

Azure Machine Learning を活用した 手書き文字認識アプリ(ユニバーサル Windows 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 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 および Python 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 ユニバーサル Windows プラットフォーム (UMP) 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、および評価を行う。
- 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 ラボを行うには、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、無料試用版にサインアップしてください。
- Windows 10 SDK がインストールされている Visual Studio 2017 Community エディションまたはこれ以降。

目次

| | |
|---|----|
| 演習 1: Azure Machine Learning Studio ワークスペース を作成する | 5 |
| • ML Studio ワークスペースの作成 | 6 |
| • ML Studio の起動 | 8 |
| • 新規 Experiment の作成 | 10 |
| 演習 2: データセットをアップロードする | 12 |
| • データセットの新規作成とアップロード | 12 |
| • モデルへのデータセットの追加 | 13 |
| • データセットの確認 | 14 |
| 演習 3: 分類モデルを用いたトレーニングを行う | 16 |
| • [Edit Metadata] モジュールによるデータの抽出と編集 | 16 |
| • [Split Data] モジュールによるトレーニング用、評価用データの分離 | 19 |
| • [Train Model] モジュールによる学習アルゴリズムの設定 | 21 |
| 演習 4: モデルを評価する | 25 |
| • [Score Model] モジュールによる評価用データによる実行 | 25 |
| • [Evaluate Model] モジュールによるモデルの評価 | 27 |
| 演習 5: モデルを Web サービスとして展開する | 30 |
| • Predictive Experiment の作成 | 30 |
| • Web サービスの展開 | 31 |
| • Azure ML Web Services の REST API 情報の表示 | 32 |
| 演習 6: ユニバーサル Windows アプリケーションを構築する | 34 |
| • Windows 10 での開発者モードの有効化 | 34 |

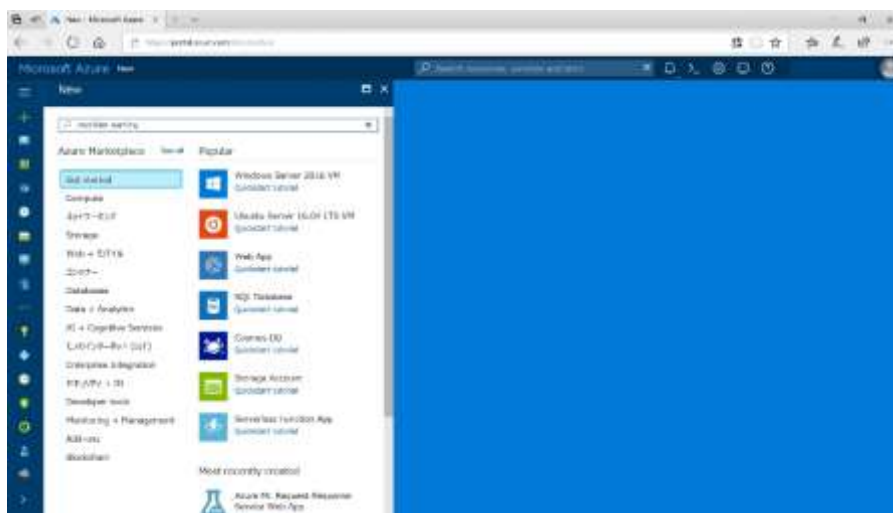
| | |
|--|----|
| • 新しい UWP プロジェクトの作成 | 35 |
| • NuGet によるパッケージの管理と Web API クライアントのインストール | 36 |
| • UI (MainPage.xaml) のコーディング | 37 |
| • 実行部分 (MainPage.xaml.cs) のコーディング | 38 |
| • アプリの動作確認 | 45 |
| まとめ | 47 |

このラボの推定所要時間: 60 分

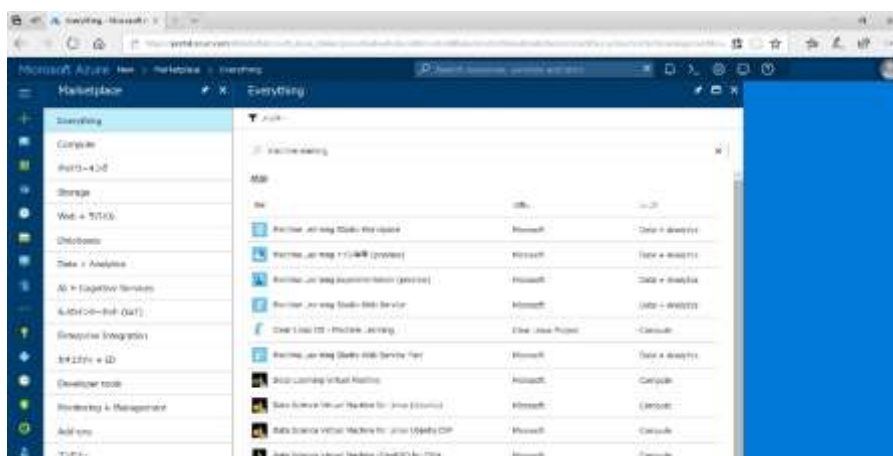
演習 1: Azure Machine Learning Studio ワークスペース を作成する

Azure Machine Learning (Azure ML) を利用する最初のステップは、Machine Learning のワークスペース (ML Studio ワークスペース) と、そこで行う Experiment (開発する機械学習のモデル) の作成です。この演習では、Azure ポータルから Azure Machine Learning Studio (ML Studio) のワークスペースを作成し、Machine Learning の Experiment を作成します。

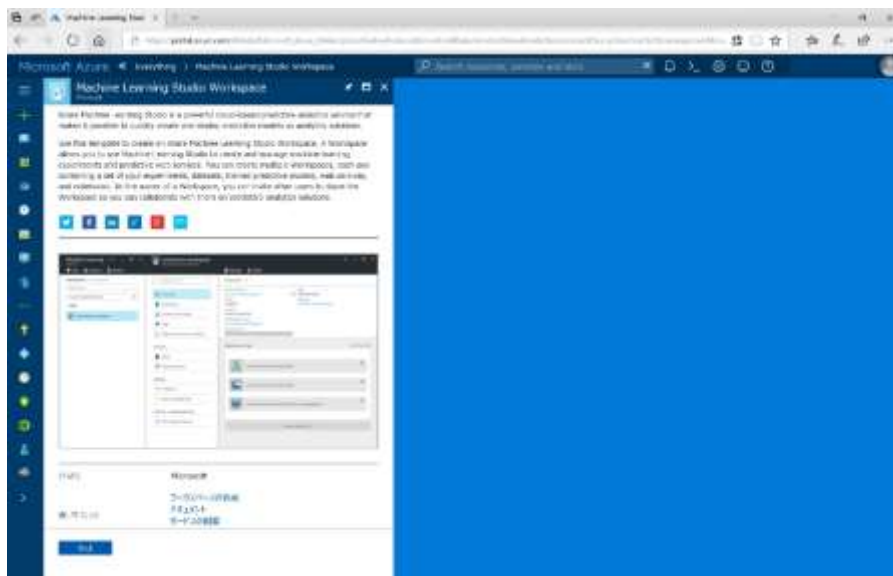
1. ブラウザーで Azure ポータル (<https://portal.azure.com>) を開きます。ログインするよう求められたら、Microsoft アカウントを使用してログインします。
2. **[+ New]** をクリックし、検索ボックスに **Machine Learning** と入力して検索します。



3. Machine Learning Studio Workspace をクリックします。

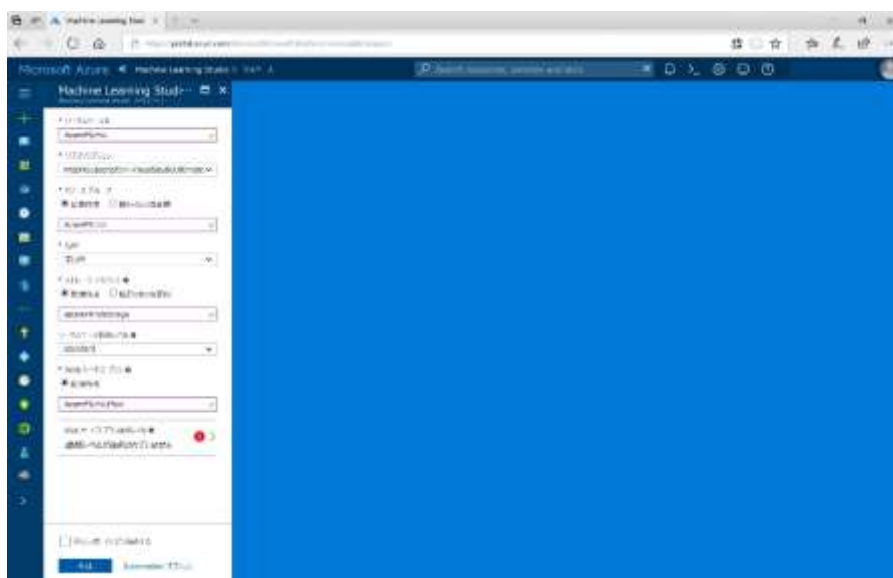


4. [Machine Learning Studio Workspace] ブレードで左下の【作成】をクリックして、ワークスペースを作成します。



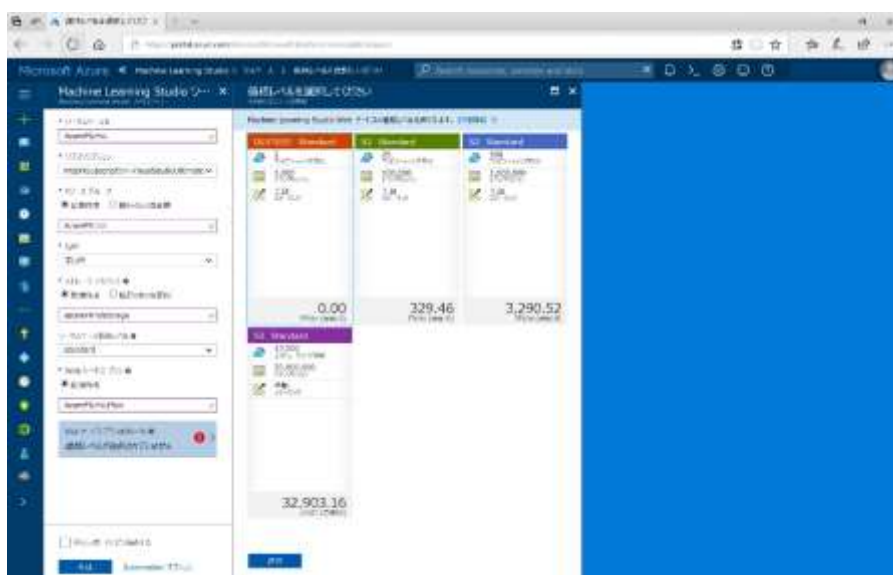
ML Studio ワークスペースの作成

5. [Machine Learning Workspace] ブレードで、**AzureMLHOL** などお好み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。[リソースグループ] は【新規作成】を選択し、お好みのリソース グループ名を入力します。（または既存のリソースグループをご利用いただいても構いません。）[場所] は【東日本】、またはお好みのデータセンターを選択します。次に、[ストレージ アカウント] は【新規作成】でデフォルトで生成される名前、またはお好みのストレージ アカウント名を入力します。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。[Web サービスプラン] はデフォルトで生成された名前、またはお好みのプラン名を入力し、緑色のチェック マーク ✓ が表示されることを確認します。



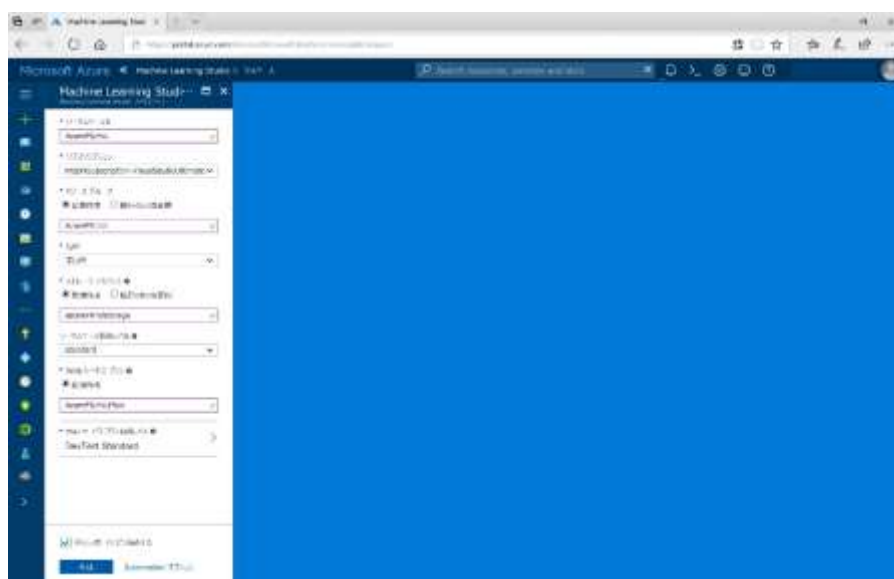
ストレージ アカウント名は、3 ～ 24 文字の長さで、数字と小文字のみを使用でき、Azure 内で一意である必要があります。名前の横の緑色のチェック マークは、それぞれのサービスの命名規則を満たしていることを表します。

6. **[Web サービスプラン価格レベル]** をクリックします。表示される価格レベルから **[DEVTEST Standard]** をクリックし、**[選択]** をクリックします。



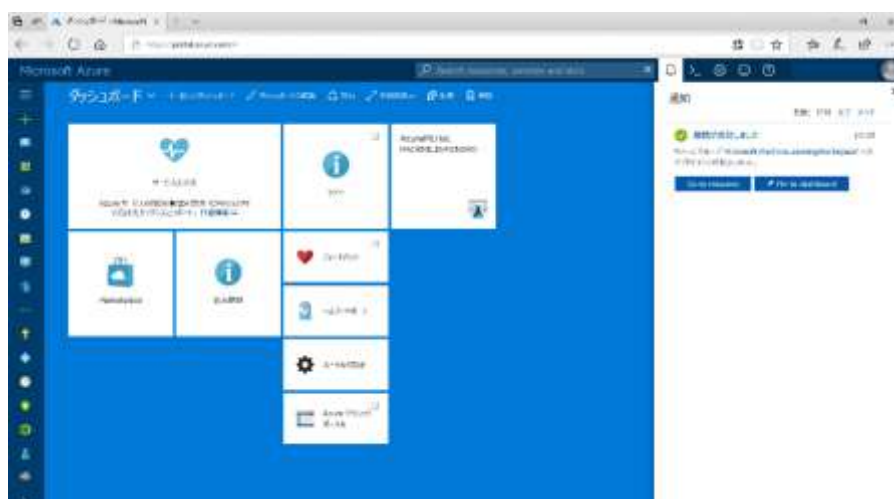
7. **[Machine Learning Workspace]** ブレードの下部にある **[ダッシュボードにピン留めする]** にチェックをつけ、**[作成]** ボタンをクリックして ML ワークスペースの作成を

完了します。



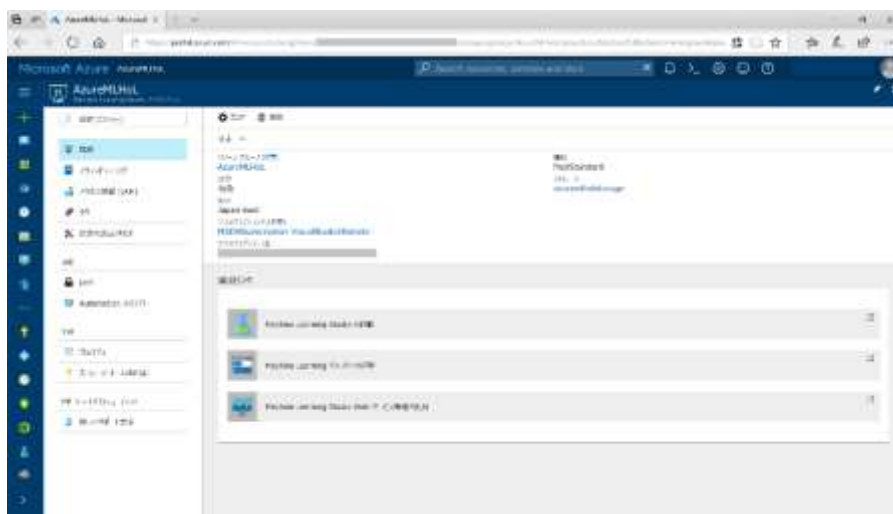
ML Studio の起動

8. 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。[展開が成功しました] と通知が表示されたら、[Go To resource] をクリックして、ML Studio ワークスペースを表示します。



作成した ML Studio ワークスペース画面が自動的に表示されない場合は、ダッシュボードに表示される ML Studio ワークスペースのタイルをクリックして表示します。

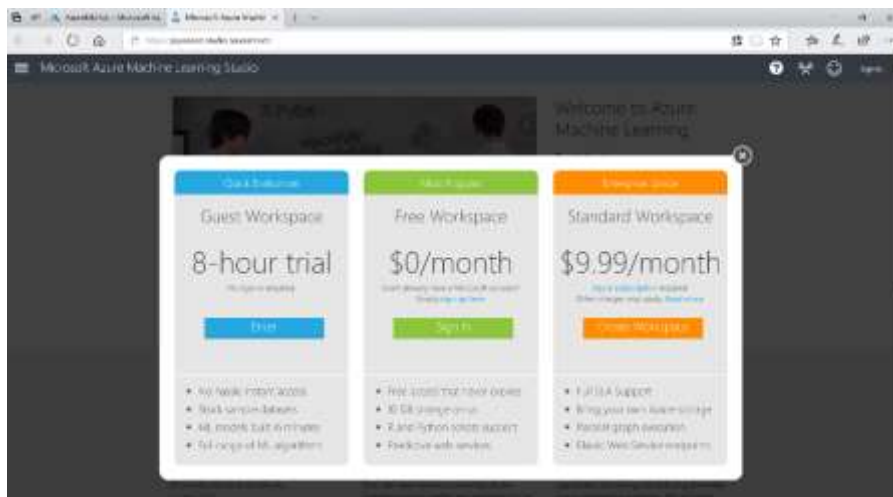
9. [Machine Learning Studio の起動] をクリックします。



10. ブラウザーの新しいタブが開き、Azure Machine Learning Studio (ML Studio) のページが表示されます。[Sign Up] 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。



11. 利用プランは、**Free Plan(\$0)/Month** を利用します。[Sign-in] をクリックします。

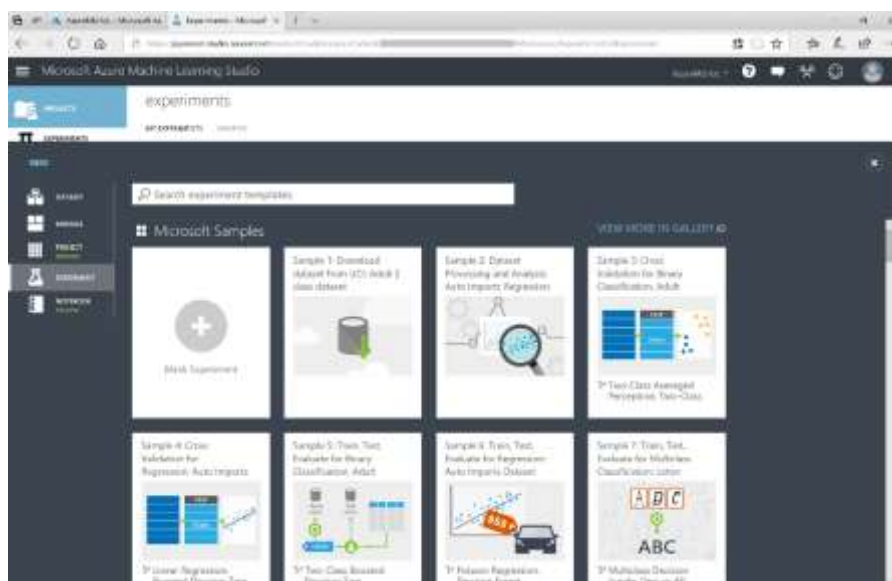


新規 Experiment の作成

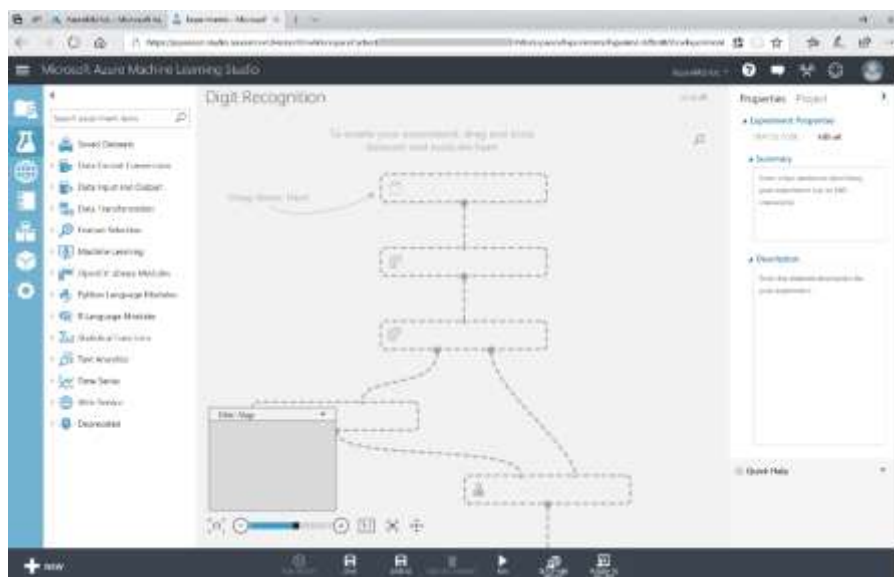
12. ML Studio で、左下隅にある **[+ NEW]** をクリックします。



13. [Blank Experiment] をクリックして、新しい Experiment を作成します。



14. Experiment の構成ページが表示されます。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** などの新しい名称を入力します。



これで、Experiment 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、Kaggle、カリフォルニア大学アーバイン校の Machine Learning Repository 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。

データセットの新規作成とアップロード

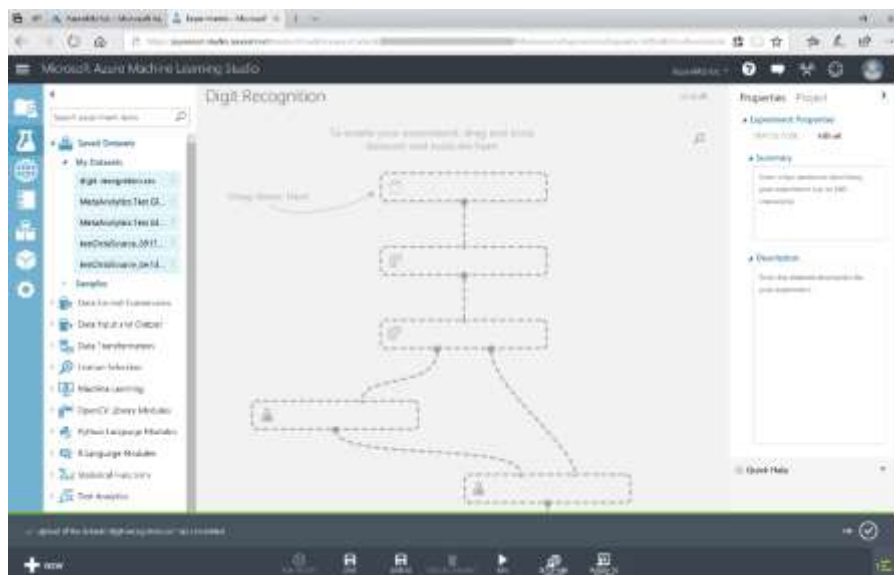
1. ML Studio の左下隅にある **[+ NEW]** をクリックします。**[DATASET]**、**[FROM LOCAL FILE]** の順にクリックします。



2. [Upload a new dataset] の画面で、**[ファイルを選択]** ボタンをクリックします。このハンズオントレーニングのディレクトリ (<https://github.com/ayako/TS17-AzureMLHol>) に移動して、**digit-recognition.csv** という名前のファイルをダウンロードします。[SELECT A TYPE FOR THE NEW DATASET] で [Generic CSV File with a header (.csv)] が自動で選択されることを確認します。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。

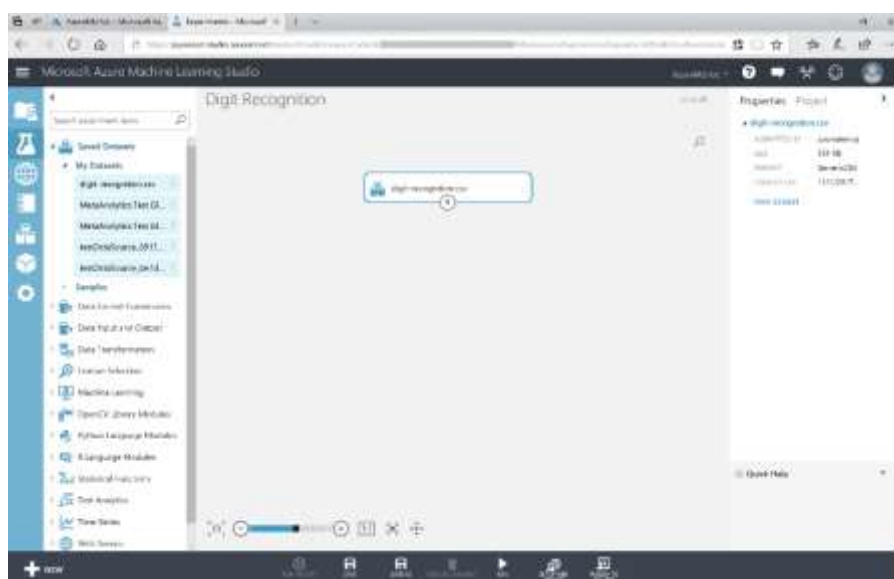


3. アップロードが完了するのを待ちます。完了すると、下部バーに完了メッセージが表示されます。左バーに表示されているモジュール パレットを参照し、[Saved Datasets] > [My Datasets] の下に、アップロードしたデータセット (digit-recognition.csv) が表示されていることを確認します。

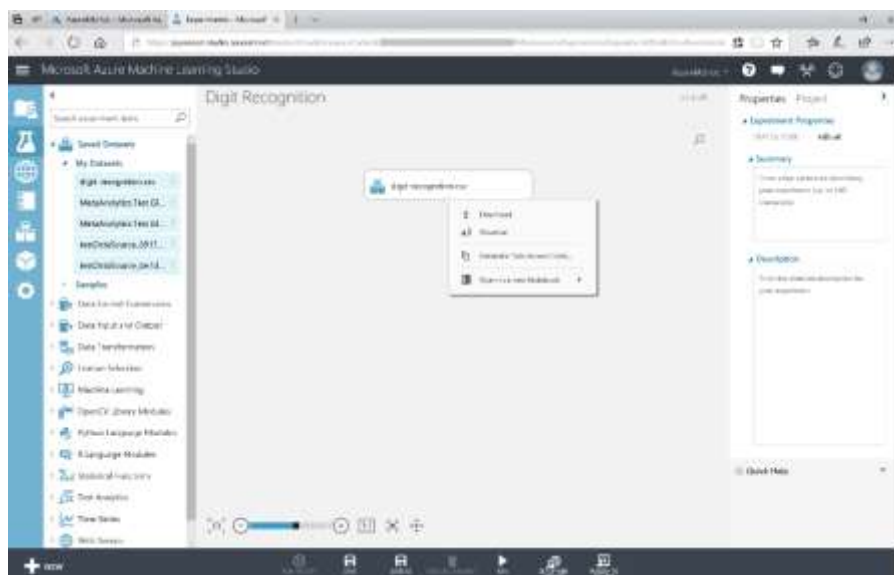


モデルへのデータセットの追加

4. モジュール パレットからキャンバス (中央の灰色の領域) に、データセット (digit-recognition.csv) をドラッグ アンド ドロップします。



5. データセットがどのような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。



データセットの確認

6. このデータセットでは、各行が、0 ～ 9 の数字を表し、画像データを読み取った変数（フィーチャー）が列として表示されます。データセットには、3,823 行(数字の文字数)と 65 列(画像データを数値化したデータ)が含まれています。この p01~64 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。画像データは 32 × 32 配列にスキャンされており、1 つの画像（数字）あたり、合計 1,024 個のピクセルデータを持つことになります。データセットでは各スキャンデータは 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。



- [illegible]

- 15 -

演習 3: 分類モデルを用いた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構成します。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。トレーニングを行うことで、コンピュータでは、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認識されます。

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、予測数値を生成します。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。

今回の演習では、分類モデルによるトレーニングを行います。この種類のモデルでは、入力を、一連の定められた出力のいずれかの 1 つの答に分類します。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。この分類モデルを使用して、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 までの数字に分類を行います。

[Edit Metadata] モジュールによるデータの抽出と編集

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、[Edit Metadata] モジュールを検索します。



2. このモジュールを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、データセットに含まれるデータ列の選択、および種類の指定ができるようになります。



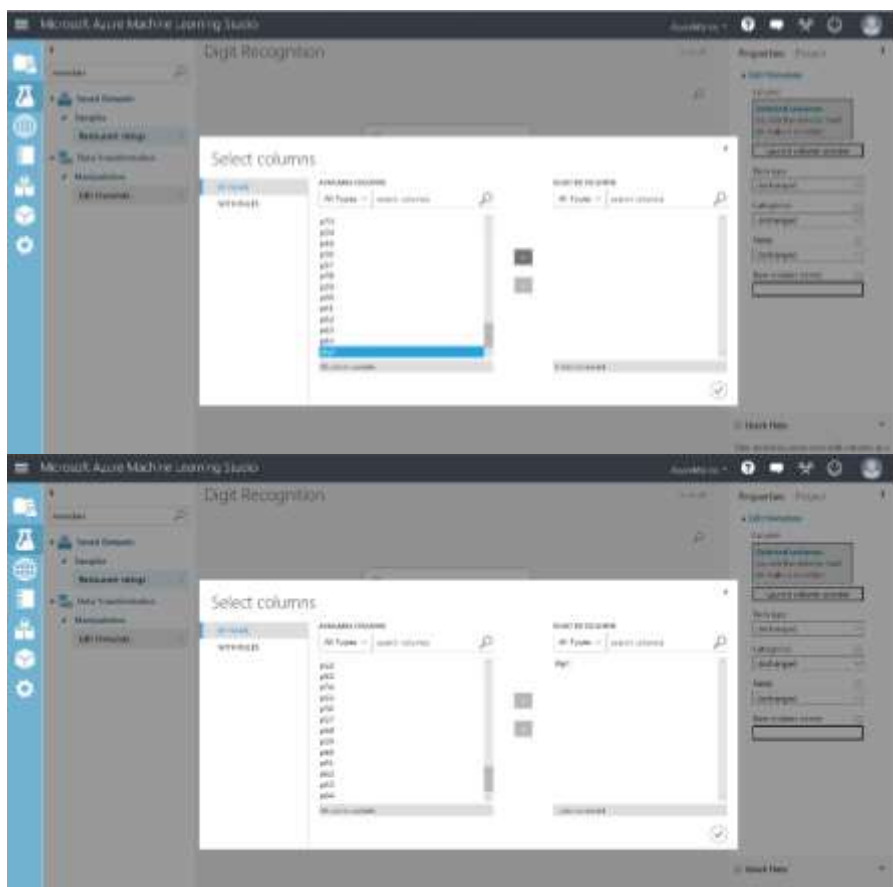
Azure ML Studio の各モジュールにはポート(出力と入力)とコネクタ(データ接続)が存在します。このステップでは、データセットの出力ポートを [Edit Metadata] モジュールの入力ポートに接続しました。コネクタを介して、1 つのモジュールから次のモジュールにデータが流れます。モジュールによっては、複数の入力および出力をサポートするため、複数の入力ポートと出力ポートを持つものもあります。ポートの機能を知りたい場合は、ポートの上にカーソルを置くと、ヒントがポップアップ表示されます。より詳しい情報が必要な場合は、モジュールを右クリックして、表示されるポップアップメニューから [Help]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

3. **[Edit Metadata]** モジュールをクリックして、選択します。(選択したモジュールは枠が青色になります。) 右側の **[Properties]** ペインにある **[Launch column selector]** ボタンをクリックします。

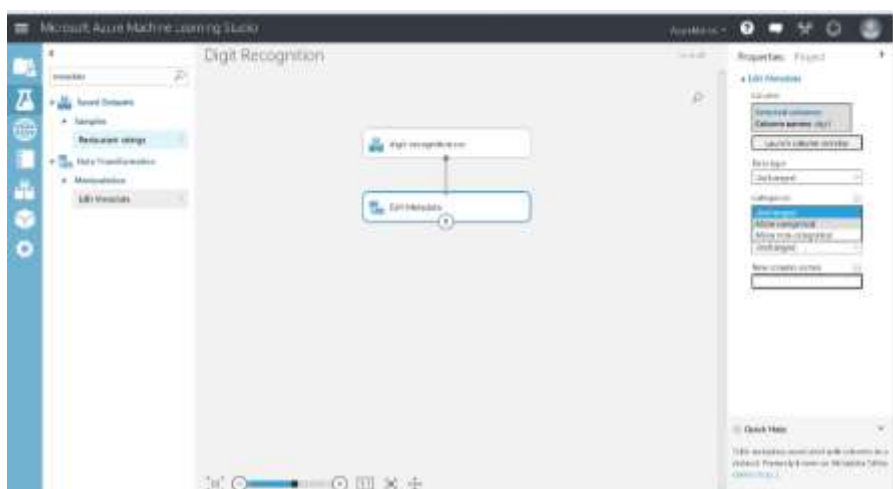


4. **[AVAILABLE COLUMNS]** の一覧の最下部までスクロールして、**[digit]** をクリックします。**[>]** ボタンをクリックして、**[digit]** を **[SELECTED COLUMNS]** の一覧に移動します。**[digit]** は、モデルが予測する値であることを思い出してください。ウィンドウ

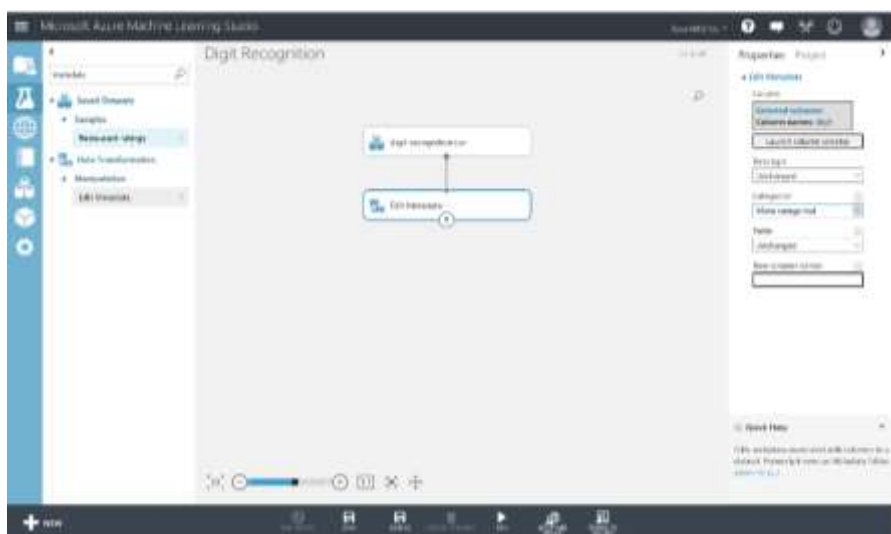
の右下隅にあるチェック マークをクリックして、完了します。



5. [Edit Metadata] モジュールを選択したまま、[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] 列内の値を離散型変数 (7.5 や 8.6 などの中間値をとることができない、非連続の定まった数値) として扱うようモデルに指示されます。



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



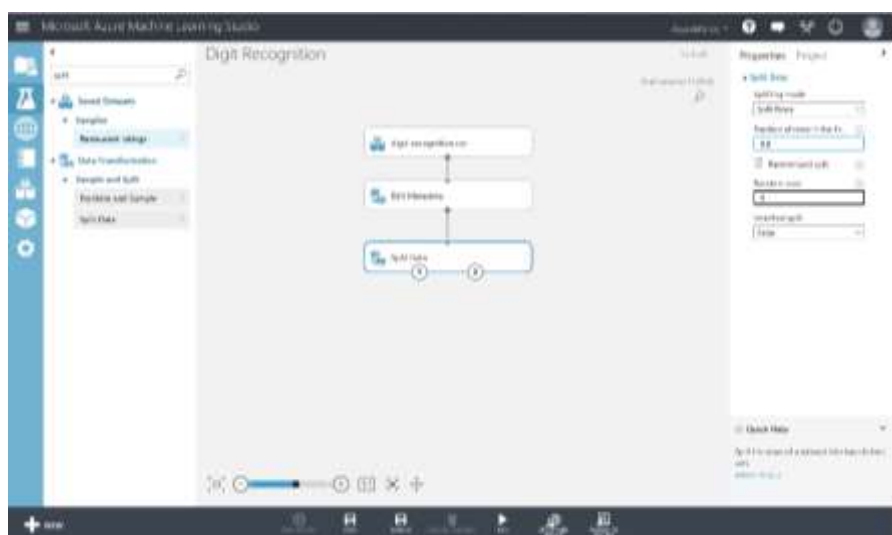
Azure Machine Learning Studio を使用して作業する場合、こまめに Experiment を保存するようにし、特にモデルを実行する前には忘れずに保存してください。また、Experiment を保存する前にブラウザーの [Back] ボタンをクリックすると、作業が失われる可能性がありますのでご注意ください。

[Split Data] モジュールによるトレーニング用、評価用データの分離

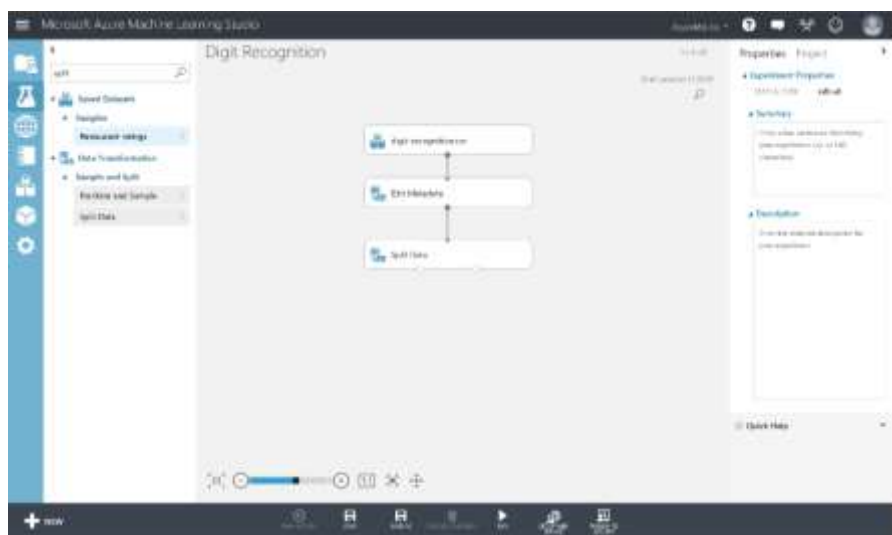
7. モジュール パレットの検索ボックスに **split** と入力します。キャンバスに [Split Data] モジュールをドラッグし、**[Edit Metadata]** モジュールの出力ポート を **[Split Data]** モジュールの入力ポート に接続します。[Split Data] モジュールを利用して、データセットをトレーニング用とテスト用(トレーニング後にモデルのテスト、評価を行うためのデータ)に分離します。[Split Data] モジュールは、トレーニング用と評価用に個別のデータセットがない場合に有効です。



8. **[Split Data]** モジュールを選択し、右側の **[Properties]** ペインで、**Fraction** (割合) 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% をテストに使用します。**[Randomized split]** のチェック 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になります。



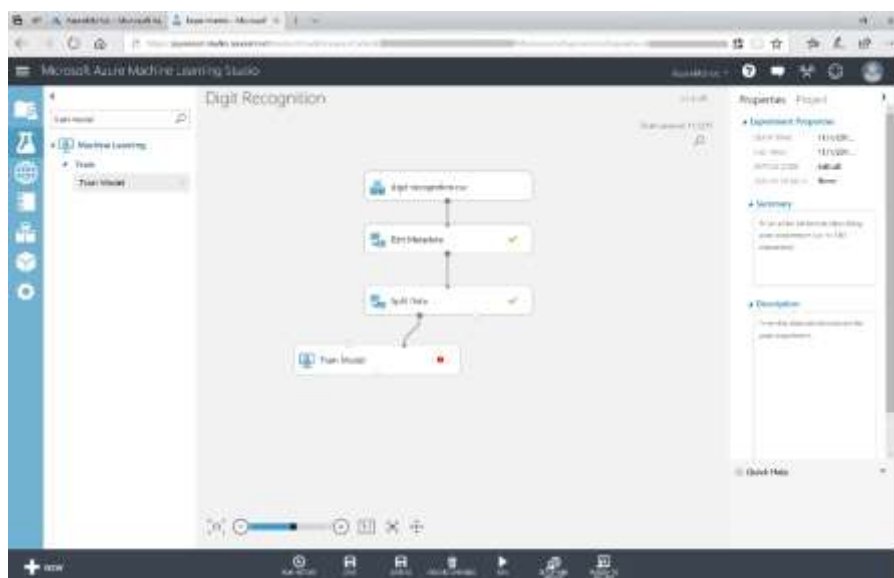
9. ページ下部の **[SAVE]** ボタンをクリックして、Experiment を保存します。



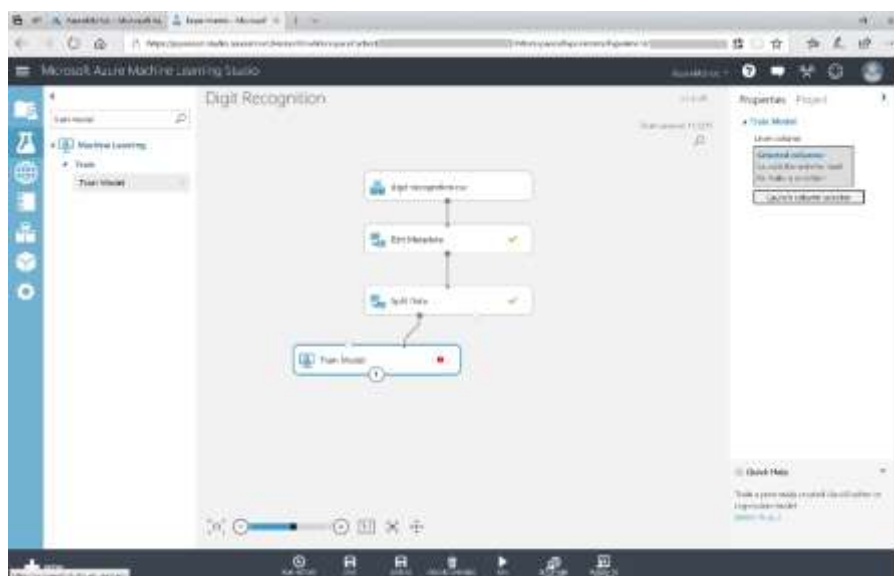
10. ページ下部の **[RUN]** ボタンをクリックして、Experiment を実行します。

[Train Model] モジュールによる学習アルゴリズムの設定

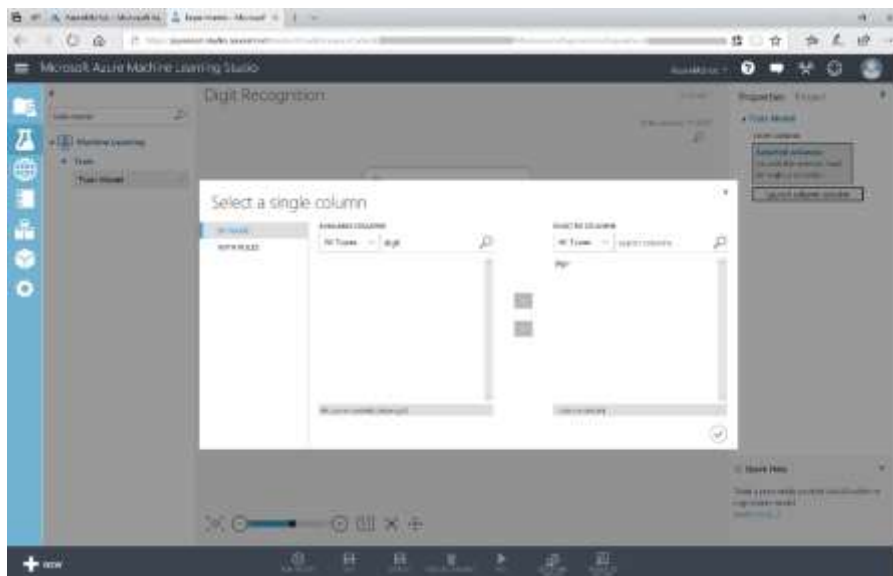
11. モジュール パレットの最上部にある検索ボックスに train model と入力して[Train Model] モジュールを表示し、キャンバスにドラッグします。[Split Data] モジュールの左の出力 を [Train Model] モジュールの右の入力に接続します。



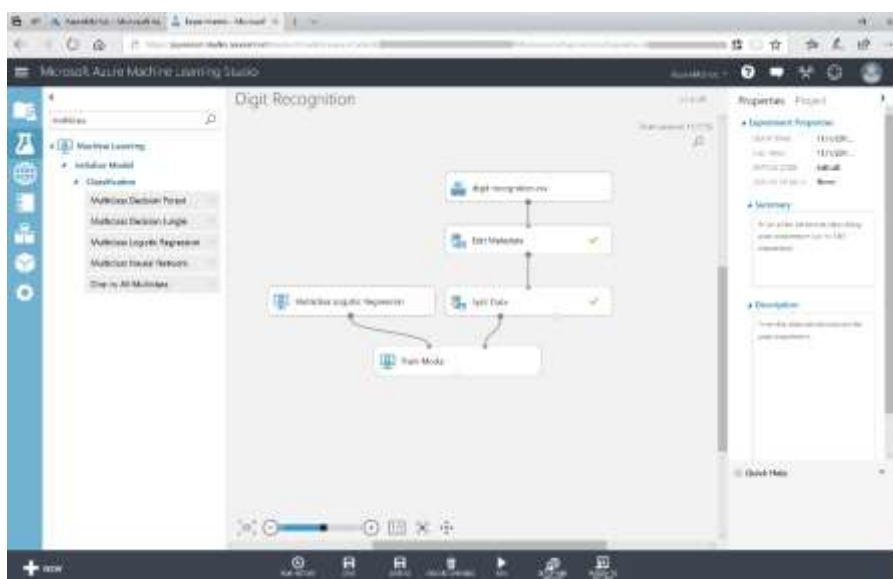
12. [Train Model] モジュールで予測することになる値を指定します。[Train Model] を 選択し、[Properties] ペインで **[Launch column selector]** をクリックします。



13. digit 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。



14. 次に、学習アルゴリズムを選択します。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。モジュール パレットの検索ボックスに multiclass と入力します。[Multiclass Logistic Regression] モジュールを見つけて、キャンバスにドラッグします。[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。

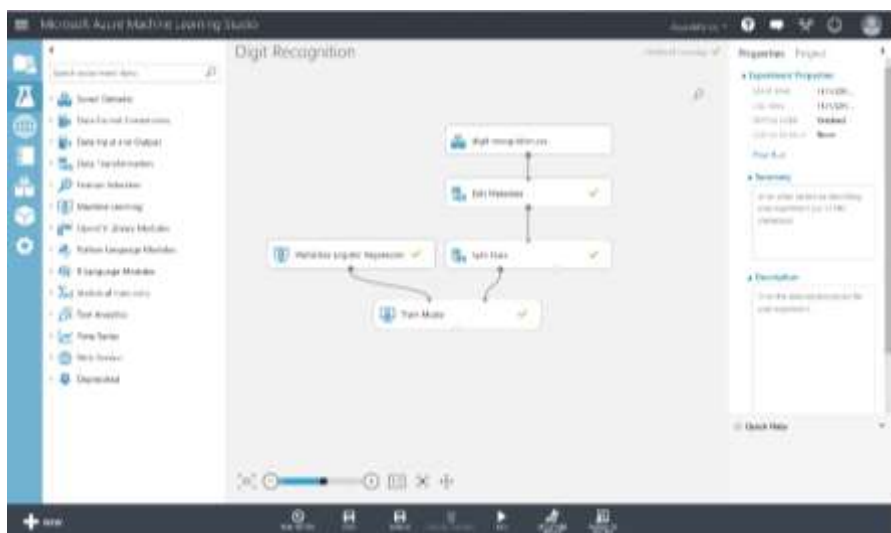


Azure Machine Learning では、数種類の分類アルゴリズムと回帰アルゴリズムをはじめ、さまざまな種類のアルゴリズムが提供されており、それぞれのアルゴリズムが、モジュール パレットでモジュールとして表されます。いつでも、R または Python で独自のアルゴリズムをコーディングすることもできます。

意図したモデルの構築を行うためにどのアルゴリズムを選択すべきかは、Azure Machine Learning チームが作成している [チート シート](#) をご覧ください。

15. **[SAVE]** ボタンをクリックして、Experiment を保存します。

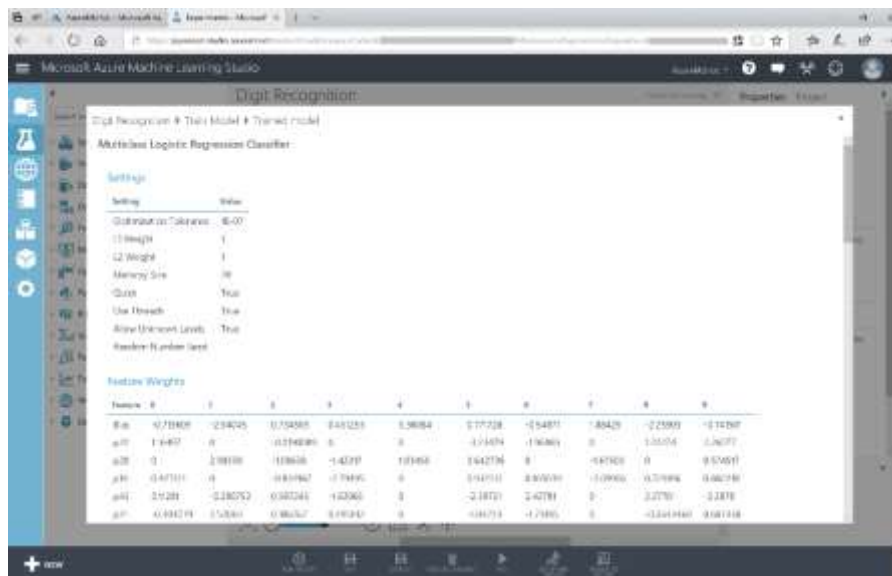
16. **[RUN]** ボタンをクリックして、Experiment を実行します。すべてのモジュールに、正常に実行されたことを示す緑色のチェック マークが表示されるまで待ちます。



17. 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。



18. モデルの Feature (列) と、それに割り当てられている Weight (重み付け) の一覧が表示されます。右上隅にある [x] をクリックして、Visualize ウィンドウを終了します。



Digit Recognition - This Model - Trained model

Multiclass Logistic Regression Classifier

Settings

| Setting | Value |
|------------------------|-------|
| Optimization Technique | SGD |
| L1 Weight | 0 |
| L2 Weight | 0 |
| Memory Size | 30 |
| Quiet | True |
| Use Threads | True |
| Stop Unknown Levels | True |
| Random Number Seed | |

Feature Weights

| Feature | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---------|----------|----------|-----------|---------|---------|----------|----------|----------|-----------|----------|
| bias | -0.71808 | -0.54725 | 0.75455 | 0.41233 | 0.36084 | 0.77228 | -0.54871 | -0.8428 | -0.25893 | -0.14797 |
| x[0] | 1.4407 | 0 | -0.040895 | 0 | 0 | -0.04094 | -0.06861 | 0 | 0.05054 | -0.0077 |
| x[1] | 0 | 0.08188 | 0.0836 | -0.079 | 0.0468 | 0.047794 | 0 | -0.07929 | 0 | 0.05491 |
| x[2] | -0.47111 | 0 | -0.04967 | -0.0495 | 0 | 0.04711 | 0.05039 | -0.09602 | 0.04986 | 0.00296 |
| x[3] | 0.04214 | -0.00752 | 0.00745 | -0.0065 | 0 | -0.0071 | 0.00781 | 0 | 0.00779 | -0.0078 |
| x[4] | 0.004214 | 0.00401 | 0.00407 | 0.00401 | 0 | -0.00413 | -0.00401 | 0 | -0.004060 | 0.004018 |

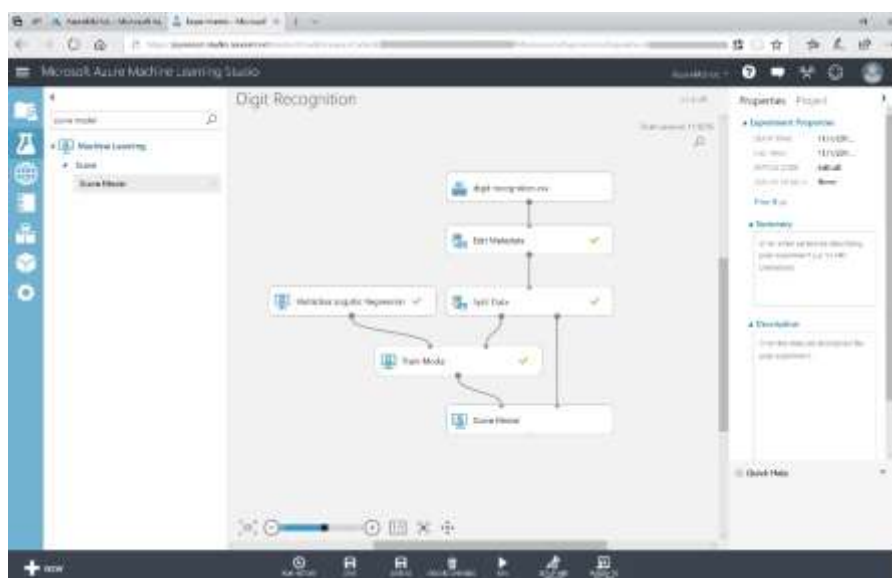
これでモデルにトレーニングを行うことができました。ただし、値の予測において、モデルはどれほど正確なのでしょうか。これが、モデルの評価を行う理由です。

演習 4: モデルを評価する

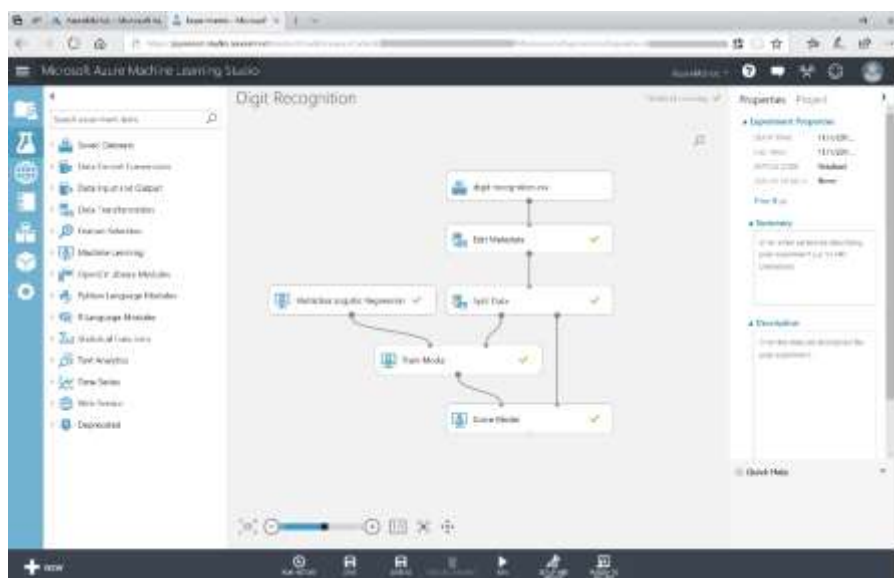
この演習では、前の演習でトレーニングを行ったモデルを評価します。これによって、モデルのトレーニングがどれほどうまく行われたか、つまり、モデルが投入された値からターゲット値をどれほど正確に予測できるかを判断できます。ML Studio では、モデルの評価として “スコア付け” するプロセスを簡単に実行できます。これまでの演習で、アップロードしたデータセットの 80% をモデルのトレーニングに使用しており、残りの 20% をモデルの評価に使用します。

[Score Model] モジュールによる評価用データによる実行

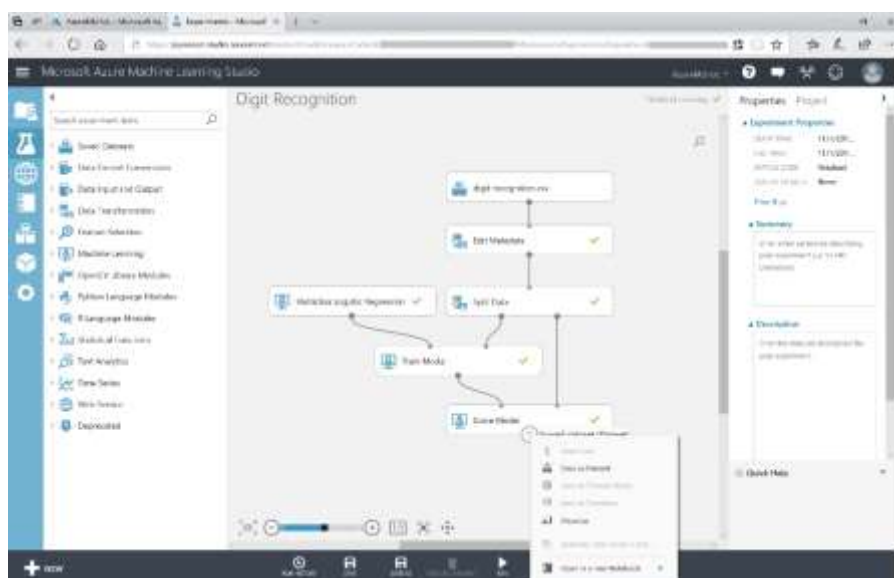
1. モジュールパレットから [Score Model] モジュール を検索し、キャンバスに追加します。[Train Model] モジュールの出力 を [Score Model] モジュールの左の入力に接続します。次に、[Split Data] モジュールの右の出力 を [Score Model] モジュールの右の入力 に接続します。この接続は、トレーニングには使用されなかった 20% のデータを表します。



2. **[SAVE]** ボタンをクリックして、Experiment を保存します。
3. **[RUN]** ボタンをクリックして、Experiment を実行し、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。



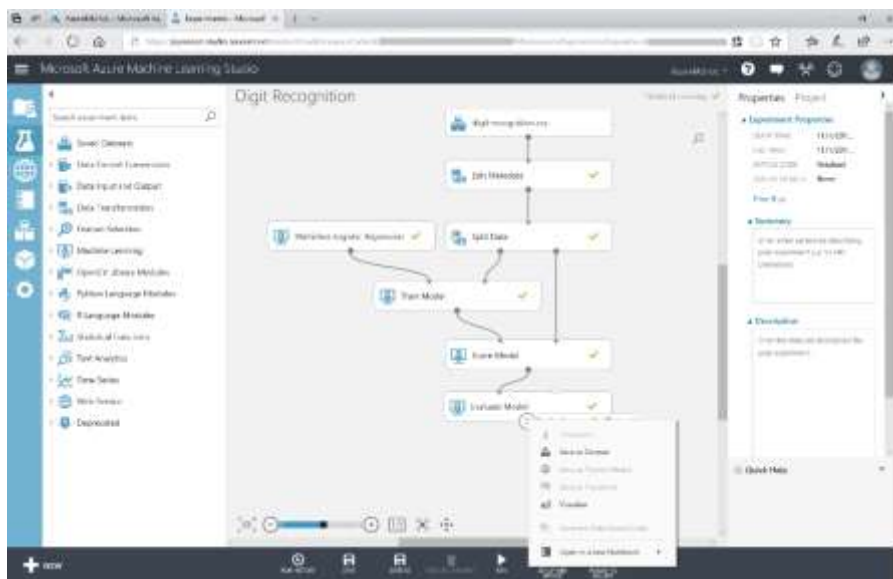
4. 実行が完了したら、[Score Model] モジュールの出力 を右クリックして、[Visualize] を選択します。



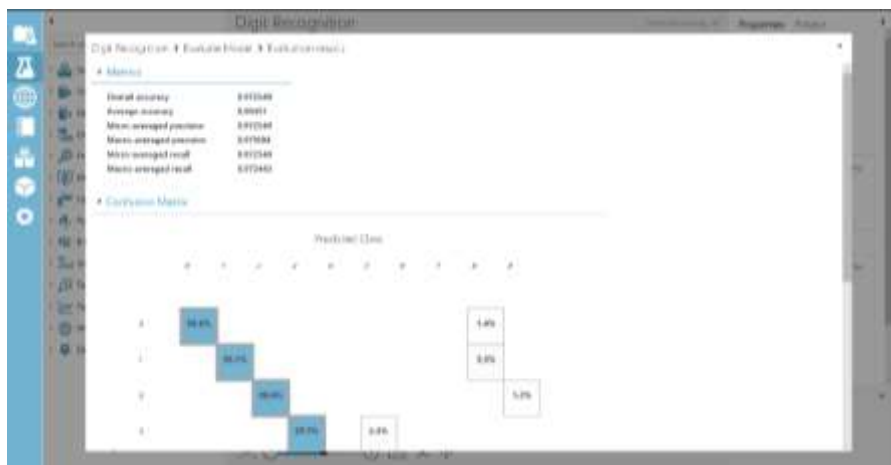
5. モデルの評価結果が表示されます。右にスクロールすると、[p01]~[p64] と [digit] の列には、[Score Model] モジュールに渡された 20% のデータセットの実際の値が表示されています。その横にある[Scored Probabilities for Class "0"~[... Class "9"] という名前の列は、モデルが[p01]~[p64] の入力値を使用して、定められた 10 個の出力値 (数字 0 ~ 9) に対してどのようにスコア付けしたかを示しています。数値が高い方が、一致の可能性が高くなります。最終列の [Scored Labels] は、入力値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

[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。

8. **[Save]** をクリックして、Experiment を保存します。**[RUN]** ボタンをクリックして、もう一度 Experiment を実行します。
9. **[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。



10. [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。今回の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。



11. 右上隅にある [x] をクリックして、Visualize 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、次はモデルの利用に移ります。最終的な目的は、このモデルを利用するアプリ（クラウドに接続可能な PC またはモバイルアプリ）を構築することです。アプリから簡単にモデルを利用するには、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

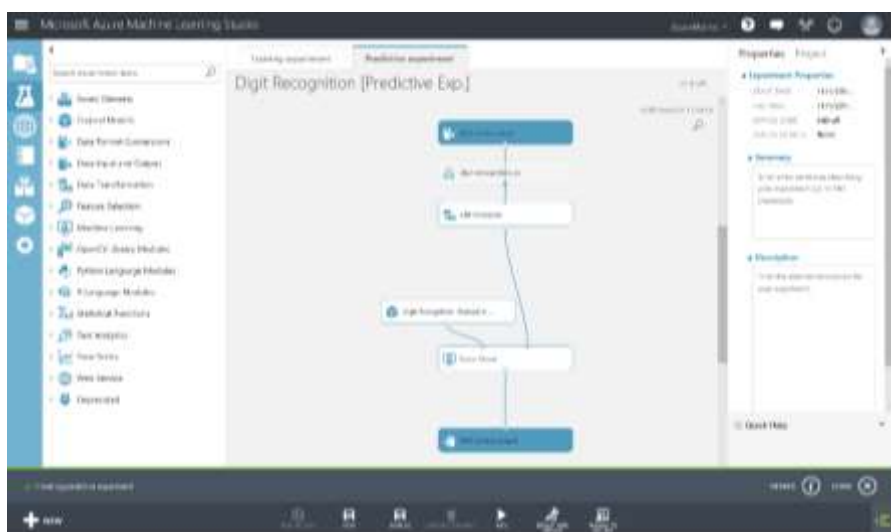
モデルのトレーニングと評価が完了したら、そのモデルを Web サービスとして展開することで、アプリケーションから利用できるようになります。Web サービスとして展開する前に、モデルを効率化する必要があります。これには、トレーニングを行ったモデルを元にした新しい Experiment の作成、不要なモジュールの削除、および Web サービスの入力および出力モジュールの追加が含まれます。ML Studio ではこれらすべての処理を簡単に行うことができます。

Predictive Experiment の作成

1. **[RUN]** ボタンをクリックして、Experiment 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。



3. ML Studio が数秒間処理を行い、その後に Predictive Experiment が表示されます。
[Split Data] と [Train Model] がなくなり、そのすべてのトレーニング データが [Score Model] に流れるようになります。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ます。



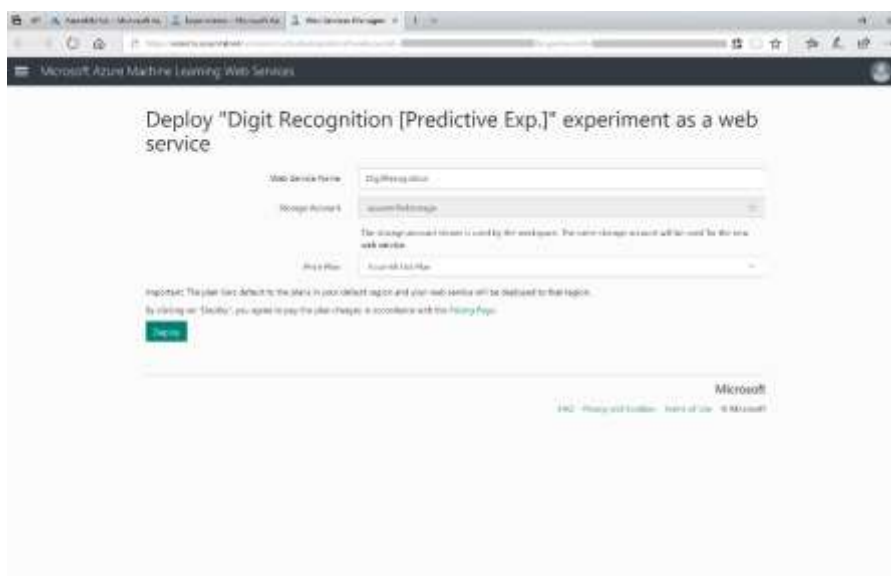
元のモデルはなくなったわけではなく、ページの最上部にある [Training experiment] タブをクリックすると表示できます。

Web サービスの展開

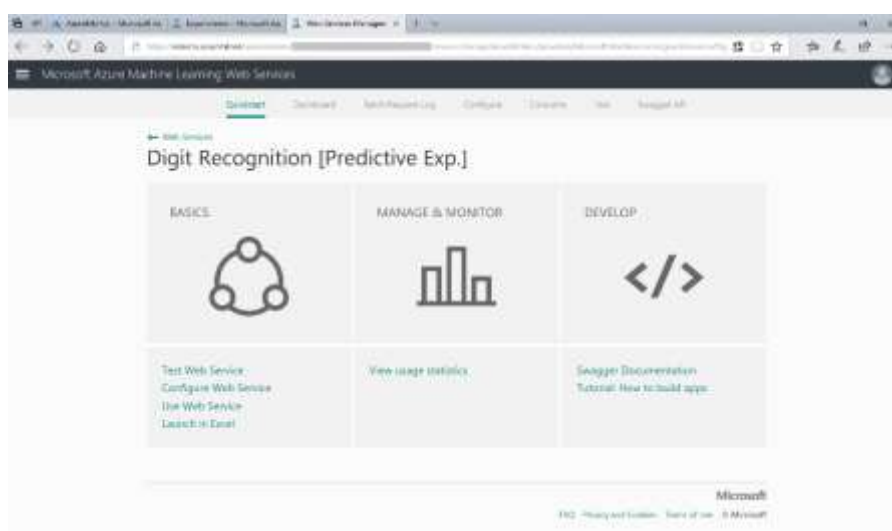
4. モデルを利用するための呼び出し可能な Web サービスを作成するには、[RUN] をもう一度クリックします。処理が完了したら、[DEPLOY WEB SERVICE (New) Preview] をクリックして、Web サービスを展開します。



5. ブラウザーの別のタブが開きます。新しい Web サービスを発行するための情報を入力します。Storage Account と Service Plan は Azure ポータルで作成したものを選択します。[Deploy] をクリックして発行します。

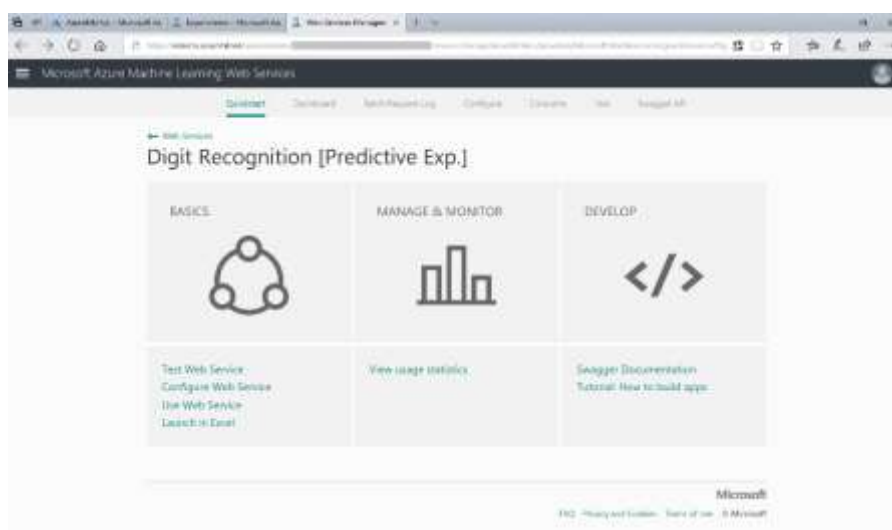


6. 発行された Web サービスのダッシュボードが表示されます。



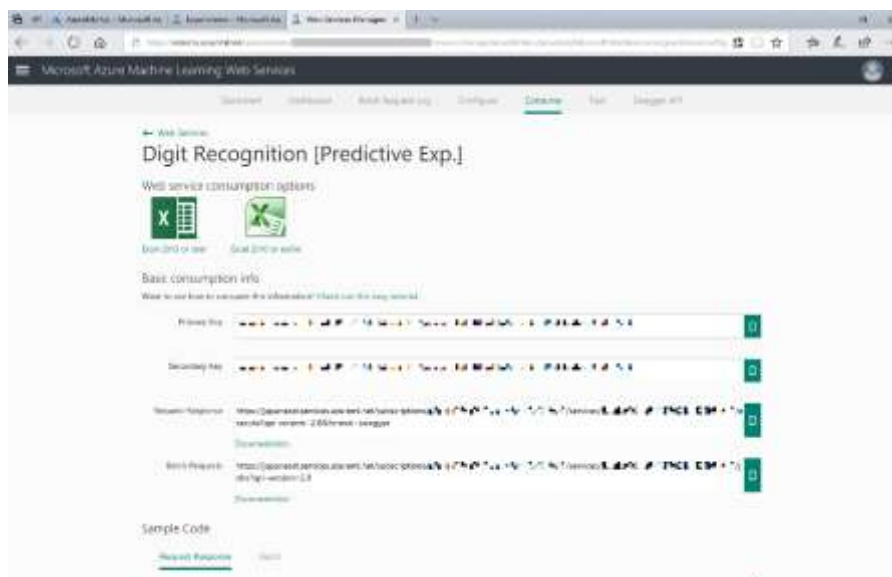
Azure ML Web Services の REST API 情報の表示

7. **Basic** の枠に表示されている **Use Web Service** をクリックして、[REST](#) を使用した HTTPS 経由での Web サービスの呼び出しを行うための情報画面を表示します。



このダッシュボードには、データを直接入力して Web サービスをテストする **Test Web Service** なども用意されています。(今回のモデルでは、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)

8. Web サービスの詳細画面には、サービスに対して認証済み呼び出しを行うときに使用する API キー (Primary Key、Secondary Key)、および リクエスト URL (Request-Response、Batch Requests) が表示されています。



9. **Primary Key** 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保存しておきます。同様に **Request-Response** の URL も保存しておきます。

このページには、Web サービスを利用するアプリを作成する開発者向けに、API 呼び出しの方法を示したサンプル コード(C#, Python, R)が含まれています。

Azure ML Web サービスの価格情報については、[Machine Learning の価格](#) ページを参照してください

演習 6: ユニバーサル Windows アプリケーションを構築する

Azure ML モデルを Web サービスとして展開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、Visual Studio を使用して、iOS、Android、または Windows で動作し、.NET の HttpClient クラスを使用して Web サービスを呼び出す Xamarin アプリを作成することもできます。

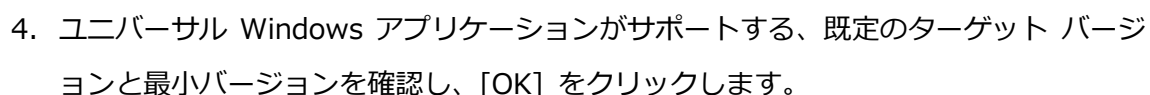
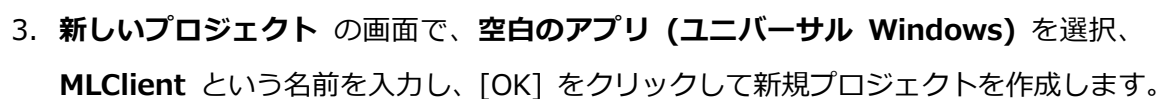
この演習では、ユニバーサル Windows プラットフォーム (UWP) を対象とするクライアント アプリを作成します。このアプリの利点は、PC、タブレット、スマートフォン、そして Xbox One まで含めた、さまざまな Windows デバイスで動作することです。画面上のグリッド内に描画した数字を読み取り、ML Web サービスを呼び出して、描画した数字を伝えるアプリを作成します。

Windows 10 での開発者モードの有効化

1. Windows 10 PC で UWP アプリを構築して実行するには、デバイスで開発者モードを有効にする必要があります。開発者モードが有効になっていることを確認するには、デスクトップの左下隅にある Windows ボタン (スタート ボタンともいいます) をクリックします。メニューから [設定] を選択して、[Windows の設定] ダイアログで [更新とセキュリティ] をクリックします。左側の [開発者向け] をクリックして、右側の [開発者モード] を選択します。



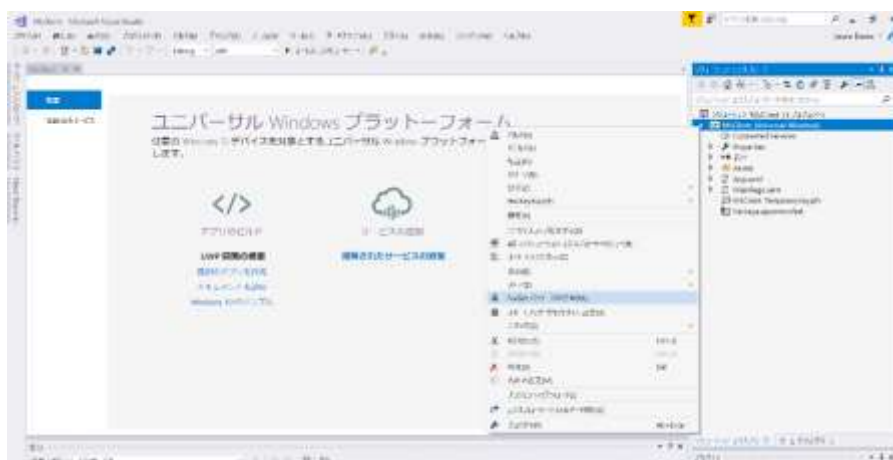
2. Visual Studio 2017 を起動して、**[ファイル] > [新規作成] > [プロジェクト]** の順にクリックします。



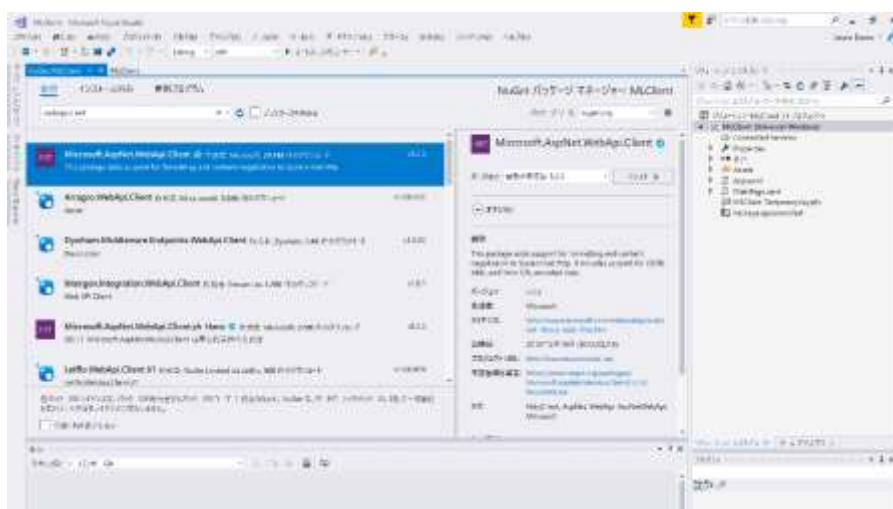
NuGet によるパッケージの管理と Web API クライアントのインストール

NuGet は、Microsoft 管理プラットフォーム向けの無料のオープンソース パッケージです。NuGet では、さまざまなタスクを実行するコードが収録された数千ものライブラリのパッケージを利用できます。NuGet は Visual Studio 2017 に統合されており、簡単に NuGet 経由でパッケージをプロジェクトに追加、管理することができます。

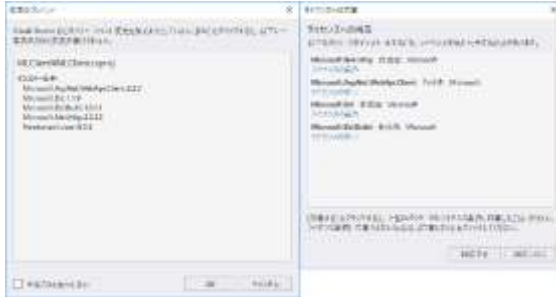
5. [ソリューションエクスプローラー] ウィンドウで、MLClient プロジェクトを右クリックし、[NuGet パッケージの管理..] を選択します。



6. [参照] をクリックします。検索ボックスに、**webapi.client** と入力します。**Microsoft.AspNet.WebApi.Client** をクリックして、NuGet のこの Web API クライアント パッケージを選択します。[インストール] をクリックして、**最新の安定版** のパッケージをインストールします。このパッケージには、アプリで Web サービスとの通信に使用する ヘルパー API が収録されています。



7. 変更の確認を求められたら [OK] をクリックし、ダウンロードしたパッケージのライセンスへの同意を求められたら [同意する] をクリックします。



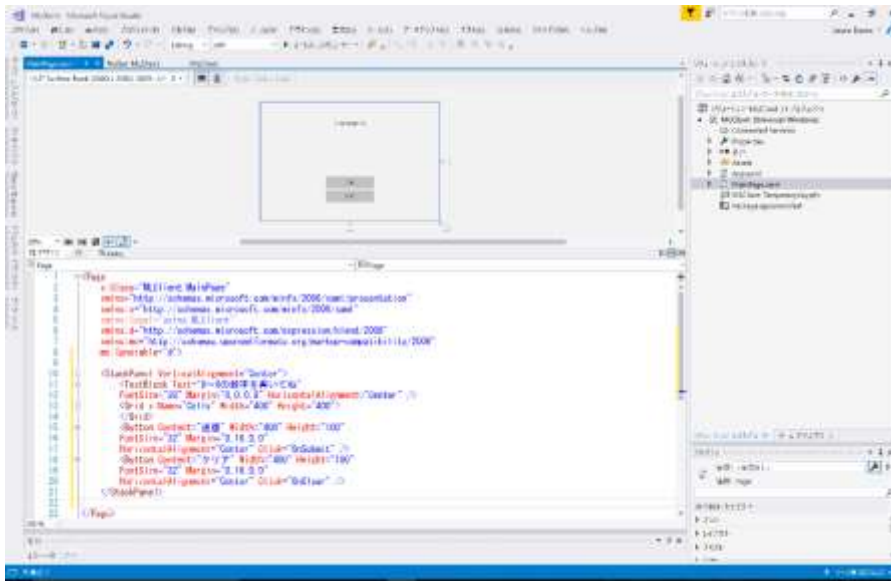
UI (MainPage.xaml) のコーディング

8. [ソリューションエクスプローラー] ウィンドウで、**MainPage.xaml** を開きます。
XAML ウィンドウ内のコードから、以下の空の Grid 要素を見つけます。

```
<Grid Background="{ThemeResource ApplicationPage ... (中略) ...}">  
  
</Grid>
```

9. 上記の空の Grid 要素を次のコードに置き換えます。

```
<StackPanel VerticalAlignment="Center">  
    <TextBlock Text="0～9 の数字を書いてね"  
        FontSize="28" Margin="0,0,0,8" HorizontalAlignment="Center" />  
    <Grid x:Name="Cells" Width="400" Height="400">  
    </Grid>  
    <Button Content="送信" Width="400" Height="100"  
        FontSize="32" Margin="0,16,0,0"  
        HorizontalAlignment="Center" Click="OnSubmit" />  
    <Button Content="クリア" Width="400" Height="100"  
        FontSize="32" Margin="0,16,0,0"  
        HorizontalAlignment="Center" Click="OnClear" />  
</StackPanel>
```



挿入したマークアップは、Extensible Application Markup Language (XAML) です。XAML は、ユーザー インターフェイスの構築用にマイクロソフトが作成した言語です。当初は WPF に向けて作成されましたが、その後ユニバーサル Windows アプリ向けに用途変更されています。XAML を Xamarin Forms と組み合わせると、iOS や Android 向けのユーザー インターフェイスの構築にも使用できます。XAML は、Visual Studio やその他のよく使用されているツールでデザイナー サポートを提供する、表現がきわめて豊富な言語です。

実行部分 (MainPage.xaml.cs) のコーディング

10. **[ソリューションエクスプローラー]** ウィンドウで、**MainPage.xaml.cs** を開き、ページの上部に既にある using ステートメントに、以下の using ステートメントを追加します。

```
using Windows.UI.Xaml.Shapes;
using Windows.UI;
using Windows.Devices.Input;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;
using Windows.UI.Core;
using Windows.UI.Popups;
using Newtonsoft.Json;
```

11. 続けて MainPage.xaml.cs の MainPage クラス内に、8 × 8 のグリッドと Rectangle を追加するコードを追加します。

変更前

```
public MainPage()
{
    this.InitializeComponent();
}
```

↓

変更後

```
private const double _margin = 2.0; // セルの配置マージン
private const double _opacity = 0.2; // セルの透明度
private Rectangle _last; // 最終入力セル

public MainPage()
{
    this.InitializeComponent();

    // Grid に行列を作成
    for (int i = 0; i < 8; i++)
        Cells.ColumnDefinitions.Add(new ColumnDefinition());
    for (int j = 0; j < 8; j++)
        Cells.RowDefinitions.Add(new RowDefinition());

    // 正方形で行列を埋める
    for (int row = 0; row < 8; row++)
    {
        for (int col = 0; col < 8; col++)
        {
            var cell = new Rectangle();
            cell.Fill = new SolidColorBrush(Colors.Blue);
            cell.Opacity = _opacity;
            cell.Margin = new Thickness(_margin);
            cell.SetValue(Grid.RowProperty, row);
            cell.SetValue(Grid.ColumnProperty, col);
            cell.PointerPressed += OnCellPressed;
            cell.PointerEntered += OnCellEntered;
            Cells.Children.Add(cell);
        }
    }
}
```

```

    }
}
}

```

このコードによる、UI の Grid 要素に 8 × 8 のグリッドと XAML の Rectangle が追加されます。ここでは、コード内で XAML 要素を作成することによって、MainPage.xaml に、Rectangle 要素と同じ行が追加されないようにしています。

12. UI による描画を操作するメソッド (OnCellPressed、OnCellEntered、ToggleCell) を追加します。

```

private void OnCellPressed(object sender, PointerRoutedEventArgs e)
{
    if (e.Pointer.PointerDeviceType == PointerDeviceType.Mouse)
    {
        var point = e.GetCurrentPoint(null);

        if (point.Properties.IsLeftButtonPressed) //マウス左ボタンのみ
        {
            var cell = (Rectangle)sender;
            ToggleCell(cell); //セルのトグル(On&Off 切り替え)
            _last = cell;
        }
    }
}

private void OnCellEntered(object sender, PointerRoutedEventArgs e)
{
    var cell = (Rectangle)sender;

    if (e.Pointer.PointerDeviceType == PointerDeviceType.Mouse)
    {
        var point = e.GetCurrentPoint(null);

        if (!point.Properties.IsLeftButtonPressed)
            return; //マウス左ボタン以外は無視

        if (cell == _last)

```



```

        {
            _last = null;
            return; // 入力中は無視
        }
    }

    ToggleCell(cell);
}

private void ToggleCell(Rectangle cell)
{
    // セルのトグル(もう一度クリックすると On <-> Off 切り替え)
    cell.Opacity = (cell.Opacity < 1.0) ? 1.0 : _opacity;
}

```

OnCellEntered は、指、ペン、またはマウス (UWP の用語では、ポインティングデバイス) が 64 個の Rectangle のいずれかに接触したときに呼び出されるメソッドです。OnCellEntered は、Rectangle の不透明度を変更することによる、"オン" と "オフ" の切り替えです。

13. [送信] および [クリア] ボタンをクリックしたときの動作を記述します。

```

private async void OnSubmit(object sender, RoutedEventArgs e)
{
    // 画面の入力を取得
    string[] values = new string[65];

    for (int row = 0; row < 8; row++)
    {
        for (int col = 0; col < 8; col++)
        {
            int index = (row * 8) + col;
            values[index]
                = ((Rectangle)Cells.Children[index]).Opacity == 1.0 ? "16" : "0";
        }
    }
}

```

```

values[64] = "0"; // digit 値は 0 に固定

try
{
    // ML 呼び出し
    await MLSubmitAsync(values);
}
catch (Exception ex)
{
    // エラーハンドラー (エラーメッセージを画面表示)
    var dialog = new MessageDialog(ex.Message);
    await dialog.ShowAsync();
}
}

private void OnClear(object sender, RoutedEventArgs e)
{
    for (int i = 0; i < 64; i++)
        ((Rectangle)Cells.Children[i]).Opacity = _opacity;
}

```

OnSubmit メソッドは、[Submit] ボタンをクリックしたときに呼び出されます。OnSubmit メソッドは、8 × 8 グリッドをスキャンして、どのスクエアが "オン" であるか判別し、データを MLSubmit に渡します。

14. 演習 1 ～5 で作成した ML Web サービスを呼び出すコードを記述します。

```

private async Task MLSubmitAsync(string[] v)
{
    using (var client = new HttpClient())
    {
        // ML に送信するデータセットの作成
        var scoreRequest = new
        {
            Inputs = new Dictionary<string, List<Dictionary<string, string>>>()
            {
                {

```

```

        "input1",
        new List<Dictionary<string, string>>()
        {new Dictionary<string, string>()
        {
            {"p01", v[0]}, {"p02", v[1]}, {"p03", v[2]}, {"p04", v[3]},
            {"p05", v[4]}, {"p06", v[5]}, {"p07", v[6]}, {"p08", v[7]},
            {"p09", v[8]}, {"p10", v[9]}, {"p11", v[10]}, {"p12", v[11]},
            {"p13", v[12]}, {"p14", v[13]}, {"p15", v[14]}, {"p16", v[15]},
            {"p17", v[16]}, {"p18", v[17]}, {"p19", v[18]}, {"p20", v[19]},
            {"p21", v[20]}, {"p22", v[21]}, {"p23", v[22]}, {"p24", v[23]},
            {"p25", v[24]}, {"p26", v[25]}, {"p27", v[26]}, {"p28", v[27]},
            {"p29", v[28]}, {"p30", v[29]}, {"p31", v[30]}, {"p32", v[31]},
            {"p33", v[32]}, {"p34", v[33]}, {"p35", v[34]}, {"p36", v[35]},
            {"p37", v[36]}, {"p38", v[37]}, {"p39", v[38]}, {"p40", v[39]},
            {"p41", v[40]}, {"p42", v[41]}, {"p43", v[42]}, {"p44", v[43]},
            {"p45", v[44]}, {"p46", v[45]}, {"p47", v[46]}, {"p48", v[47]},
            {"p49", v[48]}, {"p50", v[49]}, {"p51", v[50]}, {"p52", v[51]},
            {"p53", v[52]}, {"p54", v[53]}, {"p55", v[54]}, {"p56", v[55]},
            {"p57", v[56]}, {"p58", v[57]}, {"p59", v[58]}, {"p60", v[59]},
            {"p61", v[60]}, {"p62", v[61]}, {"p63", v[62]}, {"p64", v[63]},
            {"digit", v[64]},
        }
        },
        },
        GlobalParameters = new Dictionary<string, string>()
        {
        }
    };

    // 「api_key」にご自分の Api キーをコピーしてください
    const string apiKey = "api_key";
    client.DefaultRequestHeaders.Authorization
        = new AuthenticationHeaderValue("Bearer", apiKey);
    // 「web_service_url」にご自分の サービス URL をコピーしてください
    client.BaseAddress = new Uri("web_service_url");

    // ML にアクセスして結果を取得
    HttpResponseMessage response
    = await client.PostAsJsonAsync("", scoreRequest).ConfigureAwait(false);

    await this.Dispatcher.RunAsync(CoreDispatcherPriority.Normal, async () =>

```

```

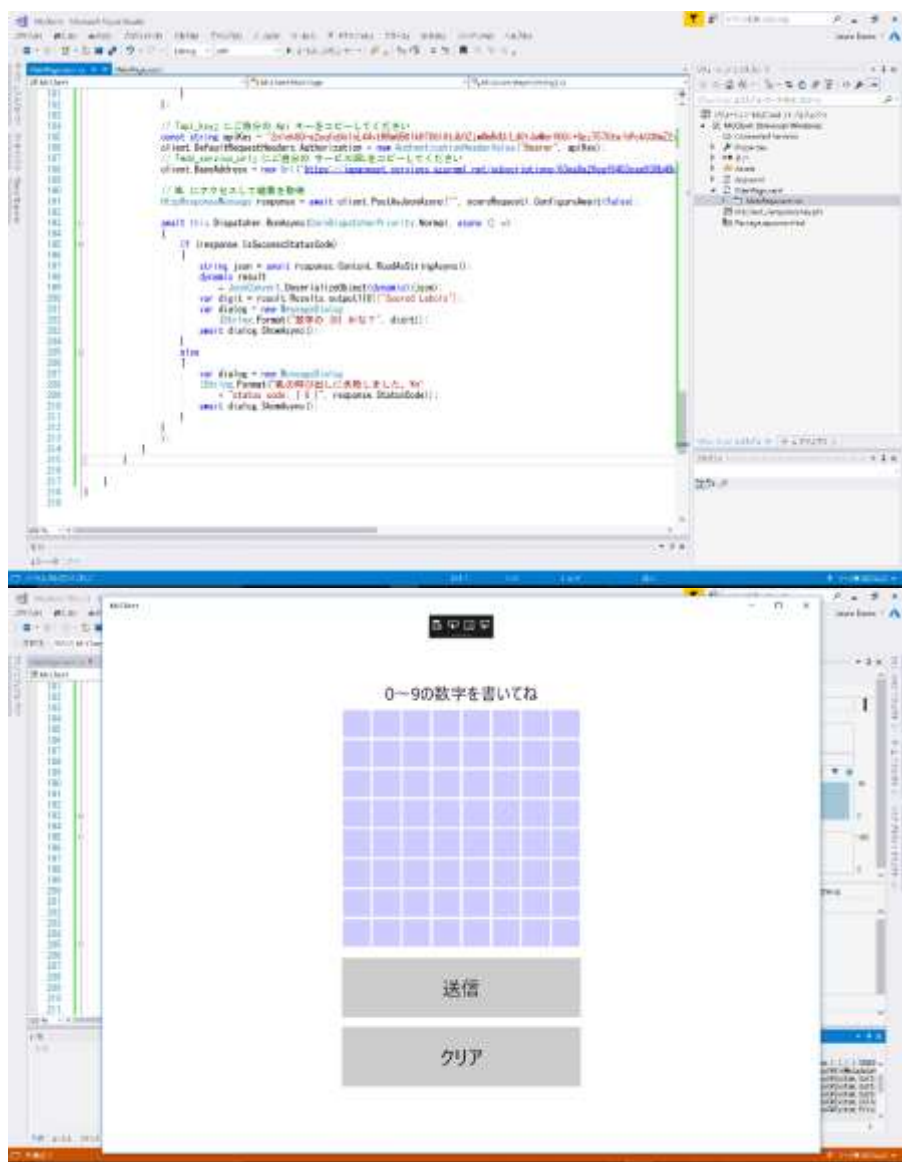
        {
            if (response.IsSuccessStatusCode)
            {
                string json = await response.Content.ReadAsStringAsync();
                dynamic result
                    = JsonConvert.DeserializeObject<dynamic>(json);
                var digit = result.Results.output1[0]["Scored Labels"];
                var dialog = new MessageDialog
                    (String.Format("数字の {0} かな？", digit));
                await dialog.ShowAsync();
            }
            else
            {
                var dialog = new MessageDialog
                    (String.Format("ML の呼び出しに失敗しました。¥n"
                        + "status code: { 0 }", response.StatusCode));
                await dialog.ShowAsync();
            }
        }
    };
}
}

```

MLSubmitAsync は、UWP の HttpClient クラスを使用して、Web サービスに対して REST 呼び出しを実行します。MLSubmitAsync は、Web サービスのダッシュボードに表示される C# のサンプル コードと同様に動作します。

15. 12. のコード内にある api_key と web_service_url を、演習 5 で取得した API キー および Web サービスの URL に置き換えます。以上でコーディング作業は終了です。

16. [Visual Studio] ウィンドウの上部にある **[ビルド] > [ソリューションのビルド]** をクリックして、ソリューションをビルドします。ビルド エラーがある場合はそれを修正してから、Ctrl + F5 キーを押してアプリを起動し、アプリが動作するのを確認してください。

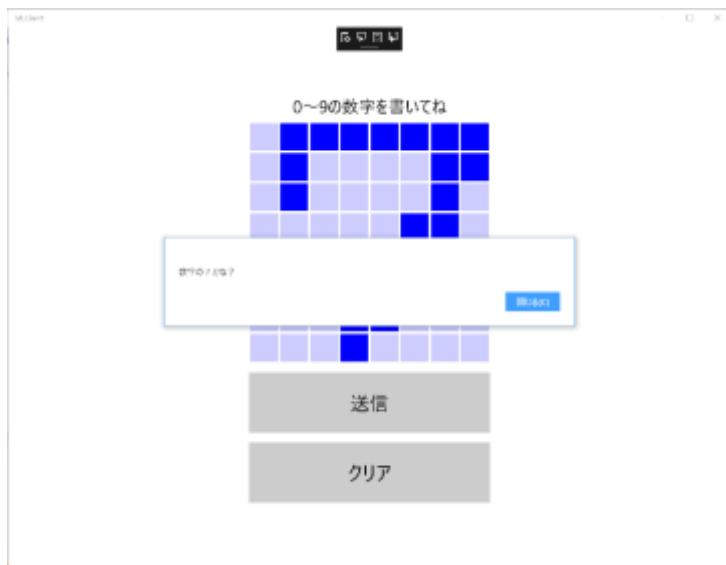


アプリの動作確認

MLClient は、視覚的なフロントエンドによる Web サービスの呼び出しをおこなうアプリケーションです。グリッドのスクエアの上に指、ペン、または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。[送信] ボタンをクリックすると、64 個のグリッドがオンになっている値によって配列が作成され、その配列が JSON にシリアル化されて、ML Web サービスに渡されます。Web サービスからの返される 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。

17. マウスを使用してグリッドに「7」と描画します。その後、**[送信]** ボタンをクリックし

ます。ポップアップ ウィンドウが表示され、描画した数字が示されます。数字は合っていますか。



18. [クリア] 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るでしょう。

答えが間違っている理由として、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る（今回のデータセットは約 0.5 MB で、ビッグデータの標準としては、小規模です）ということ、また、MLClient アプリの画面上のグリッド表示では、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 しか使用されていないということが挙げられます。

おつかれさまでした。以上で、Azure Machine Learning を利用したインテリジェンスを利用するユニバーサル Windows アプリの完成です。

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 Experiment 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評価を行う
- 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、予測分析のわくわくする世界を探索してください。

Copyright 2017 Microsoft Corporation. All rights reserved.