Hands on Machine Learning with Scikit-Learn &TensolFlow

Chapter 4

Training Linear Models

Created by Yusuke FUJIMOTO

はじめに

- この資料は「Hands-On Machine Learning with Scikit-Learn and TensorFlow - O'Reilly Media」を読んだ際の（主にソースコードに関する）簡単な解説を残したものです。
- 全部を解説したわけではないので注意
- 余裕があればソースコード周りの背景知識もまとめたい
- 何かあったら yukkyo12221222@gmail.com まで

Chapter 4

Training Linear Models

今回のポイント

- これまで
  - 勝手に解を見つけてくれた（fit で）
  - もしくはパラメータを指定した範囲やランダムで当てはめて良いモデルを探した
- でも実際は・・・
  - 最適化したい関数はそんなに簡単ではない
  - パラメータが多すぎ、範囲広すぎ
  - もしくはデータが大きすぎて公式で解けない
  - →　どのように解を見つけるのかを確認

- 今回は
  - より難しい関数（解の公式が無い場合等）等の解をどのように探すのか
  - 勾配法 (Gradient method)
    - 確率的勾配法 (Stochastic Gradient Descent)
    - ミニバッチ勾配法 (Mini-batch gradient descent)

- 他に扱うモデル
  - 多項回帰 (Polynomial regression)
  - 正規化モデル (Regularized models)
  - ロジスティック回帰 (Logistic regression)

Linear regression using the Normal Equation

```
# add x0 = 1 to each instance
X_b = np.c_[np.ones((100, 1)), X]
theta_best = \
    np.linalg.inv(X_b.T.dot(X_b)).dot(X_b.T).dot(y)
```

上記のコードは以下の式を実装している。
まとめて掛け算に入れるため `np.c_` で繋いでいる。

$$\theta_{best} = (\mathrm{X}_b^{\mathrm{T}} \times \mathrm{X}_b)^{-1} \times \mathrm{X}_b^{\mathrm{T}} \times y$$

線形回帰の Normal Equation（正規方程式）について

Linear regression using batch gradient descent

勾配法: ある地点からゴール（最適解）に向けて近づいていく方法の一つ

$$\theta_{t+1} = \theta_t - \eta \cdot \mathrm{gradients}(t)$$

$$\mathrm{gradients}(t) = \frac{1}{len(\mathrm{X}_b)} \cdot \mathrm{X}_b^{\mathrm{T}} \times (\mathrm{X}_b \times \theta_t - y)$$

- θ_t : t 番目(iteration t 回目）のパラメータ
- η : 更新率（ステップ幅とか）
- gradients : 勾配

Stochastic Gradient Descent

通常の勾配法（gradient descent）の場合、訓練データ $\mathbf{X}$ を一度にすべて使って更新していた

それに対し、確率的勾配法（Stochastic Gradient Descent）では、 $\mathbf{X}$ の中ランダムに選んだデータ $\mathbf{x}_i$ を使ってパラメータである θ を更新していく

- 局所最適解に落ちずらい
- 冗長な学習データがある場合勾配法より高速学習
- 学習データを収集しながら逐次敵に学習できる

Mini-batch gradient descent

- batch がまとめて更新
- stochastic が 1 つずつ更新
- Mini-batch では小さいバッチサイズを決めてそのサイズ個のデータを使ってパラメータを更新していく
  - batch と stochastic の間をとったようなもの

Polynomial regression

- 多項式（x^n とかが入っている）の学習と予測について
- 今回は下記の式を扱う（ e はランダムなエラー）

$$\frac{1}{2}x^2 + x + 2 + e$$

- `np.random.randn(m, n)` は m 行 n 列の乱数一覧を渡す（平均 0 分散 1）

`PolynomialFeatures` って何？

- 多項回帰式を求めるために、多項式次元に特徴ベクトルを写像している？
- 方針: 写像した後の空間で linear で回帰
- 高次の多項式だとクロスタームやらいっぱい出てきて書くのが大変
  - PolynomialFeature で特徴を作れば楽
- サンプルコードの例では、元のベクトルは 1 次元で、PolynomialFeatures の degree (設定した関数の次数)は 2 なので、$x -> (x, x^2)$ となっている
- TODO: 教科書の該当箇所を確認する

- もちろん予測したいときも、予測する特徴ベクトルを PolynomialFeature で写像する必要がある。

Q: plot_learning_curves は何をしている？

- 前の図の通り、今回は「二次関数」がベスト
  - トレーニングデータに fit すればいい訳ではない
  - 300 次は overfitting、1次は underfitting
- ではどうやって判断する？ → ① learning curve や ② parameter curve を見る
  - ① training samples を増やしたときの Accuracy を図にプロットしたもの
  - ② パラメータ変えたときの Accuracy

- leaning curve は何を見れば良いの？
  - 訓練データだけ精度が高くなっていないか？
  - 求める精度に対してどの程度精度が出ているか
  - training と test の精度の差が少ないと嬉しい
- parameter curve は何を見れば良いの？
  - どのパラメータをどのように調整すると過学習や、逆に学習しなさすぎるようになるのかを見ることができる
  - 正則化（後述）の強さを調整したりするのに使える
- ここでも扱っている 参考サイト

- 教科書の一枚目（1次）と二枚目（10次）、形似てるけど何が違うの？
  - training data に対するエラーが違う
  - また training curve と validation curve の gap が違う
    - 二枚目の方が gap が大きい → 過学習している可能性がある

Regularized models

- 過学習（Over fitting）を防ぐ方法
  - →　汎化性能（テストデータに対する性能）を上げたいため
- 代表的な方法として、L1 正則化(絶対値)や L2 正則化(二乗の値)
- 以下のようなモデルは正則化を予め含んでいる
  - リッジ回帰（Ridge Regression）
  - ラッソ回帰（Lasso Regression）
  - Elastic Net

- 各手法の正則化項（ペナルティ項）
  - $$\mathrm{Ridge}(\theta) = \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$
  - $$\mathrm{Lasso}(\theta) = \alpha \sum_{i=1}^{n} |\theta_i|$$
  - $$\mathrm{Elastic}(\theta) = r\alpha \sum_{i=1}^{n} |\theta_i| + \frac{1-r}{2} \alpha \sum_{i=1}^{n} \theta_i^2$$
- BGD = Batch Gradient Descent

- Lasso 回帰（L1 ノルム）はパラメータの一部を完全に0にする
  - モデルの推定と変数選択を同時に行える
  - → 疎（スパース、殆どが 0）な解になりやすい
- Ridge 回帰（L2 ノルム）は微分可能であり解析的に解けるが、L1ノルムは解析的に計算できない
- 開始点 $(0.25, -1)$ からのパラメータの推移を見ると、たしかに Lasso 回帰の場合は θ_2 が 0 になってからあまり変わろうとしていないことが分かる
- ソースコード詳細は省略（メインじゃないので）

Logistic regression

- いくつかの回帰アルゴリズムは分類にも使える
- ロジスティック関数は値が 0.5未満か以上かで0か1を返す二値分類などに使える

$$\mathrm{sigmoid}(t) = \sigma(t) = \frac{1}{1 + \exp(-t)}$$

- 図の例の場合「petal width」の値によってVirginia のか否かの確率を出力できている
- decision boundary = 決定境界
  - 図の例だと 1.6 あたりが境界になっている

- logistic_regression_contour_plot
  - contour = 等高線
  - 決定境界が等しい等高線（図だとただの直線だけど）を、決定境界を少しずつ変更してプロットしたもの
  - meshgrid でマス目を作れる
  - `plt.contour` で等高線が引ける

Softmax Regression

- logistic は二値分類
- logistic regression を組み合わせて複数のクラスを直接出力するモデルを「**Softmax Regression**」や「**Multinomial Logistic Regression**」と呼ぶ
- 出力例: (クラスAの確率, クラスBの確率, クラスCの確率) = (0.1, 0.7, 0.2)

数式は以下の感じ

$$p(k) = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp(s_k(\mathbf{x}))}{\sum_{j=1}^{K} \exp(s_j(\mathbf{x}))}$$

- K : クラスの数
- $\mathbf{s}(\mathbf{x})$: 入力 $\mathbf{x}$ の各クラスに対するスコアを表すベクトル
- $\sigma(\mathbf{s}(\mathbf{x}))_k$: 入力 $\mathbf{x}$ がクラス k に属する確率

Exercise

- Q1: めっちゃ特徴ベクトルが多い（100万個とか）ときどの線形回帰アルゴリズムを使うか？
  - SGD か Mini-batch GD、メモリに乗らないから
- Q2: トレーニングセット内の特徴間のスケールが違いすぎる。どのアルゴリズムが悩まされるか？どのように？
  - スケール違うとコスト関数が引き伸ばされたボウル状になる。よってGDの収束にめっちゃ時間かかる（勾配が小さすぎて）。解決するためにはデータをスケーリングする。

- Q3: 勾配法は線形回帰モデルにおいて局所解に陥るか？
  - 陥らない。コスト関数が凸状だから。
  - 凸状: 関数の2点を通る直線を引いたときに途中で関数を横切らない

- Q4: 全ての勾配法は、十分に長くイテレーションを回した場合同じモデル（パラメータ？）になるか
  - ならない。BGD や Mini-BGD はステップ幅が大きいと解の付近を行ったり来たりする可能性がある
- Q5: BGD 使って validation error 描いてみたら、一貫して上昇し続けた。どうなってるのこれ？どう直すの？
  - 解に近づいていない。学習率下げる

- Q6: Mini-batch GD（ミニバッチ勾配法）で validation error が上がったから止めたけどこれは良いアイデア？
  - 良くない。ランダムでデータ選んでるし、十分にイテレーション回した方が良い。
- Q7: 今回扱った勾配法の中で最適解の近傍（vicinity）に最も早くたどり着くのはどれ？実際に収束するの？どうやったら他のものはうまく収束しそう？
  - SGD。一回の更新で1データしか使わないから。実際に収束するのはGD。ほかは学習率を下げるべし。

- Q8: Polynomial Regression を使うとして、learning curve 描いたら training error と validatoin error に大きい差があったのだけれど何が起きてる？どのように解決する？3通り答えよ。
  - 過学習してる。
  - 以下の3通り
    - 次数減らす。正則化する。トレーニングデータ増やす。
- Q9:

- Q10: どうしてAよりBを使いたいのか説明せよ
  - 1: 線形回帰よりリッジ回帰
    - 正規化したいから
  - 2: リッジ回帰よりラッソ回帰
    - パラメータの次元が多いから
  - 3: ラッソ回帰より Elastic Net
    - 知らん

- Q11: outdoor/indoor、daytime/nighttime で画像分類したいとする。2つのロジスティック回帰分類器と1つのソフトマックス回帰分類器のどちらを実施しべきか
  - ２つは依存関係があるから、一緒にソフトマックスでやったほうがよさそう
- Q12: 早期ストップ機能を備えたバッチ勾配法でパラメータ更新を行うソフトマックス回帰を実装せよ（Scikit-Learn を使わずに）
  - そのうち記載する

今後の課題

- 等高線の書き方が謎すぎ
  - そのうち時間あるときに調べる
- Exercise の内容確認
  - 見る