インテル® DAAL を利用した線形回帰モデルの最適化

この記事は、インテル® デベロッパー・ゾーンに公開されている「Optimize Linear Regression Model with Intel® DAAL」の日本語参考訳です。

企業は、売上を伸ばすため、将来の広告費をどのように予測しているのでしょうか? 同様に、生産性を向上させるため、研修プログラムにどれぐらい投資すべきでしょうか?

どちらの場合も、これまでの変数の関係、例えば前者の場合は将来の結果を予測するため売上と広告費の関係に依存します。

これは、マシンラーニングでうまく解決できる典型的な回帰問題です [1]。

この記事では、線形回帰 [2] と呼ばれる一般的な回帰分析について説明し、インテル® データ・アナリティクス・アクセラレーション・ライブラリー (インテル® DAAL)[3] を使用してこのアルゴリズムをインテル® Xeon® プロセッサー・ベースのシステム向けに最適化する方法を紹介します。

線形回帰とは?

線形回帰 (LR) は、予測分析に使用される最も基本的な回帰です。LR は、変数間の線形関係と、ある変数が 1 つまたは複数の変数によってどのような影響を受けるかを示します。ほかの変数によって影響を受ける変数は従属変数、応答変数、または結果変数と呼ばれ、ほかの変数は独立変数、説明変数、または予測変数と呼ばれます。

LR 分析を使用するため、LR がこのデータセットに適しているか検証する必要があります。そのためには、データの分布を確認します。次の 2 つのグラフについて考えます。

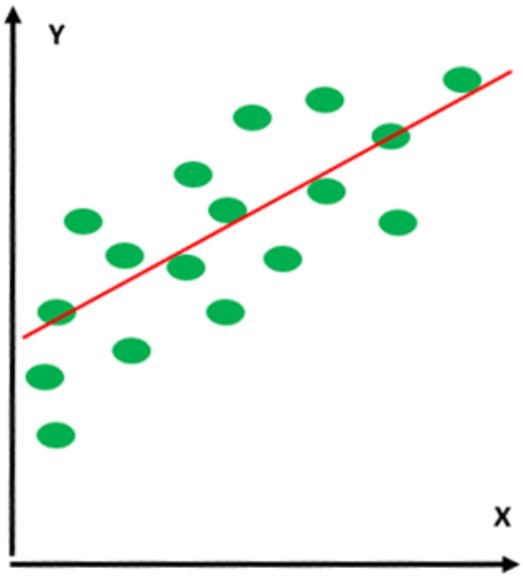

図 1. データポイントを通り抜ける直線を描ける場合

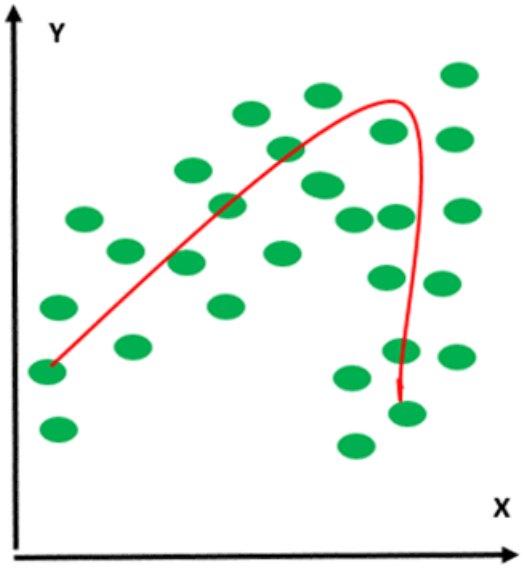

図 2. データポイントを通り抜ける直線が描けない場合

図 1 ではデータポイントを通り抜ける直線を描くことができますが、図 2 では (直線ではなく) 曲線のみ描くことができます。そのため、図 1 のデータセットでは線形回帰分析を行うことができますが、図 2 のデータセットではできません。

独立変数の数に応じて、LR は単純線形回帰 (SLR) と多重線形回帰 (MLR) に分けられます。

LR は、独立変数が 1 つの場合は SLR と呼ばれ、複数の場合は MLR と呼ばれます。

1 つの従属変数と 1 つの独立変数を持つ方程式の最も単純な形式は、次のように定義できます。

$$y = Ax + B \qquad (1)$$

説明:
- y: 従属変数
- x: 独立変数
- A: 回帰係数または線の傾き
- B: 定数

問題は、従属変数 (y) の観測値と予測値の差が最小になる最適な線をどのように見つけるかです。つまり、方程式 (1) で $|y_{observed} - y_{predicted}|$ が最小となる A と B を見つけます。

最適な線は、最小 2 乗法を使用して見つけることができます [4]。各データポイントから線までの垂直方向の差 (観測値と予測値の差) の 2 乗和を最小化することで、最適な線を求めます。垂直方向の差は残差とも呼ばれます。

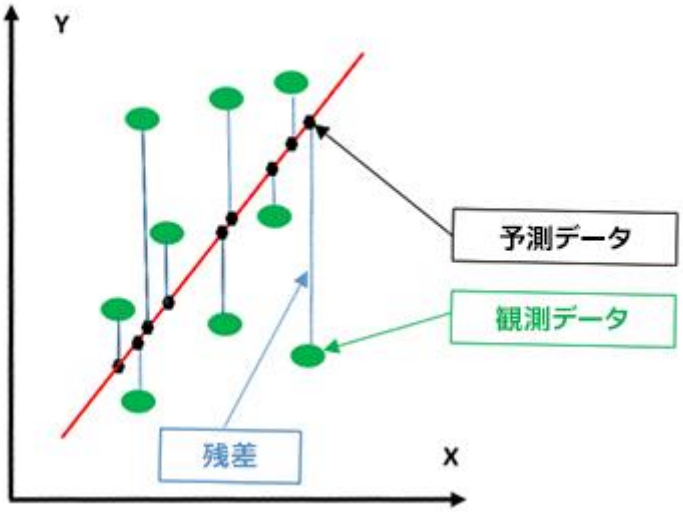

図 3. 線形回帰グラフ

図 3 の緑の点は実際のデータポイントを示します。黒い点は、回帰線 (赤い線) へのデータポイントの垂直投影 (垂線ではない) を示します。黒い点は予測データとも呼ばれます。実際のデータと予測データの垂直方向の差は残差と呼ばれます。

線形回帰の適用例

以下は、線形回帰の適用です。

- 将来の販売予測
- 製品の販売におけるマーケティング効果と価格設定の分析
- 金融サービスや保険のリスク評価
- 自動車における試験データからのエンジン性能の調査

線形回帰のメリットとデメリット

以下は、LR のメリットとデメリットです。

- **メリット**
  - 独立変数と従属変数の関係がほぼ線形である場合に最適な結果が得られます。
- **デメリット**
  - LR は外れ値に対して非常に敏感です。
  - 非線形関係のモデル化には適していません。
  - 線形回帰は、数値出力の予測に限定されます。

インテル® DAAL

インテル® DAAL は、データ解析とマシンラーニング向けに最適化された多くの基本ビルディング・ブロックからなるライブラリーです。これらの基本ビルディング・ブロックは、最新のインテル® プロセッサーの機能向けに高度に最適化されています。LR は、インテル® DAAL が提供する予測アルゴリズムの 1 つです。この記事では、インテル® DAAL の Python* API を使用して基本的な LR 予測子を作成します。インテル® DAAL をインス

トールするには、「Linux* でインテル® DAAL の Python* バージョンをインストールする方法」(英語)[5] の手順に従ってください。

インテル® DAAL の線形回帰アルゴリズムを使用する

このセクションでは、インテル® DAAL の Python* 線形回帰アルゴリズム [6] を呼び出す方法を示します。

アプリケーションのテストに使用可能な無料のデータセットへのリンク [7] は、「関連情報」セクションにあります。

次のステップに従って、インテル® DAAL から線形回帰アルゴリズムを呼び出します。

1. from コマンドと import コマンドを使用して、必要なパッケージをインポートします。
    1. 次のコマンドを実行して NumPy* をインポートします。
       ```
       import numpy as np
       ```
    2. 次のコマンドを実行して、インテル® DAAL の数値テーブルをインポートします。
       ```
       from daal.data_management import HomogenNumericTable
       ```
    3. データを格納するため、数値テーブルに必要な関数をインポートします。
       ```
       from daal.data_management import ( DataSourceIface,
       FileDataSource, HomogenNumericTable, MergeNumericTable,
       NumericTableIface)
       ```
    4. 次のコマンドを実行して、LR アルゴリズムをインポートします。
       ```
       from daal.algorithms.linear_regression import training,
       prediction
       ```
2. データ入力が .csv ファイルからの場合、ファイル・データ・ソースを初期化します。
   ```
   trainDataSet = FileDataSource(
       trainDatasetFileName, DataSourceIface.notAllocateNumericTable,
       DataSourceIface.doDictionaryFromContext
   )
   ```
3. トレーニング・データと従属変数の数値テーブルを作成します。
   ```
   trainInput = HomogenNumericTable(nIndependentVar, 0,
   NumericTableIface.notAllocate)

   trainDependentVariables = HomogenNumericTable(nDependentVariables, 0,
   NumericTableIface.notAllocate)

   mergedData = MergedNumericTable(trainData, trainDependentVariables)
   ```
4. 入力データをロードします。
   ```
   trainDataSet.loadDataBlock(mergedData)
   ```
5. モデルをトレーニングする関数を作成します。
    1. 最初に、次のコマンドを実行して、モデルをトレーニングするアルゴリズム・オブジェクトを作成します。
       ```
       algorithm = training.Batch_Float64NormEqDense()
       ```

       注: このアルゴリズムは、線形最小 2 乗問題を解くため正規方程式を使用します。インテル® DAAL は、QR 分解/因数分解もサポートしています。

2. 次のコマンドを実行して、トレーニング・データセットと従属変数をアルゴリズムに渡します。
```
algorithm.input.set(training.data, trainInput)
algorithm.input.set(training.dependentVariables,
trainDependentVariables)
```

説明:

`algorithm`: 上記のステップで定義したアルゴリズム・オブジェクト。

`trainInput`: トレーニング・データ

`trainDependentVariables`: トレーニング従属変数

3. 次のコマンドを実行して、モデルをトレーニングします。
```
trainResult = algorithm.compute()
```

説明:

`algorithm`: 上記のステップで定義したアルゴリズム・オブジェクト。

6. モデルをテストする関数を作成します。
    1. 上記のステップ 2、3、および 4 と同様に、テスト・データセットを作成する必要があります。
        i.
        ```
        testDataSet = FileDataSource(
        testDatasetFileName, DataSourceIface.doAllocateNumericTable,
        DataSourceIface.doDictionaryFromContext
        )
        ```
        ii.
        ```
        testInput = HomogenNumericTable(nIndependentVar, 0,
        NumericTableIface.notAllocate)
        testTruthValues = HomogenNumericTable(nDependentVariables,
        0, NumericTableIface.notAllocate)
        mergedData = MergedNumericTable(testDataSet,
        testTruthValues)
        ```
        iii.
        ```
        testDataSet.loadDataBlock(mergedData)
        ```

    2. 次のコマンドを実行して、モデルをテスト/予測するアルゴリズム・オブジェクトを作成します。
    ```
    algorithm = prediction.Batch()
    ```

    3. 次のコマンドを実行して、テストデータとトレーニング済みモデルをモデルに渡します。
    ```
    algorithm.input.setTable(prediction.data, testInput)
    algorithm.input.setModel(prediction.model,
    trainResult.get(training.model))
    ```

    説明:

    `algorithm`: 上記のステップで定義したアルゴリズム・オブジェクト。

    `testInput`: テストデータ。

    4. 次のコマンド実行して、モデルをテスト/予測します。
    ```
    Prediction = algorithm.compute()
    ```

まとめ

線形回帰は、非常に一般的な予測アルゴリズムです。インテル® DAAL の線形回帰アルゴリズムは最適化されています。インテル® DAAL を使用することで、アプリケーションを変更せずに、インテル® DAAL の最新バージョンにリンクするだけで、将来の世代のインテル® Xeon® プロセッサーの新機能を利用できます。

関連情報

1. Wikipedia - 機械学習
2. 線形回帰
3. インテル® DAAL の概要 (英語)
4. 最小 2 乗法
5. Linux* でインテル® DAAL の Python* バージョンをインストールする方法 (英語)
6. Python* ウェブサイト (英語)
7. 一般的なデータセットのリスト (英語)

製品とパフォーマンス情報

[1] インテル® コンパイラーでは、インテル® マイクロプロセッサーに限定されない最適化に関して、他社製マイクロプロセッサー用に同等の最適化を行えないことがあります。これには、インテル® ストリーミング SIMD 拡張命令 2、インテル® ストリーミング SIMD 拡張命令 3、インテル® ストリーミング SIMD 拡張命令 3 補足命令などの最適化が該当します。インテルは、他社製マイクロプロセッサーに関して、いかなる最適化の利用、機能、または効果も保証いたしません。本製品のマイクロプロセッサー依存の最適化は、インテル® マイクロプロセッサーでの使用を前提としています。インテル® マイクロアーキテクチャーに限定されない最適化のなかにも、インテル® マイクロプロセッサー用のものがあります。この注意事項で言及した命令セットの詳細については、該当する製品のユーザー・リファレンス・ガイドを参照してください。

注意事項の改訂 #20110804