

ラピッドチャレンジ「課題レポート」

佐藤晴一

2021 年 5 月 24 日

ビデオ視聴学習者 提出区分け	科目	章タイトル	1点100文字 以上で要点の まとめ	実装演習結果 キャプチャー 又はサマリー と考察	「確認デス ト」など、自 身の考察結果	演習問題や参 考図書、終了 課題など関連 記事レポート による加減
【d】 1つのURLで提出	深層学習day3 (基準点：15点)	Section1：再帰型ニューラルネットワークの概念	1点	1点	1点	1点
		Section2：LSTM	1点	1点	1点	1点
		Section3：GRU	1点	1点	1点	1点
		Section4：双方向RNN	1点	1点	1点	1点
		Section5：Seq2Seq	1点	1点	1点	1点
		Section6：Word2Vec	1点	1点	1点	1点
		Section7：Attention Mechanism	1点	1点	1点	1点
	深層学習day4 (基準点：6点)	Section1：強化学習	1点	不要	不要	不要
		Section2：AlphaGo	1点	不要	不要	不要
		Section3：軽量化・高速化技術	1点	不要	不要	不要
		Section4：応用モデル	1点	不要	不要	不要
		Section5：Transformer	1点	不要	不要	不要
		Section6：物体検知・セグメンテーション	1点	不要	不要	不要

- 講義動画視聴およびソース実装演習を確実に実施しているかを審査します。
- レポートが講義本筋に沿ってまとめているか。受講者自身の言葉で課題や気づきが記載されているか。
- 必須要件を満たさない場合、他の受講者のレポートのコピーなど不正が発覚した場合、差し戻しとします。
- 数式やコード演習結果の個々の間違いは審査に影響しません。なお、講師からのフィードバックはありませんのでご了承ください。
- 各科目以下の場合差し戻し
- 基準点以下
- 実装演習が2点以上不足（深層学習day3まで）

第 1 章

機械学習

コンピュータプログラムは、タスク T (アプリケーションにさせたいこと) を性能指標 P で測定し、その性能が経験 E (データ) により改善される場合、タスク T および性能指標 P に関して経験 E から学習すると言われている (トム・ミッチェル 1997)。また、人がプログラムするのは認識の仕方ではなく学習の仕方 (数学で記述) である。

機械学習に関するまとめサイトは、元産業総合技術研究所の赤穂氏に詳しい^{*1}。

■モデリングプロセス 機械学習モデリングプロセス (深層学習でも同じ) は以下の通り。

1. 問題設定
2. データ選定
3. データの前処理
4. 機械学習 モデルの選定
5. モデルの学習
6. モデルの評価

1.1 線形回帰モデル

■要点 (まとめ)

全 般

教師あり学習に分類され、予測タスクに適している。最小二乗法及び尤度最大化によりパラメータの推定、最適化を図る。モデルの選択・評価においては、ホールドアウト法^{*2}や交差検証法を用いる。

回帰問題は、ある入力 (離散あるいは連続値) から出力 (連続値) を予測する問題のこと。この際、直線で予測する場合を線形回帰。曲線で予測する場合を非線形回帰という。また、回帰で扱うデータ入力 (各要素を説明変数または特徴量と呼ぶ) は、 m 次元のベクトル ($m = 1$ の場合はスカラー)。出力 (目的変数) は、スカ

^{*1} <http://ibisforest.org/index.php?%E6%A9%9F%E6%A2%B0%E5%AD%A6%E7%BF%92>

^{*2} 有限のデータを学習用とテスト用の 2 つに分割し、「予測精度」や「誤り率」を推定する為に使用。学習用を多くすればテスト用が減り学習精度は良くなるが、性能評価の精度は悪くなる。逆にテスト用を多くすれば学習用が減少するので、学習そのものの精度が悪くなることになる。手元にデータが大量にある場合を除いて、良い性能評価を与えないという欠点がある。

基底展開法に基づく非線形回帰モデルでは、基底関数の数、位置、バンド幅の値とチューニングパラメータをホールドアウト値を小さくするモデルで決定する非線形回帰モデル

ラー値 (目的変数) となる。

データの分割とモデルの汎化性能測定

●データの分割

学習用データ：機械学習モデルの学習に利用するデータ

検証用データ：学習済みモデルの精度を検証するためのデータ

●なぜ分割するか

- モデルの汎化性能 (Generalization) を測定するため
- データへの当てはまりの良さではなく、未知のデータに対してどれくらい精度が高いかを測りたい

このように、学習用の検証用のデータを分けて、学習時の予測精度向上を向上させるとともに、あくまでも最終目標は未知のデータのインプットについての汎化性能を向上させることが重要である。

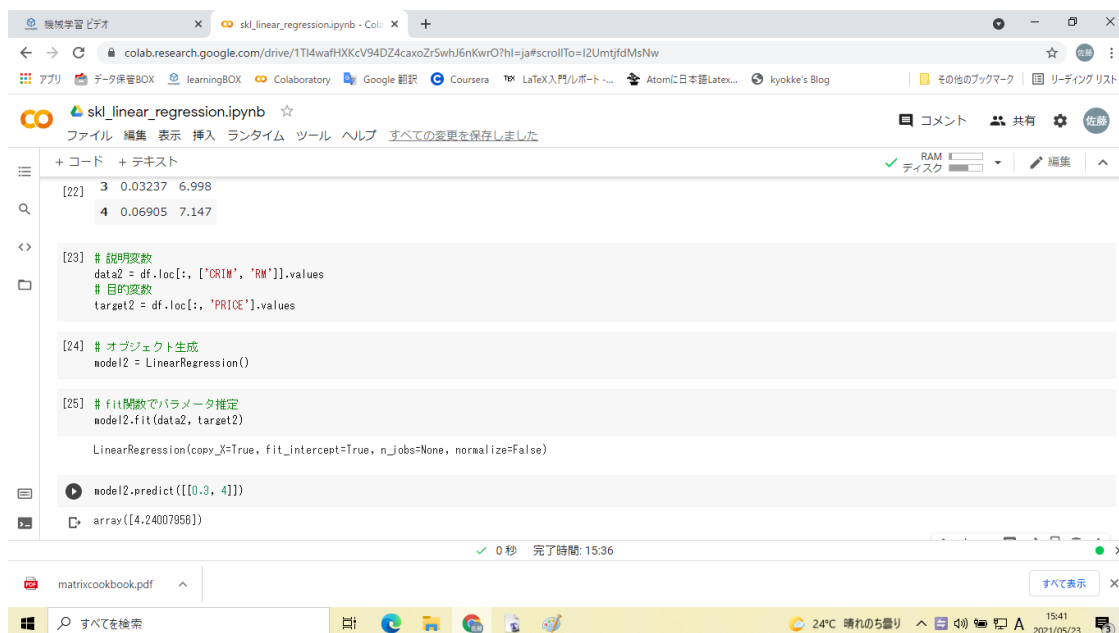
■実装演習成果 (キャプチャ、サマリー、考察)

設 定

「ボストンの住宅データセットを線形回帰モデルで分析」 適切な査定結果が必要。高すぎても安すぎても会社に損害がある。

課 題

部屋数が 4 で犯罪率が 0.3 の物件はいくらになるか？線形回帰モデルデータセット名ボストンの住宅データセット (ボストン市の住宅価格) レコード数 506 カラム数 14 詳細 UCI Machine Learning Repository: Housing Data Set 備考よく線形回帰モデルのテストセットとして使用される適切な査定。



```
[22] 3 0.03237 6.998
     4 0.06905 7.147

[23] # 説明変数
data2 = df.loc[:, ['CRIM', 'RM']].values
# 目的変数
target2 = df.loc[:, 'PRICE'].values

[24] # オブジェクト生成
model2 = LinearRegression()

[25] # fit関数でパラメータ推定
model2.fit(data2, target2)

LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

model2.predict([[0.3, 4]])

array([4.24007956])
```

Ans. 4.24(1000 ドル)

重回帰モデルの回帰係数と切片

$$\hat{y} = w_0 + w_1 x_{CRIM} + w_2 x_{RM} = -29.2 - 0.26 \times x_{CRIM} + 8.39 \times x_{RM}$$

1.2 非線形回帰モデル

■要点（まとめ）

全 般

教師あり学習に分類され、予測タスクに適している。最小二乗法及び尤度最大化によりパラメータの最適化を図る。モデルの選択・評価においては、ホールドアウト法や交差検証法^{*3}やグリッドサーチ^{*4}を用いる。特に複雑な非線形構造を内在する現象に対して、非線形回帰モデリングを実施する。一般的にデータの構造を線形で捉えられる場合は限られる。一方で、非線形な構造を捉えられる仕組みが必要となる。

未学習 (underfitting) と過学習 (overfitting)

学習データに対して、十分小さな誤差が得られないモデルのことを「未学習」という。対策としては、モデルの表現力が低いため、表現力の高いモデルを利用することがあげられる。

反対に小さな誤差は得られたものの、テスト集合（検証データ）誤差との差が大きいモデルのことを過学習という。対策としては、下記の3点があげられる。

1. 学習データの数を増やす
2. 不要な基底関数 (変数) を削除して表現力を抑止
3. 正則化法を利用して表現力を抑止

特に 2、3 はモデルの複雑さを調整する 2 つの方法としてポピュラーな手法である。

線形（非線形）回帰で得られたモデルが、データに対して未学習しているか過学習しているか以下の分類に分けられる。

1. 訓練誤差もテスト誤差もどちらも小さい⇒汎化しているモデルの可能性
2. 訓練誤差は小さいがテスト誤差が大きい⇒過学習
3. 訓練誤差もテスト誤差もどちらも小さくならない⇒未学習

^{*3} 個々のモデルの汎化性能を評価する手法であり、データを学習用と評価用に分割。次に検証データで各モデルの精度を計測する。この時、あるモデル 1 に各精度の平均を CV 値と呼ぶ (モデル 1 の汎化性能)。最初に分割した各モデルにおいてそれぞれ計算した CV 値が最大になるものを採用する。

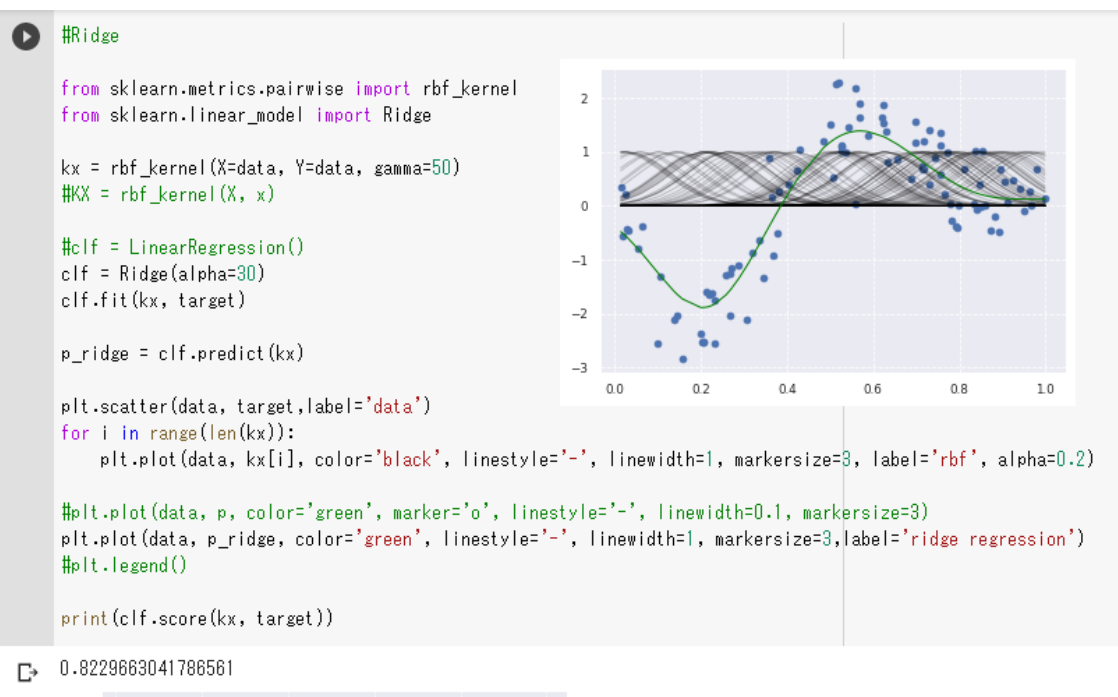
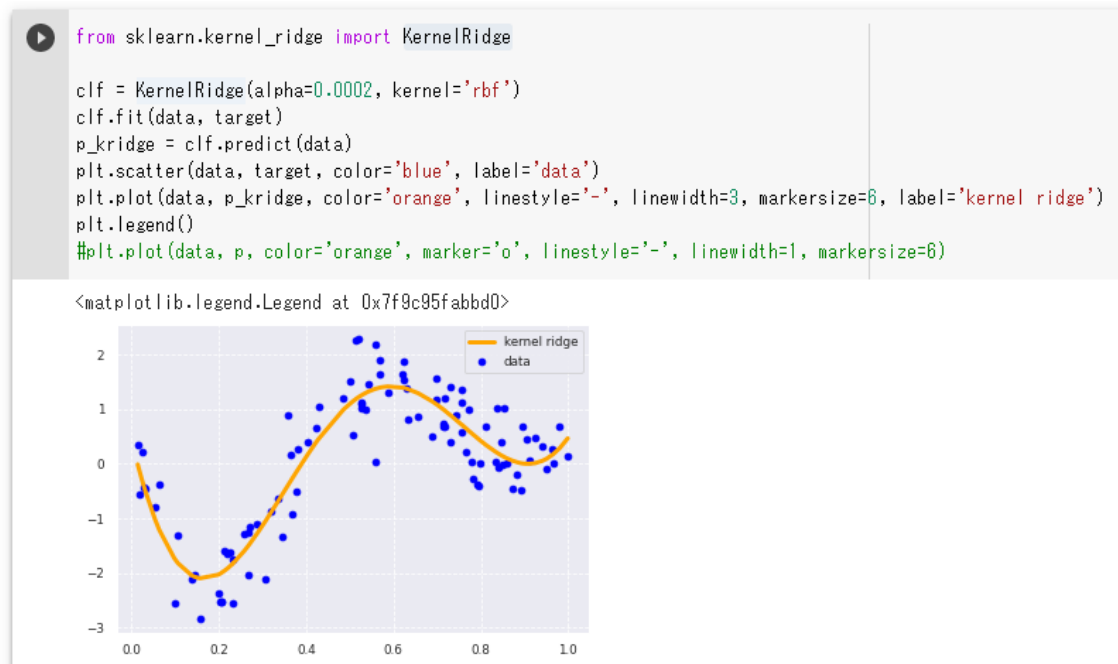
^{*4} 機械学習のハイパーパラメータ探索の方法であり、全てのチューニングパラメータの組み合わせで評価値を算出する。その結果、最も良い評価値を持つチューニングパラメータを持つ組み合わせを、「いいモデルのパラメータ」として採用する。

■実装演習成果（キャプチャ、サマリー、考察）

設 定

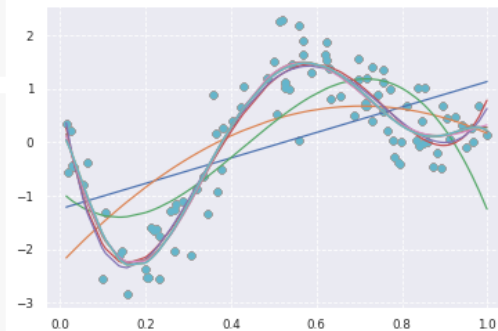
各種非線形関数を用いた回帰方法の実装とパラメータ依存性等の理解

キャプチャ



```
[11] from sklearn.preprocessing import PolynomialFeatures
      from sklearn.pipeline import Pipeline
```

```
▶ #PolynomialFeatures(degree=1)
  deg = [1,2,3,4,5,6,7,8,9,10]
  for d in deg:
      regr = Pipeline([
          ('poly', PolynomialFeatures(degree=d)),
          ('linear', LinearRegression())
      ])
      regr.fit(data, target)
      # make predictions
      p_poly = regr.predict(data)
      # plot regression result
      plt.scatter(data, target, label='data')
      plt.plot(data, p_poly, label='polynomial of degree %d' % (d))
```



サマリ

今回のポイントは、scikit-learn などを用いて非線形回帰の各種実装要領を学べたことにある。非線形関数を用いることでデータセットに対して過学習する恐れがあるので、リッジ、ラッソなどの正則化手法が有効であることが理解できた。回帰結果の精度評価については、必ず検証用データで実施すべきことは忘れてはならない。

1.3 ロジスティック回帰モデル

■要点（まとめ） 教師あり学習に分類され、分類タスクに適している。尤度最大化（最尤法）によりパラメータの最適化を図る。モデルの選択・評価においては、ホールドアウト法や交差検証法を用いる。有名なデータセットとしては、タイタニックや IRIS データなどがあげられる。

これまでの線形（非線形）回帰と異なる部分は、入力 x と m 次元パラメータ m の線形結合をシグモイド関数の入力として用いるところ。ここでシグモイド関数とは

$$\sigma = \frac{1}{1 + \exp(-ax)}$$

シグモイド関数の性質として、シグモイド関数の微分がシグモイド関数自身で表現することが可能であることがあげられる。この性質は、尤度関数の微分を行う際に利用することで計算が容易となり、連鎖律の計算を通常の乗算でプログラミングできる。

$$\frac{\partial \sigma(x)}{\partial x} = \frac{\partial}{\partial x} \left(\frac{1}{1 + \exp(-ax)} \right) = a\sigma(x)(1 - \sigma(x))$$

ここでロジスティック回帰では、シグモイド関数の出力を $Y = 1$ になる確率に対応させ、ベルヌーイ分布のパラメータの推定問題と捉える。この際、最適なパラメータ w を最尤推定により決定する。ここで $y_1 \sim y_n$ のデータが得られた際の尤度関数は次式で与えられる。

$$P(y_1, y_2, \dots, y_n; p) = \prod_{i=1}^n p^{y_i} (1 - p)^{1 - y_i}$$

いま、確率 p はシグモイド関数となるため、推定するパラメータは重みパラメータとなる。 $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ を生成するに至った尤もらしいパラメータを探すことになる。次式は

n 番目のデータの時の尤度関数

$$P(Y = y_n | x_n) = p^{y_n} (1 - p)^{1 - y_n} = \sigma(\mathbf{w}^T \mathbf{x}_n)^{y_n} (1 - \sigma(\mathbf{w}^T \mathbf{x}_n))^{1 - y_n}$$

つまり、 y_1, \dots, y_n までデータが得られた際の尤度関数は、

$$P(y_1, y_2, \dots, y_n | w_1, w_2, \dots, w_m) = \prod_{i=1}^n p^{y_i} (1 - p)^{1 - y_i} = \prod_i \sigma(\mathbf{w}^T \mathbf{x}_i)^{y_i} (1 - \sigma(\mathbf{w}^T \mathbf{x}_i))^{1 - y_i} = \mathbf{L}(\mathbf{w})$$

となり、上式が最大となるパラメータ \mathbf{w} を推定することになる。

ここから極値（微分による計算）を求めることを容易化するために $\mathbf{L}(\mathbf{w})$ の対数を取り、その微分計算を試みることを考える。これにより、同時確率の積が和に変換可能となり、指数が積の演算に変換可能となる。さらに、対数尤度関数が最大になる点と尤度関数が最大になる点は同じ^{*5}であるため、「尤度関数にマイナスをかけたものを最小化」し、「最小 2 乗法の最小化」と合わせることにする。その結果、ロジスティック回帰で求めるべき、尤度関数は次式で与えられる。

$$\mathbf{E}(w_1, w_2, \dots, w_m) = -\log \mathbf{L}(w_1, w_2, \dots, w_m) = -\sum_{i=1}^n \{y_i \log p_i + (1 - y_i) \log(1 - p_i)\}$$

勾配降下法 (Gradient descent)

ここから解析的に尤度関数の最小値を求めるわけだが、勾配降下法 (Gradient descent) と呼ばれる反復学習によりパラメータを逐次的に更新するアプローチの一つを採用する。

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta \frac{\partial \mathbf{E}}{\partial \mathbf{w}} = \mathbf{w}^{(k)} + \eta \sum_{i=1}^n (y_i - p_i) \mathbf{x}_i$$

ここで η は学習率と呼ばれるハイパーパラメータでモデルのパラメータの収束しやすさを調整するものである^{*6}。このように勾配降下法では、パラメータを更新するのに n 個全てのデータに対する和を求める必要がある。したがって、データ数 n が巨大になった時にデータをオンメモリに載せる容量が足りなくなったり、計算時間が莫大になるなどの問題がある。このような状況を回避するために、次に説明する確率的勾配降下法を利用して解決する。

確率的勾配降下法 (SGD)

確率的勾配降下法^{*7}は、データを一つずつランダムに（「確率的」に）選んでパラメータを更新する手法である。このため、勾配降下法でパラメータを 1 回更新するのと同じ計算量でパラメータを n 回更新できるので効率よく最適な解を探索可能となる。

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} + \eta (y_i - p_i) \mathbf{x}_i$$

このように毎回の更新でデータを 1 つ（又は少量）しか見ない。

次に学習済みの「ロジスティック回帰モデル」の性能を測る指標について見ていく。以下の表のように 4 つの区分で性能指標を評価する^{*8}。

^{*5} 対数関数は単調増加（ある尤度の値が $x_1 < x_2$ の時、必ず $\log(x_1) < \log(x_2)$ となる）

^{*6} <https://qiita.com/masatomix/items/d4e5fb3b52fa4c92366f>。線形回帰モデル（最小 2 乗法）は、MSE のパラメータに関する微分が 0 になる値を解析に求めることが可能である。一方、ロジスティック回帰モデル（最尤法）は、対数尤度関数をパラメータで微分して 0 になる値を求める必要があるのだが、解析的にこの値を求めることは困難である。

^{*7} http://ibis.t.u-tokyo.ac.jp/suzuki/lecture/2018/kyoto/Kyoto_02.pdf

^{*8} <https://www.codexa.net/ml-evaluation-cls/>

表 1.1 混同行列 (confusion matrix)

		検証用データ	
		Positive	Negative
予測結果	Positive	真陽性 (True Positive, TP)	偽陰性 (False Negative, FN)
	Negative	偽陽性 (False Positive, FP)	真陰性 (True Negative, TN)

正解率 (Accuracy)

正解率とは、その名の通り「全ての予測のうち、正解した予測の割合」を指す。非常にシンプルで直感的にも解釈しやすい指標です。しかし、機械の異常検知など、ほとんどが Negative で稀に Positive が出現するような、偏りが大きいデータを扱う場合にはあまり有効な評価指標とは言えないことから適用にあたっては注意を要する。

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

適合率 (Precision)

適合率は、「陽性と予測したもののうち、実際に陽性であるものの割合」を表す指標。適合率は、FP を小さくすることを重視する指標なので、誤って陽性と判断しては困る場合に用いると有効。一方、適合率の弱点は陰性の予測を無視している点。つまり、どれだけ偽陰性の予測を出しても、適合率には反映されないことから、偽陰性が多いことが問題になるケースでは、用いないほうがよい。

$$Precision = \frac{TP}{TP + FP}$$

再現率 (感度, Recall, True Positive Rate, TPR)

再現率は、「実際に陽性であるもののうち、正しく陽性と予測できたものの割合」です。再現率は、適合率と対照的な指標で、FN を小さくすることを重視する指標となっている。つまり、誤って陰性と判断しては困る場合に用いられます。また、再現率は適合率と対照的な指標であるため、適合率の長所が再現率の短所となります。つまり、再現率は偽陽性 (FP) を無視する指標であるため、やたらめったら陽性判定をしてたくさん偽陽性を出したとしても、再現率が下がることはないため、偽陽性が多いと困る場合には重視しない方が賢明と言える^{*9}。

$$Recall = \frac{TP}{TP + FN}$$

F 値 (F-measure, F1 値, F1-measure)

F 値は対照的な特徴を持つ適合率と再現率の調和平均です。簡単に言えば、対照的な両者の特徴をバランスよく含んでいる指標と言える。分子が適合率と再現率の積になっているため、片方が極端に低い場合に、正しく低い評価をつけることができる。言い換えれば、偽陽性や偽陰性が極端に多い場合には、値が小さくなるため、どちらの指標もそれなりに高い値である必要があります。値の範囲はこれまでの指標と同様に 0 から 1 で、1 に近づくほど予測性能がいいということになる。

$$F - measure = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

^{*9} PCR 検査の感度 (= 再現率、実際にウイルスに感染している人のうち、検査で陽性と判定される割合) として一番よい値になるのは、「感染から 8 日目 (症状発現の 3 日後) の 80 %」とのことです。すなわち、最も感染者を検出しやすいタイミングでも、感染者のうち 20 % は陰性と判定してしまうことになる。

■実装演習成果（キャプチャ、サマリー、考察）

設 定 タイタニックの乗客データを利用しロジスティック回帰モデルを作成

課 題 年齢が 30 歳で男の乗客は生き残れるか

キャプチャ

The first screenshot shows the initial data exploration. The notebook title is 'skl_logistic_regression.ipynb'. The first cell contains the code `titanic_df.head()`, which displays the first five rows of the dataset. The columns are: Survived, Pclass, Sex, Age, SibSp, Parch, Fare, Embarked, and AgeFill. The data shows that for the first row, a 22-year-old male survived, while for the last row, a 35-year-old male did not survive. The second cell contains the code `data1 = titanic_df.loc[:, ["AgeFill"]].values`, which extracts the 'AgeFill' column into a NumPy array. The third cell contains the code `data1 = titanic_df.loc[:, ["AgeFill"]].values`, which is a duplicate of the second cell. The second screenshot shows the model training and prediction. The first cell contains the code `from sklearn.linear_model import LogisticRegression`. The second cell contains the code `model=LogisticRegression()`. The third cell contains the code `model.fit(data1, label1)`, which triggers a `DataConversionWarning` from sklearn. The fourth cell contains the code `model.predict([[30]])`, which returns the array `array([0])`. The fifth cell contains the code `model.predict_proba([[30]])`, which returns the array `array([[0.61755823, 0.38244177]])`. The notebook interface includes a file explorer on the left, a top bar with 'ファイル', '編集', '表示', '挿入', 'ランタイム', 'ツール', and 'ヘルプ', and a bottom status bar showing '0秒' and '完了時間: 13:18'.

サマリー

今回のポイントは、タイタニックデータの年齢欠損値を既存データの中央値として再度設定したこと。それによってもって、データセットの column データの呼び名を”Age”から”AgeFill”を参照させることでデータを読み込み scikit-learn の logistic function に食わせることが可能となる。ロジスティック回帰の結果、年齢が 30 歳の乗員は”死亡”という結果が得られた。割合は「生存 (38%)」、「死亡 (62%)」

1.4 主成分分析 (PCA)

教師なし学習に分類され、次元削減タスクに適している。分散最大化によりパラメータの最適化を図る。

■要点 (まとめ)

多変量データの持つ構造をより少数個の指標に圧縮すること。一方で、変量の個数を減らすことに伴う、情報の損失はなるべく小さくしたいという要求を伴う。例えば、少数変数を利用した分析や可視化 (2・3 次元の場合) が、実現可能となる。主成分軸は、情報の量を分散の大きさと捉えて、線形変換後の変数の分散が最大となる射影軸を探索すれば良い。

計算は、目的関数である分散が制約条件 (係数ベクトルのノルムが 1) を満たしつつ最大となるものを解くことになる。ここで制約条件下の極値を求める数学的手法としてラグランジュの未定乗数法があり、PCA においても適用する。ここまで整理すると下記の固有値問題を解くことと同値になる。

$$\text{Var}(\bar{X})a_j = \lambda a_j$$

■実装演習成果 (キャプチャ、サマリー、考察)

設 定

- 乳がん検査データを利用しロジスティック回帰モデルを作成。
- 主成分を利用し 2 次元空間上に次元圧縮

課 題 32 次元のデータを 2 次元上に次元圧縮した際に、うまく判別できるかを確認

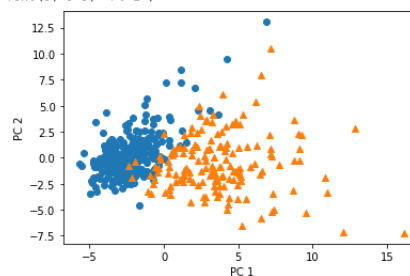
キャプチャ

```
# PCA
# 次元数2まで圧縮
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_scaled)
print('X_train_pca shape: {}'.format(X_train_pca.shape))
# X_train_pca shape: (426, 2)

# 寄与率
print('explained variance ratio: {}'.format(pca.explained_variance_ratio_))
# explained variance ratio: [ 0.43315126  0.19588506]

# 散布図にプロット
temp = pd.DataFrame(X_train_pca)
temp['Outcome'] = y_train.values
b = temp[temp['Outcome'] == 0]
m = temp[temp['Outcome'] == 1]
plt.scatter(x=b[0], y=b[1], marker='o') # 良性は○でマーク
plt.scatter(x=m[0], y=m[1], marker='^') # 悪性は△でマーク
plt.xlabel('PC 1') # 第1主成分をx軸
plt.ylabel('PC 2') # 第2主成分をy軸
```

X_train_pca shape: (426, 2)
explained variance ratio: [0.43315126 0.19588506]
Text(0, 0.5, 'PC 2')



サマリ

上図のように 2 次元に次元圧縮した結果、それぞれの主成分の分散の寄与率は、0.43 及び 0.20 であることから元の 32 次元データの約 7 割 (70%) を再現できていることを示している。

1.5 アルゴリズム

1.5.1 k 近傍法 (k-Nearest Neighbor method, kNN)

■要点 (まとめ)

教師あり学習に区分され、分類問題のための機械学習手法である。手法は簡単であり、新しいデータ点に対して最近傍のデータを k 個取ってきて、それらがもっとも多く所属するクラスに識別するだけである。もちろん、 k の数を変化させれば、識別結果も異なる。特に、 $k = 1$ の場合を最近傍法と呼ぶ。また、一般的に、 k の値を大きくすることで、クラス識別の決定境界は滑らかになっていく。

■実装演習成果 (キャプチャ、サマリー、考察)

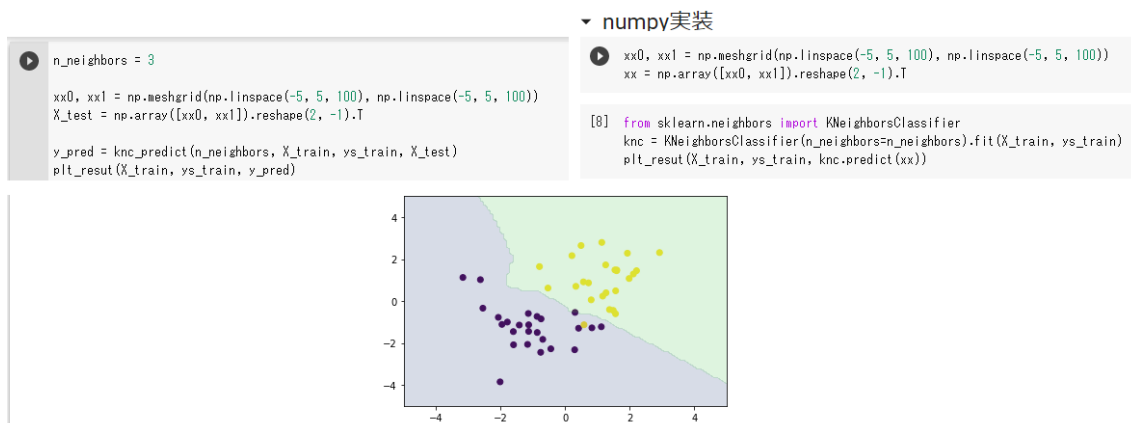
設 定

人口データを分類

課 題

人口データと分類結果をプロットしてください

キャプチャ



サマリ

最近傍法における k の数を変化させて、分類の景況及び決定境界について外観したが、講義資料で説明のあった通り、 k の数を増加させることで決定境界が滑らかになることが確認できた。

1.5.2 k 平均法 (k-means)

■要点 (まとめ)

教師なし学習に区分され、クラスタリングのための機械学習手法である。与えられたデータを k 個のクラスに分類するものである。ここでクラスタリングとは、特徴の似ている者同士をグループ化することである。

k 平均法 (k-means) のアルゴリズム

1. 各クラスタ中心の初期値を設定する*¹⁰
2. 各データ点に対して、各クラスタ中心との距離を計算し、最も距離が近いクラスタを割り当てる
3. 各クラスタの平均ベクトル (中心) を計算する
4. 収束するまで 2, 3 の処理を繰り返す

■実装演習成果 (キャプチャ、サマリー、考察)

設定 ワインデータをクラスタリング

課題 ワインデータのクラスタリング結果を求めよ

キャプチャ



サマリ

本実装で用いたワインデータは、13 個の特徴量があった。前処理として正規化や標準化の他、次元削減により、例えば 2 次元まで落とすと直感的にクラスタリングの様子がわかりやすくなるとのこと。また、クラスタリング数は、分析する前に決定する必要があるが、エルボー法と呼ばれる手法により、適切なクラスタリング数が評価できる。より詳しいは、考察記事*¹¹に詳しい (Jupyter notebook 含む)。

*¹⁰ 初期値同士が近い場合、うまくクラスタリングされない場合もあるので注意。そのほか、 k の値を変えると最終的なクラスタリング結果も当たり前だが変わる。

*¹¹ <https://obgynai.com/unsupervised-learning-machine-learning/>

1.6 サポートベクターマシン (SVM)

■要点 (まとめ)

教師あり学習に区分され、分類タスクに適している。マージン最大化によりパラメータの最適化を図る。モデルの選択・評価においては、ホールドアウト法や交差検証法を用いる。

2 クラス分類のための機械学習手法であり、線形モデルの正負で2値分類する。

$$y = \mathbf{w}^T \mathbf{x} + b = \sum_{j=1}^m w_j x_j + b$$

サポートベクターマシンでは、上式で得られた数値 (実数) 次の符号関数で出力 (分類) する。

$$\text{sign}(y) = \begin{cases} +1 & (y > 0) \\ -1 & (y < 0) \end{cases}$$

実際の実装アルゴリズムとしては、線形判別関数 $\mathbf{w}^T \mathbf{x} + b = 0$ の直線と最も近いデータ点との距離をマージンと呼び、このマージンが最大となるように線形判別関数^{*12}を求めることになる。

SVM の主問題

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2, \quad t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 \quad (i = 1, 2, \dots, n)$$

上式の最適化問題をラグランジュ未定乗数法で解くことになる。

ソフトマージン SVM サンプルを線形分離できないとき、誤差を許容し誤差に対してペナルティを与える手法である。この時、マージン内に入るデータや誤分類されたデータに対して誤差を表す変数 ξ を導入する。

ソフトマージン SVM の目的関数と制約条件

$$\min_{\mathbf{w}, b} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i, \quad t_i(\mathbf{w}^T \mathbf{x}_i + b) \geq 1 - \xi_i \quad (i = 1, 2, \dots, n; \quad \xi_i \geq 0)$$

ここで第2項は、誤差に対するペナルティである。 C は、トレードオフを制御するハイパーパラメータである。このように、ソフトマージン SVM は、線形分離できない場合でも対応でき、ハイパーパラメータ C の大小で決定境界が変化する。一般的に

$$\begin{cases} C \text{ が小さい時} \dots \dots \text{誤差をより許容する} \\ C \text{ が大きい時} \dots \dots \text{誤差をあまり許容しない} \end{cases}$$

次に、取得したデータセットがソフトマージン SVM でもうまく線形分離できないときは、非線形分離を考える。手法は簡単であり、単純に特徴空間に写像 $\phi(\mathbf{x})$ し、その空間で線形に分離するだけである。

つまり、当初得られたデータセットが、線形に分離できないときは、特徴空間に写像 $\phi(\mathbf{x})$ し、特徴空間上で線形に分離可能だとするとき、もとの空間上では非線形な分離が実現する仕組みである。

■実装演習成果 (キャプチャ、サマリー、考察)

^{*12} マージンをパラメータに依存する関数として表現

設定 練習用データ（線形分離、非線形分離、重なり合い）を分類

課題 SVM の各手法を用いて分類した結果を求めよ

キャプチャ

サポートベクターマシン(SVM)

```
[9] %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

訓練データ生成①（線形分離可能）

```
[7] def gen_data():
    x0 = np.random.normal(size=50).reshape(-1, 2) - 2.
    x1 = np.random.normal(size=50).reshape(-1, 2) + 2.
    X_train = np.concatenate([x0, x1])
    y_train = np.concatenate([np.zeros(25), np.ones(25)]).astype(np.int)
    return X_train, y_train
```

```
[10] X_train, y_train = gen_data()
```

訓練データ生成②（線形分離不可能）

```
factor = .2
n_samples = 50
linspace = np.linspace(0, 2 * np.pi, n_samples // 2 + 1)[:-1]
outer_circ_x = np.cos(linspace)
outer_circ_y = np.sin(linspace)
inner_circ_x = outer_circ_x * factor
inner_circ_y = outer_circ_y * factor

X = np.vstack((np.append(outer_circ_x, inner_circ_x),
                np.append(outer_circ_y, inner_circ_y))).T
y = np.hstack([np.zeros(n_samples // 2, dtype=np.intp),
               np.ones(n_samples // 2, dtype=np.intp)])
X += np.random.normal(scale=0.15, size=X.shape)
x_train = X
y_train = y
```

```
[16] plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
```

ソフトマージンSVM

訓練データ生成③（重なりあり）

```
[ ] x0 = np.random.normal(size=50).reshape(-1, 2) - 1.
x1 = np.random.normal(size=50).reshape(-1, 2) + 1.
x_train = np.concatenate([x0, x1])
y_train = np.concatenate([np.zeros(25), np.ones(25)]).astype(np.int)
```

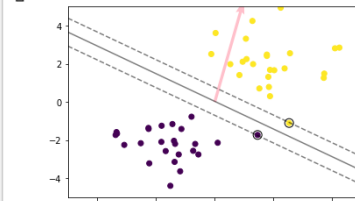
```
plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
```

```
<matplotlib.collections.PathCollection at 0x7f49d2fb36d0>
```

```
# マージンと決定境界を可視化
plt.contour(xx0, xx1, y_project.reshape(100, 100), colors='k',
            levels=[-1, 0, 1], alpha=0.5, linestyle=['--', '-', '--'])
```

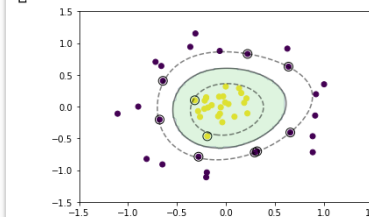
```
# マージンと決定境界を可視化
plt.quiver(0, 0, 0.1, 0.35, width=0.01, scale=1, color='pink')
```

```
<matplotlib.quiver.Quiver at 0x7fd73cc39ed0>
```



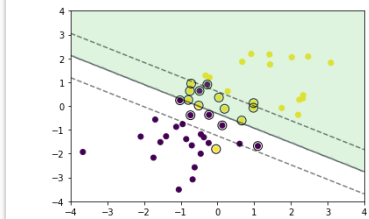
```
plt.contourf(xx0, xx1, y_pred.reshape(100, 100), alpha=0.2, levels=np.linspace(0, 1, 3))
# マージンと決定境界を可視化
plt.contour(xx0, xx1, y_project.reshape(100, 100), colors='k',
            levels=[-1, 0, 1], alpha=0.5, linestyle=['--', '-', '--'])
```

```
<matplotlib.contour.QuadContourSet at 0x7fd73492e390>
```



```
plt.scatter(support_vectors[:, 0], support_vectors[:, 1],
            s=100, facecolors='none', edgecolors='k')
# 領域を可視化
plt.contourf(xx0, xx1, y_pred.reshape(100, 100), alpha=0.2, levels=np.linspace(0, 1, 3))
# マージンと決定境界を可視化
plt.contour(xx0, xx1, y_project.reshape(100, 100), colors='k',
            levels=[-1, 0, 1], alpha=0.5, linestyle=['--', '-', '--'])
```

```
<matplotlib.contour.QuadContourSet at 0x7fd73483da50>
```



サマリ

直感的にわかりやすい線形分類は、図で示されたとおりサポートベクトルとなる点との境界線との距離（マージン）が最大となるように分類されている。さらに、非線形分離では、rbf カーネルを用いてガウシアンの特徴空間に一度写像し、特徴空間上で分類し、元の空間に戻すことで分類が実行できることを確認した。最後に、ソフトマージン SVM では、マージン内にデータ点が重なり合うような場合でもそれを許容することで分類が行えることを確認した。また、ハイパーパラメータ C を大きくすることで、許容を許さないハードマージン SVM に近づくことを実装により確認した。

参考文献

- [1] 奥村晴彦, 黒木裕介 『L^AT_EX 2_ε 美文書作成入門 第7版』(技術評論社, 2017)
- [2] 加藤公一, 『機械学習のエッセンス』(SBクリエイティブ, 2018)
- [3] 大重美幸, 『Python 3 入門ノート』(ソーテック, 2018)
- [4] 大関真之, 『機械学習入門』(オーム, 2018)
- [5] Andreas C. Müller et al., 『Python ではじめる機械学習』(オライリージャパン, 2018)
- [6] 加藤公一 (監修), 『機械学習図鑑』(翔泳社, 2019)
- [7] 岡谷貴之, 『深層学習』(講談社, 2015)
- [8] 竹内一郎, 『サポートベクトルマシン』、講談社, 2015)
- [9] 斎藤康毅, 『ゼロから作る Deep Learning』(オライリージャパン, 2019)
- [10] Guido van Rossum, 『Python チュートリアル (第4版)』(オライリージャパン, 2021)