scikit-learn、Keras、TensorFlow による実践機械学習

東京大学 工学部 3 年 阿部 慧人

2024 年 8 月 18 日

概要

インターンの課題図書の 4 章から 9 章の演習問題を自分で解いたものの答えを載せる。

1　4 章 モデルの訓練

1.1　問題 4.1

数百万個もの特徴量を持つ訓練セットに対して、有効な線形回帰訓練アルゴリズムは、バッチ GD、確率的 GD、ミニバッチ GD である。これらの勾配降下法は特徴量の数 n に対して、$O(n)$ でしか計算量が増加しない。一方で、正規方程式や SVD は、$O(n^2)$ 以上の計算量がかかる。

1.2　問題 4.2

特徴量のスケールが大きく異なる場合、勾配降下法を使うと時間がかかるというデメリットがある。これは、損失関数の等高線が平たい形になり直線的に最小値に移動できないことが原因である。

1.3　問題 4.3

ロジスティック回帰モデルの損失関数は、式 (1)(2) である。

$$J(\theta) = -\frac{1}{m}\Sigma[y^{(i)}log(p^{(i)}) + (1-y^{(i)})log(1-p^{(i)})] \tag{1}$$

$$wherep = \sigma(\boldsymbol{x}^T\theta) \tag{2}$$

この式から、$J(\theta)$ は、θ について凸関数とわかるので、極小値は唯一でそれを最小値とすればよい。

1.4　問題 4.4

十分な時間を与えても前提とするモデルの枠組みによっては、同じモデルに収束するとは限らない。具体的には、考えるモデルの損失関数の形による。極小値がただ一つであれば、その極小値に収束して、ただ一つの最適なモデルを得られる。しかし、極小値が複数あると局所的な解に陥り、必ずしも最適な回にたどり着くとはいえない。

1.5　問題 4.5

バッチ勾配降下法でエポックごとに検証誤差が大きくなっていると考えられるのならば、過学習が生じていると考えることができる。このような時には、検証誤差が最小をとっているところで早期打ち切りをすることが望ましい。

1.6　問題 4.6

ミニバッチ勾配降下法は、毎回の学習ステップで無作為に選んだインスタンスの集合を使って学習する。そのため、その損失関数は確率的な変動を常に含む。よって、検証誤差が少し上昇したとしても、それは確率的な変動によるものである可能性もあるため、過学習だと決めて早期打ち切りをするのは不適切。

1.7　問題 4.7

最も早く最適解近辺に辿り着くのは、確率的勾配降下法かバッチの小さいミニバッチ勾配降下法である。ともに各イテレーションで行う演算が少ないため、短い時間で終わる。一方で、収束はしない。これは、毎回確率的にインスタンスを選ぶことにより損失関数が上下するからである。学習が進むにつれて、学習率を下げれば、収束する。

1.8　問題 4.8

多項式回帰において、訓練誤差と検証誤差の間に大きな差があるというのは、過学習が生じていると考えられることはできる。解決する手法としては、以下の三つが考えられる。1 つが、モデルを正則化

するということ。Ridge 回帰や Lasso 回帰を使う。2 つ目が、多項式モデルの次数を下げること。次数が高いと、自由度が高く学習インスタンスに過剰に追従することがある。3 つ目が、学習インスタンスの数を増やすことである。

1.9　問題 4.9

Ridge 回帰を使っていて、訓練誤差と検証誤差がほとんと同じだが、非常に高い場合、バイアスが高いと考えられる。正則化パラメータ α を下げて、自由度を上げることで、バイアスを下げることができる。

1.10　問題 4.10

線形回帰ではなく、Ridge 回帰を使うべき理由は、線形回帰では自由度が高く過学習を引き起こすことが多いからである。

Ridge 回帰ではなく、Lasso 回帰を使うべき理由は、多くの場合意味のある特徴量が一部であり、Lasso 回帰は意味のある特徴量の係数以外は 0 にしてくれるので、意味のある特徴量のみ抽出できるから。

Lasso 回帰ではなく、Elastic Net を使うべき理由は、訓練インスタンスの数よりも特徴量の数の方が多い時や特徴量間に相関がある時、Lasso では不規則な動きをすることがあるから。

1.11　問題 4.11

2 つのロジスティック回帰分類機を作るべき。屋外・屋内と日中・夜間というクラスは相互排他的ではない。そのため、ソフトマックス回帰分類器のように多クラス出力できないものは役に立たない。

1.12　問題 4.12

Jupyter Notebook に掲載。

2　5 章 サポートベクトルマシーン

2.1　問題 5.1

サポートベクトルマシーンの基本的な考え方は、二つのクラスの分類する時にその境界として「太さを持った道」を通すことである。この道から最も近いインスタンスとの距離をマージンと呼ぶ。このマージンをできるだけ多くすると、性能が良い分類器になる。その一方で、すべてのインスタンスを完全に分類する道を考えると、マージンは小さくなる。この妥協点を見つけるのがポイント。

2.2　問題 5.2

サポートベクトルとは、サポートベクトルマシーン (SVM) の境界となる「道」の境界と内側に存在するインスタンスのこと。SVM の境界は、このサポートベクトルによってのみ決まり、それ以外のインスタンスは関係ない。

2.3　問題 5.3

SVM を使うときに、入力をスケーリングするのは SVM は特徴量のスケールの影響を受けやすいからである。スケーリングをした方が、決定境界の道が各特徴量を平等に考慮したものが出来上がる。

2.4　問題 5.4

インスタンスの分類するときの確信度を数値化する方法として、決定協会からインスタンスまでの距離をスコアとしたものが考えられる。しかし、これを直接確率に変換する方法はない。scikit-learn では、SVM クラスを使うときに、probability=True とすれば確率をロジスティック回帰によって計算するメソッドが追加される。

2.5　問題 5.5

訓練インスタンスの数 m に対して、計算量は主問題は $O(m)$ で、双対問題は $O(m^2)$ から $O(m^3)$ であるから種問題を解く方がよい。

2.6　問題 5.6

ガウス RBF カーネルを使った SVM で過小適合しているときは、γ を大きくした方が良い。γ を大きくすると、ベル曲線の幅が狭くなりより個別のインスタンスに適合するような形状になる。また、ハイパーパラメータ C についても、大きくするとマージン違反に厳しくなるため、よりインスタンスに適合する。

2.7　問題 5.7

ソフトマージン線形 SVM 分類器を QP ソルバーを使って解くことを考える。このとき各変数は以下

のように設定する。

$$\boldsymbol{H} = \begin{pmatrix} 0 & 0 & \dots & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & \dots & 0 \end{pmatrix} \tag{3}$$

$$\boldsymbol{f} = \begin{pmatrix} 0 & \dots & 0 & c & \dots & c \end{pmatrix}^{\mathrm{T}} \tag{4}$$

$$\boldsymbol{a'^{(i)}} = -t^{(i)} \times [1, \boldsymbol{x^{(i)}}] \tag{5}$$

を満たす $m \times (n+1)$ 行列 $\boldsymbol{A'}$ を考えた時、

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A'} & I_m \\ \boldsymbol{O} & -I_m \end{pmatrix} \tag{6}$$

$$\boldsymbol{b} = \begin{pmatrix} -1 & \dots & -1 & 0 & \dots & 0 \end{pmatrix}^{\mathrm{T}} \tag{7}$$

このようにすることで、QP の条件式の前半の m 個で、w に関する条件と後半の m 個で ζ に関する条件が得られる。

2.8 問題 5.8

Jupyter Notebook に掲載。

2.9 問題 5.9

Jupyter Notebook に掲載。

2.10 問題 5.10

Jupyter Notebook に掲載。

3 6 章 決定木

3.1 問題 6.1

深さの制限を設けないような決定木を構成するとき、それは Yes と No を繰り返して二分木になることが考えられる。よって、$log_2 m \approx 18$ となる。よって深さは、18 ほどだと想定される。

3.2 問題 6.2

一般に親ノードよりも子ノードの方がジニ不純度は小さくなる。これは、CART アルゴリズムでは子ノードのジニ不純度を小さくするように分割するようになっているから。しかし、これは常に成り立つわけではない。というのも、CART アルゴリズムは複数いる子ノードの不純度の加重平均が小さくなるようにしているだけだから。インスタンスを多く含む子ノードを純粋にすることで、インスタンスの少ない別の子ノードの不純度が親ノードのそれよりも上昇しても、結果としてその加重平均は親ノードよりも小さくなる。

3.3 問題 6.3

過学習をしているときは、正則化ハイパーパラメータのよって条件をより厳しくすることが望まれる。よって、max_depth という上限は下げると良い。

3.4 問題 6.4

入力特徴量の値を増やしても、スケーリングやセンタリングの影響を受けないから、意味はない。

3.5 問題 6.5

計算量は、$O(n \times mlog_2 m)$ でインスタンスの数 m に対して $mlog_2 m$ に比例する。そのため、インスタンスの数が 10 倍になれば、$10log_2 10m/log_2 m \approx 12$ 時間になる。

3.6 問題 6.6

インスタンスの数が数千未満のときは、presort することで速度は上がるが、インスタンスの数が多いときは速度がかなり下がる。

3.7 問題 6.7

Juyter Notebook に実施

3.8 問題 6.8

Juyter Notebook に実施

4 7 章 アンサンブル学習とランダムフォレスト

4.1 問題 7.1

それぞれの異なるモデルの予測の結果を多数決するアンサンブルを構成すれば性能の向上が望める。ここで大事なのは、それぞれのモデルが違うことである。もし可能であれば、それぞれのモデルが学習する訓練インスタンスも違うと、より良い結果が望める。

4.2 問題 7.2

ハード投票分類器は、単純にアンサンブルに使った異なる予測器の予測を集計して、その多数決を取りその結果をそのままアンサンブルの結果にする。一方で、ソフト投票分類器はアンサンブルに使った

異なる予測器のそれぞれのクラスに対する確率を集計して、それぞれのクラスについての和をとって、その和が最大のものを答えとすること。

4.3 問題 7.3

バギングアンサンブル、ペースティングアンサンブル、ランダムフォレストはそれぞれの予測器を独立に構成するため、それぞれの学習を別のサーバーで並列処理しても問題なく、処理スピードは上昇することが考えられる。ブースティングアンサンブルは、Ada-boosting であれば前回の予測器の誤り率や勾配ブースティングであれば前回の予測機の残差のように、前回の予測器の結果を使って次の予測器を構成する。そのため、ブースティングアンサンブルは予測器間に依存関係があるため、並列処理はできない。

スタッキングアンサンブルでは、一つ前の層の予測機の結果を使って次の層の予測器を構成する。そのため、同じ層の予測器は並列処理できるが、別の層は一つ前が終わるまで始められない。

4.4 問題 7.4

OOB 検証を使うと、検証セットをあらかじめ取り分けなくても、訓練セットのうち使われなかったもの、つまり OOB を使ってバイアスが少ない形で検証できる。そのため全てを訓練セットに使うことができて性能が上がる。

4.5 問題 7.5

Extra-Trees はランダムフォレストが使う特徴量をランダムに決めるのに加えて、その特徴量をどの閾値で分けるかも無作為に決めるという点でさらにランダム性がある。このようにすることで、バイアスは上がるが分散は下がるため、性能の向上を望める。また、決定木は最適な閾値を見つけるのに時間がかかるため、その過程を省略することで時間がかなり早くなると考えられる。

4.6 問題 7.6

アダブーストアンサンブルが過小適合指定いる場合は、推定器を増やすかそれぞれの推定器の正則化パラメータを下げる。また、学習率を上げることも役にたつ。

4.7 問題 7.7

勾配ブースティングアンサンブルが訓練セットに対して過学習を起こしているときは、個々の木の影響力を下げるため、学習率を下げると良い。また、予測器の台数が多すぎて汎化性能が下がっている可能性があるため、早期打ち切りを実施するのも有効だろう。

4.8 問題 7.8

Jupyter Notebook に掲載

4.9 問題 7.9

Jupyter Notebook に掲載

5 8 章 次元削減

5.1 問題 8.1

データセットの次元を削減する主要な理由は、データセットの次元が高いとデータの密度がかなり低くなってしまい、予測をする際に外挿を使う割合が増えて結果として、得られる予測器の信頼度が下がることを防ぐため。また単純に次元を減らすことで時間計算量や空間計算量を減らすことができる。さらに特徴量を減らすことで、重要な特徴量が何か見つけることができる。

一方で欠点として、次元削減によって重要な情報が落とされるとデータの信頼が下がる。学習パイプラインが複雑化すること、また次元削減後のデータの解釈がしづらいことが問題としてあげられる。

5.2 問題 8.2

「次元の呪い」とは次元が大きくなることで生じる様々な問題のことである。その問題のうち主要なものは、高次元になればなるほどデータが疎になることである。すると、過学習が生じやすいだけでなく、特徴量の中でパターンを見つけることが困難になり、学習の精度が下がる。

5.3 問題 8.3

一般には完全に戻すことはできない。例えば、PCA でも得られた主成分のうち d 個のものを使うことで、d 次元に落とし込み一部の情報が捨てられる。そのため、逆変換行列を使って戻す時もその情報を再現することができない。

5.4 問題 8.4

たとえ非線形なデータセットであったとしても、不要な次元を削除することができるから PCA は役に立つ。しかし、全く不要な次元がないデータセット（例えばスイスロール）に対して、PCA を実施すると必要な情報が失われるので、PCA は不適切ということになる。

5.5 問題 8.5

因子寄与率の主成分ごとの分布はデータセットによってまちまちである。そのため、1000 次元のデータであったら、1 から 950 次元のどの次元もとりいる。1 次元になる場合は、ある 1 つの主成分に分散が集中しているようなデータセットであるし、950 次元になる場合は全ての主成分に等しい分散があるようなデータセットである。

5.6 問題 8.6

通常の PCA は、特異値分解を使って全てのデータ群に対していっぺんに主成分を計算する。つまり、メモリが間に合わないと動作しない。逐次学習型 PCA は、データ数が多い時にデータをバッチごとに分割して実施するが、通常の PCA よりも動作速度は遅くなる。ランダム化 PCA は、次元を大きく削減したい時に使われる手法であり、厳密な特異値分解を実施しないため、大幅に早く実施できる。カーネル PCA は、非線形なデータセットにも役に立つ。

5.7 問題 8.7

データセットに対する次元削減アルゴリズムの性能評価は、主に 2 つ方法がある。1 つが、PCA 自体は教師なし学習でも全体のプロセスが教師あり学習であることが多いため、最終的な予測の精度をもとにグリッドサーチをすることができる。2 つ目が、完全に教師なしの場合は再構築誤差を使うという方法がある。次元削減の変換と逆変換を経て元のデータとどれくらいズレるかを確認することである。

5.8 問題 8.8

次元削減アルゴリズムを連続して実施することで、動作速度を大幅の早くできるというメリットがある。例えば、PCA を使ってざっくり次元削減をしてから、LLE を実施したとき、最初から LLE をやる場合と同じ精度の次元削減が望めるが、実行時間は大幅に減少する。

5.9 問題 8.9

Jupyter Notebook に実施。

ランダムフォレストアルゴリズムで、mnist データの分類をした。PCA の有無で動作速度と性能を比べたところ、PCA をすることによって学習時間が、177.15 秒から 262.82 秒に増えた。これは、次元削減をしたことによって逆に決定境界が複雑になってしまったことを示唆する。一方で、性能は 0.9674 から 0.9487 と微弱な低下に収まった。

ソフトマックス関数による分類器の場合、次元削減によって大幅に速度が上昇した。

5.10 問題 8.10

Jupyter Notebook に実施。

各次元削減の手法の性能と動作時間を比較した。その結果、PCA と t-SNE を組み合わせた手法が 10000 個の訓練データに対して 154.6 秒と最も速く、性能も良かった。他の手法、単体の PCA、LLE や MDS は、うまく mnist のデータを分類できていなかった。

6 9 章 教師なし学習のテクニック

6.1 9.1

参考文献

[1] オライリー・ジャパン 「scikit-learn、Keras、TensorFlow による実践機械学習」