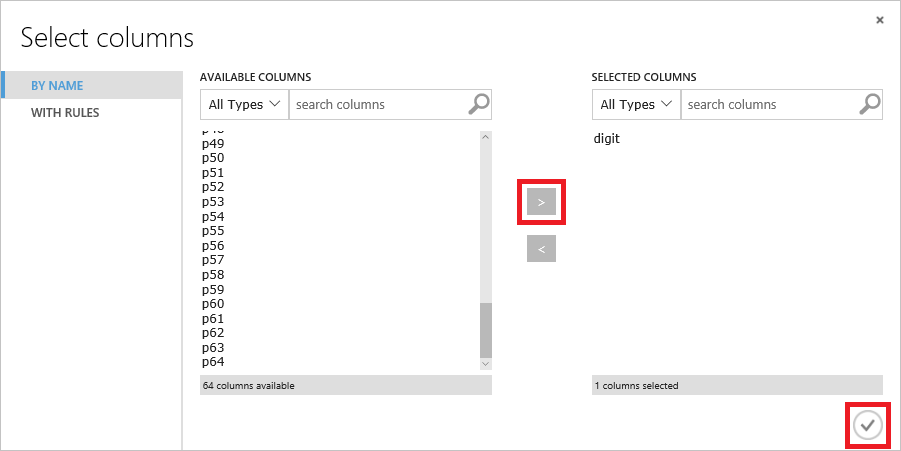
Azure ML Studio で理解しておくべき主要概念の 1 つが、ポートとコネクタの概念です。このステップでは、データセットの出力ポートを [Edit Metadata] モジュールの入力ポートに接続しました。コネクタを介して、1 つのモジュールから次のモジュールにデータが流れます。モジュールによっては、複数の入力および出力をサポートするため、複数の入力ポートと出力ポートを持つものもあります。ポートの機能を知りたい場合は、ポートの上にカーソルを置くと、ヒントがポップアップ表示されます。より詳しい情報が必要な場合は、モジュールを右クリックして、表示されるポップアップ メニューから [Help] を選択します。

1. [Edit Metadata] モジュールをクリックして、選択します (選択すると、ボールドの青い枠で囲まれます)。右側の [Properties] ペインにある [Launch column selector] ボタンをクリックします。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/launch-column-selector-1.png)

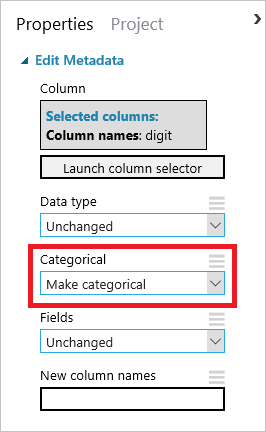
*列セレクターの起動*

1. [AVAILABLE COLUMNS] の一覧の最下部までスクロールして、[digit] をクリックします。[>] ボタンをクリックして、[digit] を [SELECTED COLUMNS] の一覧に移動します。[digit] は、モデルが予測する値であることを思い出してください。ウィンドウの右下隅にあるチェック マークをクリックして、完了します。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/select-digit-column.png)

*[digit] 列の選択*

1. [Properties] ペインで、[Categorical] という名前のドロップダウン リストから [Make Categorical] を選択します。これにより、[digit] 列内の値を離散型 (7.5 や 8.6 などの値をとることができる数値に対立するもの) として扱うようモデルに指示されます。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/make-categorical.png)

*[digit] 列をカテゴリ別として指定*

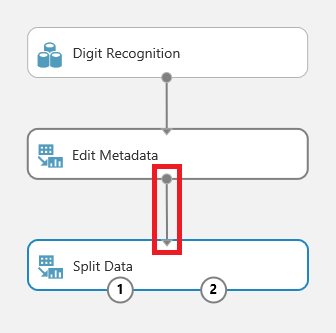
1. ページ下部の [SAVE] ボタンをクリックして、実験を保存します。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/save-button.png)

*実験の保存*

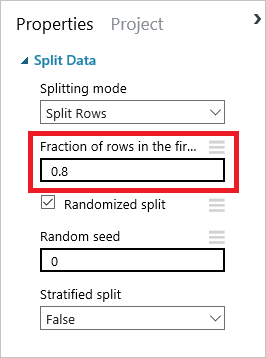
Azure Machine Learning Studio を使用して作業する場合、こまめに実験を保存するようにし、特にモデルを実行する前には忘れずに保存してください。そうすれば、問題が発生した場合でも、作業をやり直す必要がありません。また、**先に実験を保存しないでブラウザーの [Back] ボタンをクリックすると、作業が失われる可能性がある**ことにも注意してください。

1. モジュール パレットの最上部にある検索ボックスに「split」 (かぎかっこは付けません) と入力します。キャンバスに [[Split Data](https://msdn.microsoft.com/en-us/library/azure/dn905969.aspx)] モジュールをドラッグし、[Edit Metadata] モジュールの出力ポートを [Split Data] モジュールの入力ポートに接続します。[Split Data] モジュールの目的は、データセットを 2 つの部分に分割することで、1 つがトレーニング用、もう 1 つがスコア付け用です。[Split Data] モジュールは、トレーニング用とスコア付け用に個別のデータセットがない場合に有用です。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/add-split-data.png)

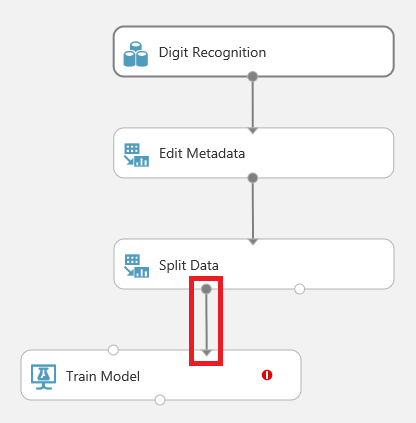
*[Split Data] モジュールの追加*

1. [Split Data] モジュールが選択されていることを確認します。次に、右側の [Properties] ペインに移動して、分割を 0.8 に設定します。行の 80% をモデルのトレーニングに使用し、残りの 20% をモデルのスコア付けに使用します。[Randomized split] チェック ボックスがオンになっていることも確認します。これは、特に、順序付けされているデータを分割するときに重要です。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/setting-split-percentage.png)

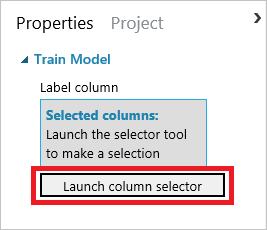
*分割の割合の設定*

1. ページ下部の [SAVE] ボタンをクリックして、実験を保存します。
2. ページ下部の [RUN] ボタンをクリックして、実験を実行します。
3. 次に、モジュール パレットの最上部にある検索ボックスに「train model」 (かぎかっこは付けません) と入力します。[[Train Model](https://msdn.microsoft.com/en-us/library/azure/dn906044.aspx)] モジュールを見つけて、キャンバスにドラッグします。[Split Data] モジュールの左の出力を [Train Model] モジュールの右の入力に接続します。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/add-train-model.png)

*[Train Model] モジュールの追加*

1. 次のステップはきわめて重要です。このステップでは、[Train Model] モジュールで予測することになる値を指定します。[Properties] ペインで [Train Model] モジュールを選択して、[Launch column selector] をクリックします。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/launch-column-selector-2.png)

*列セレクターの起動*

1. 「digit」 (かぎかっこは付けません) と入力して、[digit] 列を選択します。これが、モデルが予測することになる値です。右下隅にあるチェック マークをクリックして、完了します。

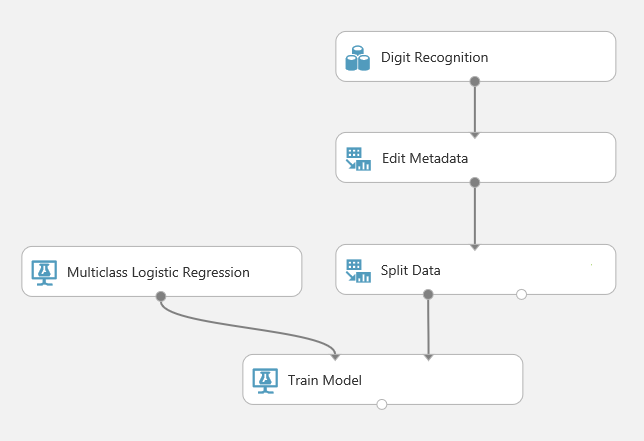
[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/specify-target-value.png)

*ターゲット値の指定*

1. 次に、学習アルゴリズムを選択します。Azure Machine Learning では、数種類の分類アルゴリズムと回帰アルゴリズムが提供されており、そこから選択することができます。また、他の種類のアルゴリズムも提供されています。それぞれのアルゴリズムが、モジュール パレットでモジュールとして表されます。いつでも、R または Python で独自のアルゴリズムをコーディングすることもできます。ここでは、モデルに "Multiclass Logistic Regression" として知られているアルゴリズムを採用します。これは、分類アルゴリズムの 1 つで、結果の確率を予測するために統計でよく使用されているロジスティック回帰が適用されます。

Azure Machine Learning チームは、お客様がモデルの意図した目的に基づいて、どのアルゴリズムを選択すべきか判断できるように、"チート シート" を作成しています。チート シートは、[こちら](http://download.microsoft.com/download/A/6/1/A613E11E-8F9C-424A-B99D-65344785C288/microsoft-machine-learning-algorithm-cheat-sheet-v6.pdf)からダウンロードできます。

モジュール パレットの最上部にある検索ボックスに「multiclass」 (かぎかっこは付けません) と入力します。[[Multiclass Logistic Regression](https://msdn.microsoft.com/en-us/library/azure/dn905853.aspx)] モジュールを見つけて、キャンバスにドラッグします。次に示しているように、[Multiclass Logistic Regression] モジュールの出力を [Train Model] モジュールの左の入力に接続します。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/add-algorithm.png)

*学習アルゴリズムの指定*

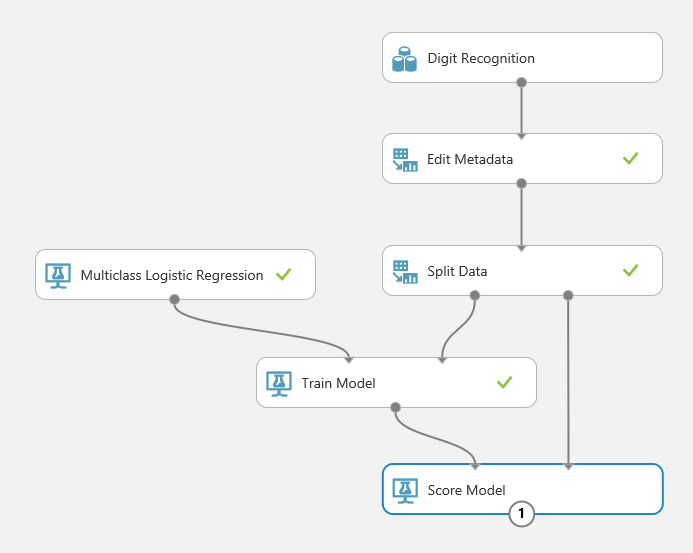
1. [SAVE] ボタンをクリックして、実験を保存します。
2. [RUN] ボタンをクリックして、モデルを実行します。すべてのモジュールに、正常に実行されたことを示す緑色のチェック マークが表示されるまで待ちます。
3. 出力の計算における入力変数それぞれの影響の大きさを確認する場合は、[Train Model] モジュールの出力ポートを右クリックして、[Visualize] を選択します。モデルのフィーチャー (列) と、それに割り当てられている重みの一覧が表示されます。

これでモデルにトレーニングを行いました。ただし、値の予測において、モデルはどれほど正確なのでしょうか。これが、スコア付けを行う理由です。

**演習 4: モデルにスコアを付ける**

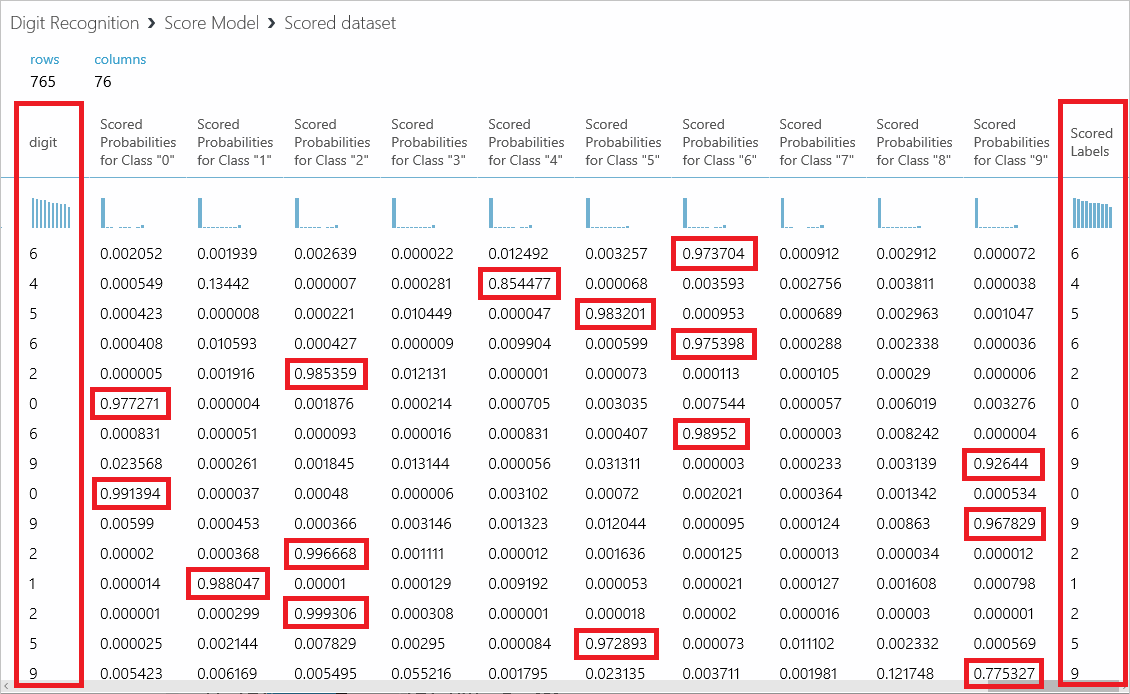
この演習では、前の演習でトレーニングを行ったモデルにスコアを付けます。"スコア付け" によって、モデルのトレーニングがどれほどうまく行われたかが判断されます。つまり、モデルが、提示された変数からターゲット値をどれほど正確に予測するかということです。ML Studio では、スコア付けのプロセスがきわめて簡単になります。アップロードしたデータセットの 80% をモデルのトレーニングに使用しました。残りの 20% をモデルのスコア付けに使用します。

1. キャンバスに [[Score Model](https://msdn.microsoft.com/en-us/library/azure/dn905995.aspx)] モジュールを追加します。[Train Model] モジュールの出力を [Score Model] モジュールの左の入力ポートに接続します。次に、[Split Data] モジュールの右の出力ポートを [Score Model] モジュールの右の入力に接続します。この接続は、トレーニングには使用されなかった 20% のデータを表します。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/add-score-model.png)

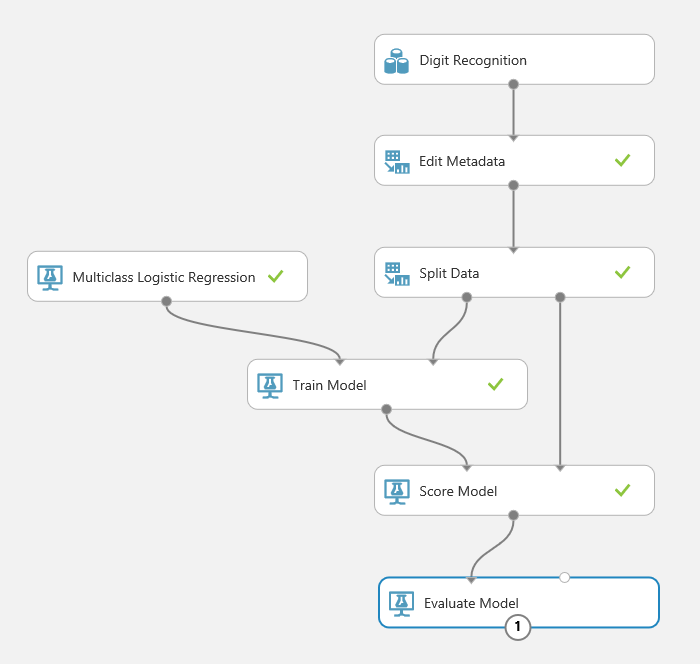
*[Score Model] モジュールの追加*

1. [SAVE] ボタンをクリックして、実験を保存します。
2. [RUN] ボタンをクリックして、実験を実行します。
3. 実行が完了したら、[Score Model] モジュールの出力ポートを右クリックして、[Visualize] を選択します。右にスクロールします。[digit] というラベルが付いている列に、[Score Model] モジュールに渡された 20% のデータセットの実際の値が表示されています。横の、[Scored Probabilities for Class X] という名前の 10 列は、モデルが左の入力値を使用して、可能な 10 個の出力値 (数字 0 ～ 9) をどのようにスコア付けしたかを示しています。数値が高い方が、一致の可能性が高くなります。最終列の [Scored Labels] は、入力値からモデルが予測した数字を示しており、これは、最も高い確率であるとスコア付けされた数字です。[digit] 列の値と [Scored Labels] の値とを比較することによって、モデルの正確性を評価できます。簡単な検査により、モデルがほとんど常に正しく機能していることが明らかになります。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/scored-dataset.png)

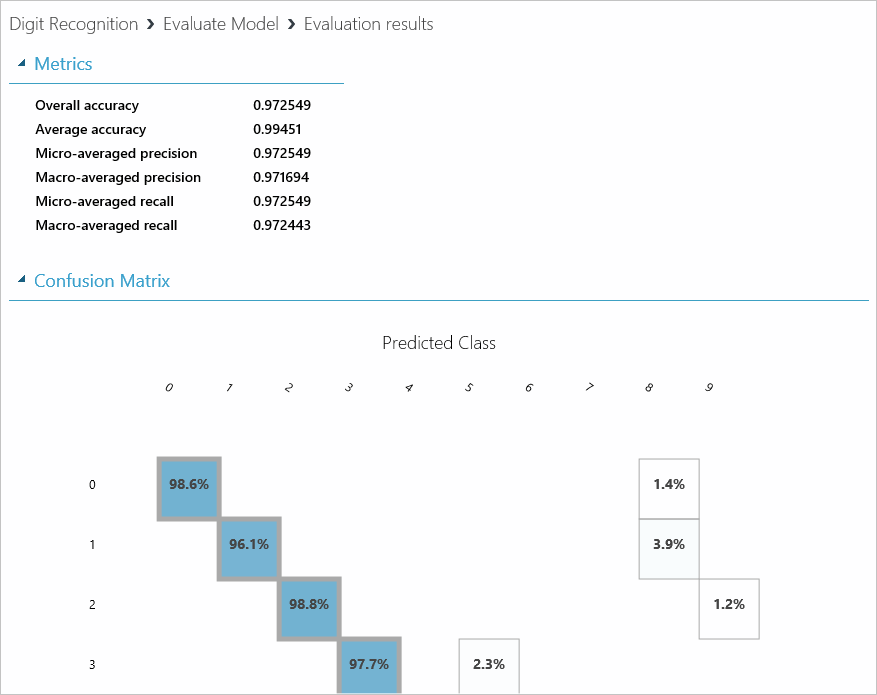
*スコア付けしたデータセット*

1. 右上隅にある [x] をクリックして、視覚化ウィンドウを終了します。
2. 数百もの値を手動で比較する必要はなく、結果の品質を評価するには、[[Evaluate Model](https://msdn.microsoft.com/en-us/library/azure/dn905915.aspx)] モジュールを実験キャンバスに追加します。[Score Model] モジュールの出力ポートを [Evaluate Model] モジュールの左の入力ポートに接続します ([Evaluate Model] は 2 つのモデルの比較に使用することがあるため、入力ポートが 2 つあります)。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/add-evaluate-model.png)

*[Evaluate Model] モジュールの追加*

1. [Save] をクリックして、実験を保存します。[RUN] ボタンをクリックして、もう一度実験を実行します。
2. [Evaluate Model] モジュールの出力ポートをクリックして、メニューから [Visualize] を選択します。[Overall accuracy] および [Average accuracy] の数値から、モデルがうまく機能していることがわかります。手書きの数字のデジタル化スキャンの場合、97% を超える高い確率で、数字を正しく識別できます。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/visualize-evaluate-model.png)

*評価の結果*

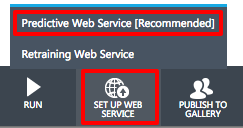
1. 右上隅にある [x] をクリックして、視覚化ウィンドウを終了します。

これで、モデルの構築とテストが完了し、次は実行に移ります。最終的な目標は、PC またはモバイル デバイスで動作し、クラウドまでつながって、モデルを実行するアプリを作成することです。これを簡単にするには、モデルを Web サービスとして展開する必要があります。

**演習 5: モデルを Web サービスとして展開する**

モデルのトレーニングとスコア付けが完了したら、そのモデルを Web サービスとして展開して、プログラムによって操作することができます。Web サービスとして展開する前に、実験を効率化する必要があります。これには、トレーニングを行ったモデルからの新しい実験の作成、不要なモジュールの削除、および Web サービスの入力および出力モジュールの追加が含まれます。幸運にも、ML Studio がこれらすべての処理を実行してくれます。

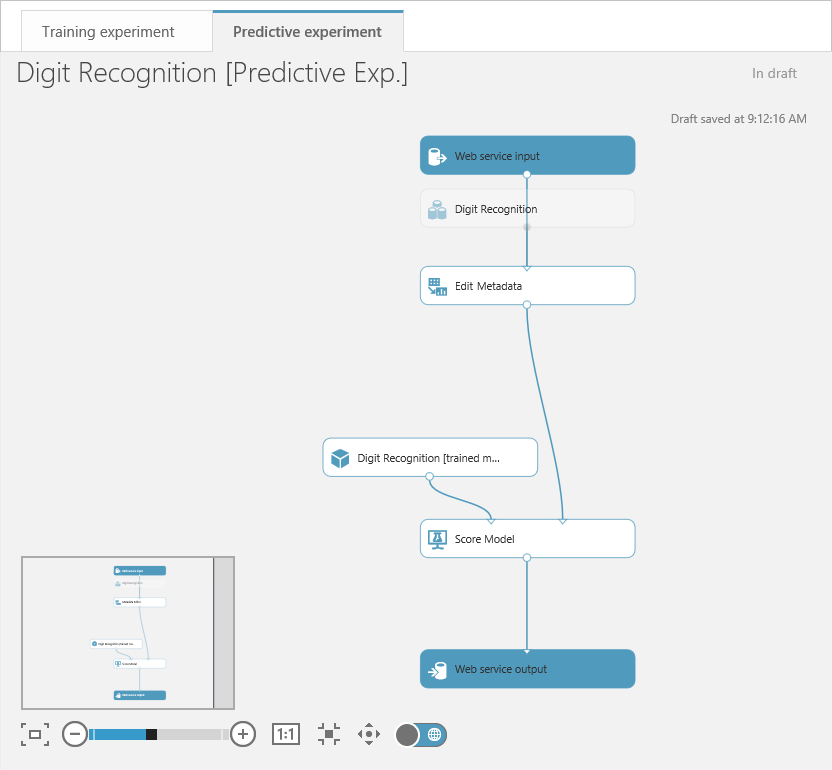
1. [RUN] ボタンをクリックして、モデルをもう一度実行します。
2. 画面の最下部にある [SET UP WEB SERVICE] ボタンをクリックして、表示されるメニューで [Predictive Web Service [Recommended]] を選択します。このオプションが灰色表示になっている場合は、[RUN] ボタンをクリックして再試行します。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/deploy-predict-button.png)

*予測 Web サービスの作成*

1. ML Studio が数秒間処理を行い、その後に予測実験が表示されます。[Split Data] と [Train Model] がなくなって、そのすべてのトレーニング データが [Score Model] に流れています。さらに、最上部と最下部に、Web サービスの入力と出力が追加されています。

元のモデルが消えてしまったと心配する必要はありません。元のモデルは残っています。ページの最上部にある [Training experiment] タブをクリックすると、元のモデルを表示できます。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/predictive-experiment.png)

*予測実験*

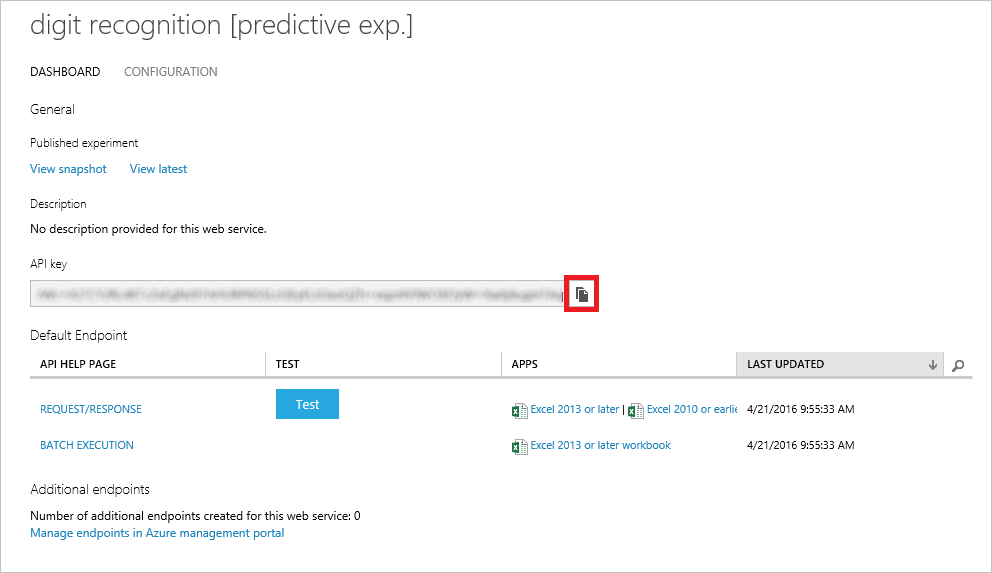
1. 予測分析を実行するために呼び出すことができる Web サービスを作成するには、[RUN] をもう一度クリックします。処理が完了したら、[DEPLOY WEB SERVICE] ボタンをクリックして、Web サービスを展開します。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/deploy-web-service.png)

*Web サービスの展開*

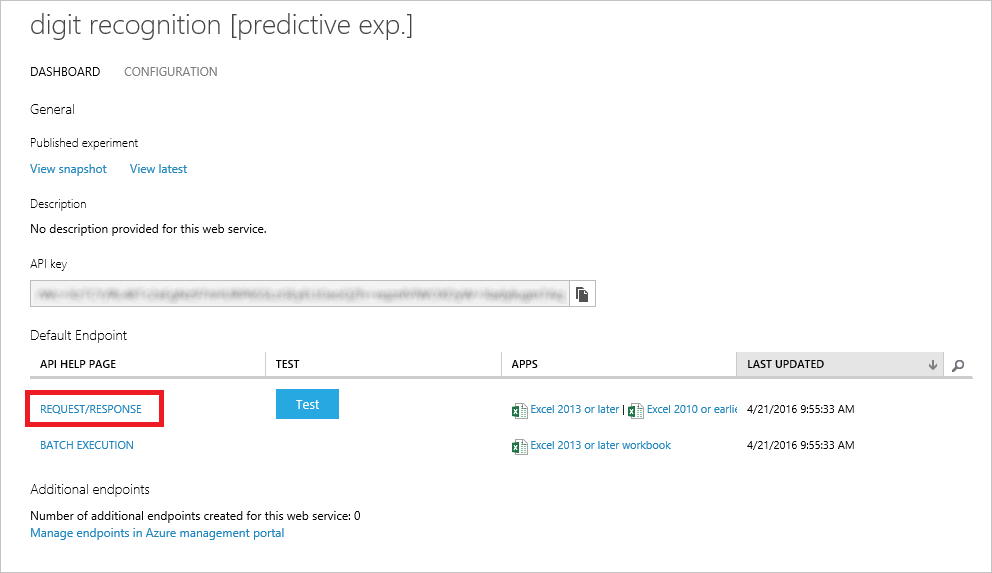
1. すぐに、新しい Web サービスのダッシュボードが表示されます。このダッシュボードには、Web サービスをテストするためにクリックできる [Test] ボタンが含まれています (ここでは、ピクセル パターンを表す 0 ～ 16 の 64 個の値を入力する必要があるため、あまり実用的ではありません)。また、アプリでサービスに対する認証済み呼び出しを行うときに使用できる API キーも含まれています。この API キーは、次の演習で使用します。

便宜上、API の右にあるボタンをクリックして、キーをクリップボードにコピーし、テキスト エディターに貼り付けておくと、後で簡単に取得できます。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/web-service-dashboard-1.png)

*API キーのコピー*

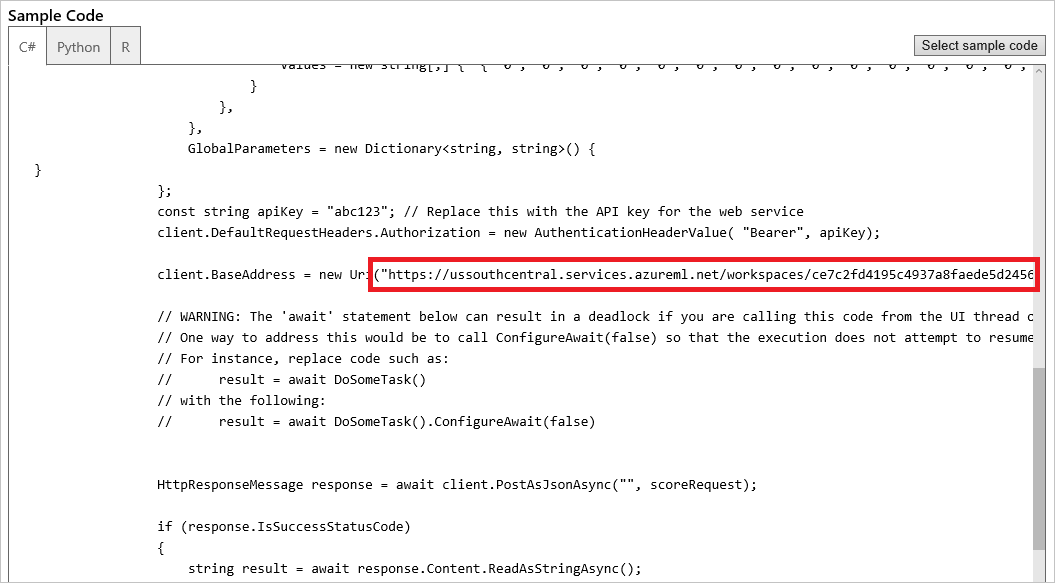
1. [Consume] をクリックして、[REST](https://en.wikipedia.org/wiki/Representational_state_transfer) を使用した HTTPS 経由での Web サービスの呼び出しに関する有用な情報を表示します。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/web-service-dashboard-2.png)

*[Request/Response] の情報の表示*

1. このページには、以下のような、Web サービスを利用するアプリを作成する必要のあるプログラマーに有用なドキュメントが提供されます。
   * HTTPS 要求のフォーマットの方法
   * HTTPS 応答のフォーマットの方法
   * 入力および出力パラメーター
   * Web サービスを呼び出す方法を示したサンプル コード

ページの最下部までスクロールすると、サンプル コードが表示されています。これは、C#、Python、および R の 3 つの言語で利用できます。[C#] を選択して、Web サービスの URL を client.BaseAddress に割り当てるステートメントを確認できるまで、下にスクロールします。次の演習で取得できるように、この URL をクリップボードにコピーして、テキスト エディターに貼り付けておきます。後で、Web サービスを呼び出すために、これと同様のコードを記述します。そのために、この URL が必要になります。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/sample-code.png)

*Web サービスの URL*

Azure ML Web サービスに関してよく聞かれるのが、コストはいくらかという質問です。現在の価格情報については、「[Machine Learning の価格](https://azure.microsoft.com/ja-jp/pricing/details/machine-learning/)」ページを参照してください。

**演習 6: Electron クライアントを構築する**

Azure ML モデルを Web サービスとして展開する理由は、そうすることによってモデルを利用するスマート アプリを構築できることにあります。そのようなアプリを構築する方法はいろいろあります。たとえば、JavaScript や AJAX を使用して Web アプリから Web サービスを呼び出すことができます。あるいは、[NativeScript](https://www.nativescript.org/) を使用して、iOS または Android で動作し、HTTP 要求を使用して Web サービスを呼び出すアプリを作成することもできます。

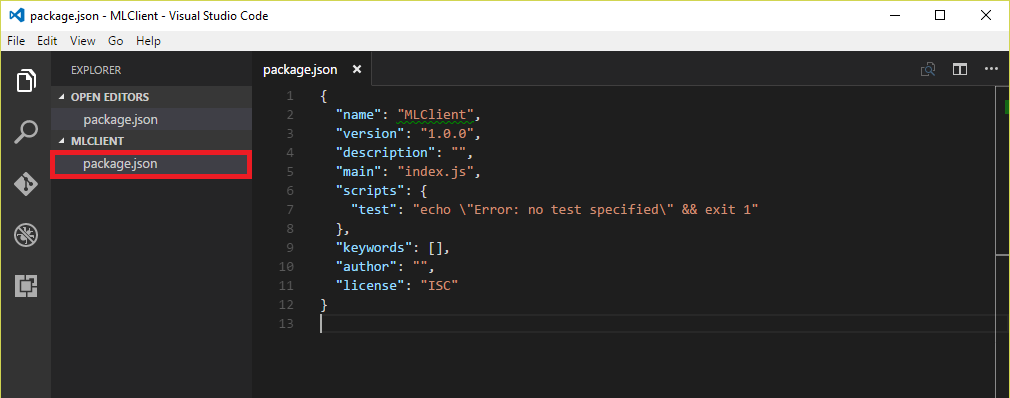
この演習では、[Node.js](https://nodejs.org) および [Electron](http://electron.atom.io/) を使用して、クロスプラットフォーム クライアント アプリを作成します。このようなアプリの利点は、Windows、Mac、または Linux コンピューターで動作することです。作成したアプリを使用すると、画面上のグリッド内に数字を描画できるようになります。そうすれば、アプリによって ML Web サービスが呼び出され、描画した数字が伝えられます。

1. プロジェクト ディレクトリとして機能する新しいディレクトリを作成します。
2. ターミナル ウィンドウまたは [Command Prompt] ウィンドウを開いて、前のステップで作成したディレクトリに移動し、それを現在のディレクトリにします。
3. 次のコマンドを実行して、Node.js バージョン 4 以降がインストールされていることを確認します(Node がインストールされている場合は、バージョン番号が表示されます)。Node がインストールされていない、またはバージョンが 4 より以前の場合は、<https://nodejs.org> にアクセスして、最新の安定バージョンをインストールします。
4. node --version
5. 次のコマンドを使用して、新しい Node.js プロジェクトを作成します。
6. npm init -y

これにより、現在のディレクトリに、空の **package.json** ファイルが作成されます。

Node.js では、npm と呼ばれるパッケージ マネージャーを使用して、Node.js アプリケーションにコンポーネントが追加されます。これらの追加コンポーネントは、アプリケーションの従属項目であり、**package.json** ファイルで定義されます。アプリケーションの構築時にのみ必要な従属項目もあれば、アプリケーションを実際に実行するときに必要な従属項目もあります。今から作成する Electron アプリケーションでは、いくつかの従属項目を使用します。

1. [Command Prompt] ウィンドウまたはターミナル ウィンドウで、以下のコマンドを実行し (コマンド末尾のスペースとピリオドに注意してください)、現在のディレクトリで Visual Studio Code を起動します。
2. code .
3. Visual Studio Code で、[View] メニューから [Explorer] コマンドを選択し、[Explorer] パネルを表示します。**Package.json** をクリックし、ファイルを開いて編集します。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/node-open-package-json.png)

*package.json を開く*

1. 以下に示した dependencies セクションと devDependencies セクションを **package.json** に追加します。**package.json** の 2 行目からこれらの項目を挿入したい場合は、そのようにできます。

{

"dependencies": {

"@angular/common": "2.0.0",

"@angular/compiler": "2.0.0",

"@angular/core": "2.0.0",

"@angular/forms": "2.0.0",

"@angular/http": "2.0.0",

"@angular/platform-browser": "2.0.0",

"@angular/platform-browser-dynamic": "2.0.0",

"@types/core-js": "0.9.34",

"@types/jasmine": "2.2.34",

"@types/node": "6.0.41",

"core-js": "2.4.1",

"reflect-metadata": "0.1.3",

"rxjs": "5.0.0-beta.12",

"systemjs": "0.19.27",

"zone.js": "0.6.23"

},

"devDependencies": {

"electron": "1.4.0",

"npm-run-all": "3.1.0",

"typescript": "2.0.3"

},

...

}

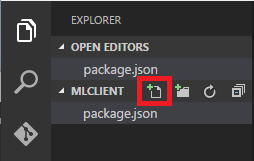
これによって、アプリケーションに多数の重要な従属項目が追加されます。dependencies プロパティの下の項目はすべて、[Angular](https://angular.io/) フレームワークおよび関連する TypeScript の定義に関連付けられます。Angular は、JavaScript アプリケーションを簡単にするクライアント フレームワークです。Angular は、ユーザー インターフェイスの実装および Web サービスの呼び出しに使用する機能を提供します。devDependencies プロパティでは、アプリケーションで使用するその他のコンポーネントが定義されます。

ここでは、アプリケーションのコーディングに、[TypeScript](https://www.typescriptlang.org/) を使用します。TypeScript は、Java Script のスーパーセットであるため、使いやすいはずです。[decorators](https://www.typescriptlang.org/docs/handbook/decorators.html) など、TypeScript 固有の機能を使用します (これは、絶対的な要件ではありませんが、これによって、Angular コードが少し簡単になります)。

1. **package.json** 内の scripts プロパティを以下の scripts プロパティに置き換えます。**package.json** に加えた変更を保存します。
2. {
3. ...
4. "scripts": {
5. "start": "npm-run-all --parallel electron tsc:w",
6. "electron": "electron .",
7. "tsc": "tsc",
8. "tsc:w": "tsc -w"
9. },
10. ...

}

1. Visual Studio Code の [Explorer] パネルで、プロジェクト ディレクトリの上にマウス カーソルを置き、[New File] ボタンをクリックして、プロジェクトに新しいファイルを追加します。ファイルに **tsconfig.json** という名前を付けます。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/node-new-file.png)

*プロジェクトへのファイルの追加*

1. アプリケーションを構築する方法に関する TypeScript 情報を追加するために、以下のステートメントを **tsconfig.json** に貼り付けます。変更内容を保存します。

{

"compilerOptions": {

"target": "es5",

"module": "commonjs",

"moduleResolution": "node",

"sourceMap": true,

"emitDecoratorMetadata": true,

"experimentalDecorators": true,

"removeComments": false,

"noImplicitAny": false

}

}

[Command Prompt] ウィンドウまたはターミナル ウィンドウに戻り、以下のコマンドを実行して、定義した従属項目をインストールします。

npm install

1. 次に、アプリケーション自体を記述します。最初のステップとして、Visual Studio Code を使用して、プロジェクト ディレクトリに **index.html** という名前のファイルを作成し、以下のステートメントを挿入します。

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>DX Machine Learning Lab</title>

<link rel="stylesheet" href="main.css">

<script src="node\_modules/core-js/client/shim.min.js"></script>

<script src="node\_modules/zone.js/dist/zone.js"></script>

<script src="node\_modules/reflect-metadata/Reflect.js"></script>

<script src="node\_modules/systemjs/dist/system.src.js"></script>

<script src="system.config.js"></script>

<script>

System.import('app').catch(function(err){ console.error(err); });

</script>

</head>

<body>

<my-app>Loading ...</my-app>

</body>

</html>

1. 次に、プロジェクト ディレクトリに、**main.css** という名前のファイルを作成して、以下のスタイル定義を追加します。

.pixel-grid {

-webkit-user-select: none;

user-select: none;

cursor: pointer;

}

.pixel-grid > div {

margin: 0;

padding: 0;

display: block;

}

.pixel-grid > div > div {

margin: 0 2px;

padding: 0;

display: inline-block;

width: 30px;

height: 30px;

background-color: #CCCCFF;

}

.pixel-grid div.selected {

background-color: #0000FF;

}

1. 作成するアプリでは、スクリプトのロードに、汎用動的モジュール ローダーの [System.js](https://github.com/systemjs/systemjs) が使用されます。System.js は、JavaScript ファイルのロードに関して構成ファイルに依存します。プロジェクト ディレクトリに **system.config.js** という名前のファイルを作成して、以下のコードを追加することによって、その構成ファイルをプロジェクトに追加します。

(function (global) {

System.config({

paths: {

'npm:': 'node\_modules/'

},

map: {

'app': '.',

'@angular/core': 'npm:@angular/core/bundles/core.umd.js',

'@angular/common': 'npm:@angular/common/bundles/common.umd.js',

'@angular/compiler': 'npm:@angular/compiler/bundles/compiler.umd.js',

'@angular/platform-browser': 'npm:@angular/platform-browser/bundles/platform-browser.umd.js',

'@angular/platform-browser-dynamic': 'npm:@angular/platform-browser-dynamic/bundles/platform-browser-dynamic.umd.js',

'@angular/http': 'npm:@angular/http/bundles/http.umd.js',

'@angular/forms': 'npm:@angular/forms/bundles/forms.umd.js',

'rxjs': 'npm:rxjs'

},

packages: {

app: {

main: './application.js',

defaultExtension: 'js'

},

rxjs: {

defaultExtension: 'js'

}

}

});

})(this);

1. Electron には、メインのスタートアップ ファイルが必要です。スタートアップ ファイルは、アプリケーションの初期構成を実行し、アプリケーションのイベント (ユーザーによるアプリケーションの終了など) に応答し、初期ユーザー インターフェイスをロードします。プロジェクト ディレクトリに **main.ts** という名前のファイルを作成し (拡張子 .ts は TypeScript を表します)、そのファイルに以下のステートメントを追加します。

const {app, BrowserWindow} = require('electron');

let win = null;

app.on('ready', () => {

createWindow();

});

app.on('window-all-closed', () => {

if (process.platform !== 'darwin') {

app.quit();

}

});

app.on('activate', () => {

if (win === null) {

createWindow();

}

});

function createWindow() {

win = new BrowserWindow({width: 800, height: 600});

win.loadURL(`file://${\_\_dirname}/index.html`);

win.on('closed', () => {

win = null

});

//win.webContents.openDevTools();

}

アプリケーションの起動時に [Chrome Developer Tools](https://developer.chrome.com/devtools) が自動的に開くようにしたい場合は、win.webContents.openDevTools() の呼び出しのコメントを外してください。これは、アプリケーションがうまく動作しない場合のデバッグで役立ちます。

1. Electron では、どれがメインのスタートアップ ファイルであるかを認識する必要があります。これは、**package.json** で指定します。このファイル内の main プロパティを以下のように変更します。

"main": "main.js",

**main.js** は、TypeScript ファイル **main.ts** を JavaScript にコンパイルするときに作成されます。

1. 次に、グリッドを表示し、マウスのクリックを処理し、設定した ML Web サービスを呼び出す Angular アプリケーションを使用します。プロジェクト ディレクトリに **application.ts** ファイルを追加して、以下のステートメントを追加します。

import { NgModule, Component, HostListener } from '@angular/core';

import { BrowserModule } from '@angular/platform-browser';

import { HttpModule, Http, Headers, RequestOptions } from '@angular/http';

import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';

import 'rxjs/add/operator/map';

const GridSize: number = 8;

function forEachIndex(count: number, callback: (idx: number) => void) {

for(let idx = 0; idx < count; idx += 1) {

callback(idx);

}

}

function mapByIndex(count: number, callback: (idx: number) => any) {

let items = new Array(count);

forEachIndex(count, idx => {

items[idx] = callback(idx);

});

return items;

}

@Component({

selector: 'pixel-grid',

template: `

<div class="pixel-grid">

<div \*ngFor="let row of rows">

<div \*ngFor="let col of cols"

[ngClass]="{ selected: grid[row\*gridSize+col] }"

(mousedown)="selectPixel(row, col, true)"

(mouseover)="selectPixel(row, col)">

</div>

</div>

</div>

<div>

<button (click)="submitGrid()">Submit</button>

<button (click)="clearGrid()">Clear</button>

</div>

<div \*ngIf="result">

Azure ML says you entered a {{result}}

</div>

<div \*ngIf="requestError"><pre>{{requestError | json}}</pre></div>

`

})

class PixelGrid {

isMouseButtonDown: boolean = false;

gridSize: number = GridSize;

grid: boolean[];

rows: number[];

cols: number[];

result: string = null;

requestError: any = null;

constructor(private http: Http) {

this.rows = new Array(GridSize);

this.cols = new Array(GridSize);

forEachIndex(GridSize, idx => {

this.rows[idx] = idx;

this.cols[idx] = idx;

});

this.clearGrid();

}

@HostListener('document:mousedown', [])

onMouseDown() {

this.isMouseButtonDown = true;

}

@HostListener('document:mouseup', [])

onMouseUp() {

this.isMouseButtonDown = false;

}

selectPixel(row, col, select = false) {

if(this.isMouseButtonDown || select) {

this.grid[row \* GridSize + col] = true;

}

}

clearGrid() {

this.grid = new Array(GridSize \* GridSize);

forEachIndex(GridSize \* GridSize, idx => {

this.grid[idx] = false;

});

this.result = null;

this.requestError = null;

}

submitGrid() {

let columnNames = mapByIndex(GridSize \* GridSize + 1, idx => {

let paramIdx = "0" + (idx + 1);

return "p" + paramIdx.substr(paramIdx.length - 2);

});

columnNames[GridSize \* GridSize] = "digit";

let values = mapByIndex(GridSize \* GridSize + 1, idx => {

return this.grid[idx] ? 16 : 0;

});

values[GridSize \* GridSize] = 0;

let request = {

inputs: {

input1: {

columnNames,

values: [values]

}

},

globalParameters: {

}

};

console.log('-----Request------');

console.log(JSON.stringify(request, null, 2));

this.postRequest(request);

}

postRequest(request: any) {

const url = 'web\_service\_url';

const apiKey = 'api\_key';

let body = JSON.stringify(request);

let headers = new Headers({

'Content-Type': 'application/json',

'Authorization': `Bearer ${apiKey}`

});

let options = new RequestOptions({

headers: headers

});

this.result = null;

this.requestError = null;

this.http

.post(url, body, options)

.map(res => res.json())

.subscribe({

next: (value) => {

console.log('-----Response------');

console.log(JSON.stringify(value, null, 2));

let valuesList = value.Results.output1.value.Values[0];

this.result = valuesList[valuesList.length - 1];

},

error: (error) => {

this.requestError = error.json() || error;

}

});

}

}

@Component({

selector: 'my-app',

template: `

<h2>Draw a digit from 0 to 9</h2>

<pixel-grid></pixel-grid>

`

})

class AppComponent {

}

@NgModule({

imports: [ BrowserModule, HttpModule ],

declarations: [ AppComponent, PixelGrid ],

bootstrap: [ AppComponent ]

})

class AppModule { }

const platform = platformBrowserDynamic();

platform.bootstrapModule(AppModule);

このアプリケーションの核となっているのが、PixelGrid コンポーネントです。これは、ユーザーが数字を描き入れる 2 次元のグリッドを表示して、そのグリッドの各スクエアのオン/オフの状態を示すブール値の配列を管理します。また、[Submit] ボタンと [Clear] ボタンのクリックも処理します。[Submit] ボタンによって、入力データが ML Web サービスに送信されます。

1. アプリケーションには、Web サービスの API キーと URL が必要です。120 行目の web\_service\_url を演習 5 のステップ 7 で保存した Web サービスの URL に置き換え、121 行目の api\_key を演習 5 のステップ 5 で保存した API キーで置き換えることによって、**application.ts** を変更します。
2. const url = 'web\_service\_url';

const apiKey = 'api\_key';

1. Visual Studio Code の [File] の [Save All] コマンドを使用して、加えた変更をすべて保存します。[Command Prompt] ウィンドウまたはターミナル ウィンドウに戻り、以下のコマンドを実行して、アプリケーションをコンパイルします。
2. npm run tsc

おつかれさまでした。Azure Machine Learning を利用してインテリジェンスを実現する Electron アプリケーションの完成です。最後のタスクでは、アプリケーションを実際に実行して、高度な処理能力を確認します。

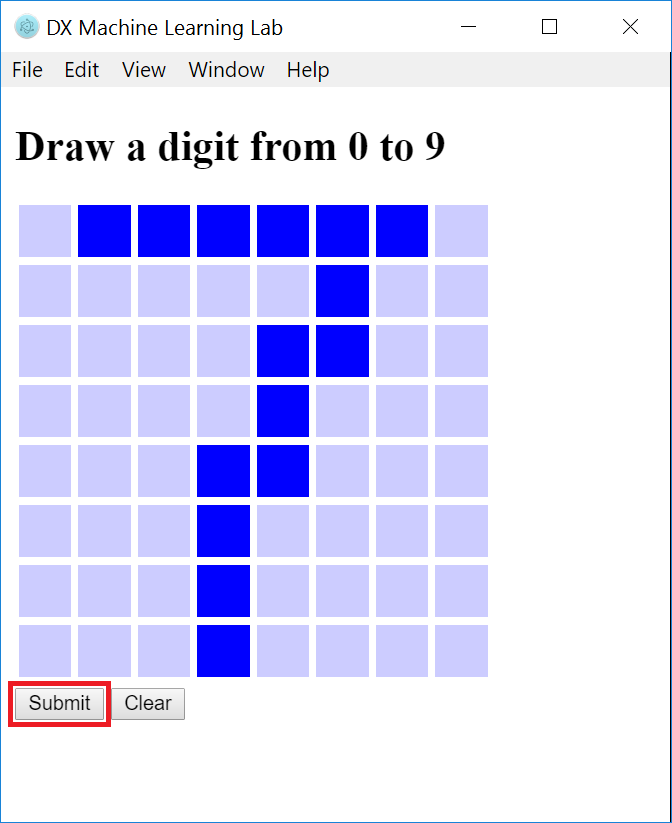
**演習 7: モデルをテストする**

Electron アプリケーションによって、Web サービスの呼び出しにグラフィカル フロントエンドが追加されます。グリッドのスクエアの上にマウスをドラッグすることによって、グリッド内に数字を描画します。[Submit] ボタンをクリックすると、64 個の値の配列が作成され (1 つのスクエアに 1 つの値、トレーニング データベースの 64 のフィーチャー列それぞれに 1 つの値)、その配列が JSON にシリアル化されて、Web サービスに渡されます。戻された JSON がアプリによって逆シリアル化され、結果が表示されます。基本的に Electron クライアントは、ML モデルの正確性を測るための高度に視覚的な方法を提供します。

1. [Command Prompt] ウィンドウまたはターミナル ウィンドウで以下のコマンドを実行して、アプリケーションを実行します。
2. npm run electron

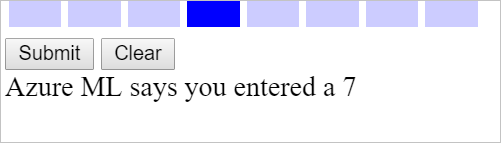
別の方法として、**npm start** を実行できます。これによって、アプリケーションが起動し、TypeScript ファイル内の変更が確認され、変更されている場合はそれらのファイルが再コンパイルされます。このため、手動による再起動をしなくても、コードに加えた変更が実行中のアプリケーションに反映されます。

1. 以下の図のように、マウスを使用してグリッドに「7」と描画します。その後、[Submit] ボタンをクリックします。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/node-mlclient-app-2.png)

*アプリのテスト*

1. すぐに、ウィンドウの最下部にメッセージが表示され、描画した数字が示されます。数字は合っていますか。

[](https://github.com/MSFTImagine/computerscience/blob/master/Workshop/4.Machine%20Learning/Images/node-it-works.png)

*Azure ML の答え*

1. [Clear] ボタンをクリックして、グリッドをクリアし、ほかの数字をいくつか試してみます。数字によって、構築したモデルでうまく識別されるものとそうでないものがあり、描かれた数字がグリッドを埋め尽くす面積が大きいほど結果がよいことがわかるでしょう。

間違った答えは、モデルのトレーニングに使用したデータセットが比較的小規模であった結果というものもあれば (データセットは約 0.5 MB でしたが、ビッグデータの標準としては、小規模です)、画面上のグリッドの表示では、モデルのトレーニングに使用したスキャンの解像度の 1/16 しか使用されないという事実によるものもあります。それでもなお、アプリでこのような方法で基本的な OCR を実行できるというのは、快挙ともいえるでしょう。また、手間のかかる作業を Azure Machine Learning に肩代わりさせることでこのようなアプリを構築できるということもおわかりいただけると思います。

**まとめ**

このハンズオン ラボでは、以下の方法について学習しました。

* Azure Machine Learning の実験を作成する
* データセットをアップロードする
* Azure Machine Learning モデルのトレーニングとスコア付けを行う
* モデルを Web サービスとして展開する
* REST を使用して Web サービスを呼び出す

Azure Machine Learning でできることはほかにもたくさんあり、今回学習した内容は出発点にすぎません。Azure Machine Learning を使ってあれこれ試しながら、生産的であるだけでなく、楽しくもあるツールで予測分析のわくわくする世界を探索してください。

Copyright 2016 Microsoft Corporation. All rights reserved. 特に注記がない限り、これらの資料は MIT ライセンスの使用条件下でライセンス供与されます。これらの資料は、MIT ライセンスに従い、プロジェクトに最も適切と判断された場合は使用することができます。このライセンスの使用条件については、<https://opensource.org/licenses/MIT> を参照してください。