令和元年度　特別研究報告書

野鳥の鳴き声識別における特徴量類似性と認識精度に関する研究

龍谷大学　理工学部　情報メディア学科

T160413　古江 智瑛

指導教員　三好 力　教授

内容概要

機械学習における教師あり学習では人間がラベルをつけたラベル付きデータを多数学習に用いるほど識別率が高くなることが知られている．しかし，多くのラベル付きデータを用意することは専門家の知識や人の手間がかかり，高コストである．このコストを削減する識別器の性能を向上させることは機械学習において重大な課題の一つである．ラベル付きデータが高コストである一方で，ラベルなしデータの場合は低コストで大量に獲得できる場合が多い．たとえば鳥の鳴き声の音声データであれば森の中に録音機を設置するだけで容易に獲得できる．

そこで本研究では特徴ベクトル間の類似性に着目し，少数のラベル付きデータから多数のラベルなしデータのクラスを特徴ベクトル間距離によって決定して訓練データとして用いる手法を検討する．ニュージーランドに生息する野鳥の鳴き声データを例に，SVMにおいて訓練データの数と識別率の推移の関係を検討するための実験を行った．

目次

1 はじめに

機械学習における教師あり学習では，人間がラベルをつけたラベル付きデータを多数学習に用いるほど識別率が高くなることが知られている．しかし，多くのラベル付きデータを用意することは専門家の知識や人の手間といったコストがかかってしまう．このコストを削減し，分類器の性能を向上させることは機械学習において重大な課題の一つである．ラベル付きデータが高コストである一方で，ラベルなしデータの場合は専門家の知識や人の手間などは必要とせず低コストで大量に獲得できる場合が多い．たとえば鳥の鳴き声の音声のデータであれば，森の中に録音機を設置するだけで容易に獲得できる．教師あり学習の多数のデータを学習に用いることで識別率が高くなるという点に着目し，もしも少数のラベル付きデータを用いて多数のラベルなしデータにラベル付けを行って学習に用いることができれば，低コストでかつ性能の高い識別器となることが期待できる．

そこで本研究では機械学習におけるデータの類似性がデータの特徴ベクトル間の距離に対応することに着目し，少数のラベル付きデータから多数のラベルなしデータのクラスを特徴ベクトル間距離によって決定し，訓練データとして用いる手法を検討する．具体的にはクラス y が既知の少数のラベル付きデータの特徴量とする中心点それぞれからの距離を測定し，その距離が近いものをラベル付きデータに類似したデータとみなしてラベルなしデータのクラス y を決定し，訓練データとして用いる．特徴量間の距離尺度についてバタチャリヤ距離の平方根など多くの距離が提案されているが，本研究では最も広く使われており，直感的に理解がしやすいユークリッド距離を距離尺として用いる．ニュージーランドに生息する野鳥の鳴き声データを例に，SVM をメイン手法とし，他の手法との訓練データの数と識別率の推移の関係を検討するための実験を行った．

2 研究概要

2.1 既存技術

藤岡らの先行研究[1]では，SVM において類似性を特徴ベクトル間のユークリッド距離によって決定している．ラベル付きデータからランダムで少数のデータを選び取り，その選んだ各々のデータの平均ベクトルを中心として，そこからのユークリッド距離を全てのデータについて測定し，定めた距離内のもの全てを少数のラベル付きデータと同じラベルを付けて訓練データとする．この手法を用いた実験では，鳴き声の種類が少なく距離が大きいと識別率が高くなり，鳴き声の種類が多い鳥は識別率が安定しない，と報告している．

2.2 基本事項

2.2.1 SVM について

Support Vector Machine(SVM)とは[2-6],V.vapnik によって，V.vapnik によって発表された教師あり学習を用いるパターン認識の手法の一つであり，特に2クラス分類問題において優れた性能を示す学習モデルである．マージン最大化によって未知のデータに対する汎化性能が優れていることでも知られており，様々パターン認識問題に利用されている．しかし，2クラス分類問題において高い性能を示すことに対して多クラス分類問題にはそのまま対応できず，計算量が多い等の問題点も指摘されており，一概に全てのパターン認識手法と比較して優れていると言える訳ではない．1963 年に線形 SVM が発表され，さらに 1992 年に非線形に拡張されたことにより，線形 SVM と非線形 SVM の2つに分類される．本研究では線形 SVM を用いている．

2.2.2 SVM のアルゴリズム

SVM はクラス $y_i \in \{-1,1\}$に属する学習データ $x_i \in R^d$ 線形分離可能な場合，分離平面とベクトルとの距離（マージン）が最大になるような分離平面を構成する．ここで y_i はデータ i が正例か負例かを表すクラス，x_i はデータ i の特徴ベクトルである．特徴空間が 2 次元である場合の例を図 2.1 に示す．

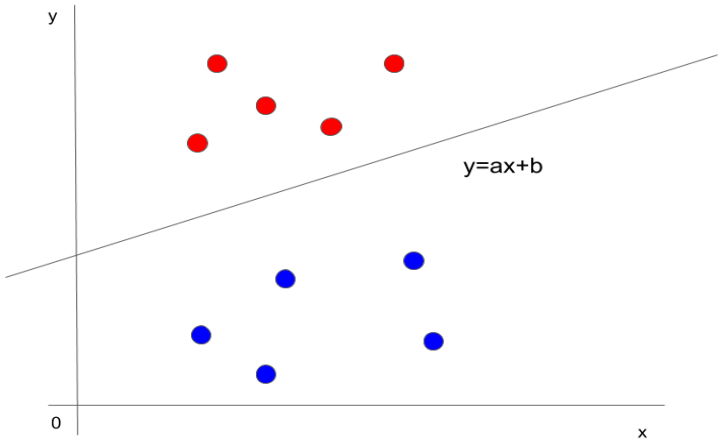


図 2.1 SVM の 2 クラス分類による分離平面

求める境界は以下の式で表される.

$$f(x) = x^t x + w_0 \tag{2.1}$$

上記の識別境界面 f(x)=0 からある点 x_i までの距離|r|は

$$|r| = \frac{f(x_i)}{||w||} \tag{2.2}$$

によって求めることができる.ある点 x_i について y_i が f(x)>0 の場合は正 f(x)<0 の場合は負になるように分類するとき,正しく分類できた場合は y_i=1,もしくは y_i=−1 をとるので,$f(x)y_i$>0 が成り立つ.このことから,正しく分類できた場合の点 x_iから,境界面までの距離は以下の式で表される.

$$|r| = \frac{y_i(x^t x + w_0)}{||w||} \tag{2.3}$$

このとき,wを定数倍しても境界面との距離の値は不変であり最適化問題には影響がないため,境界面に最も接近する点x_i について,次式が成立する.

$$y_i(w^t x_i + x_0) = 1 \tag{2.4}$$

境界に最も近い点について式(2.4)が成立していることにより,それ以外の全ての点について次式が成立する.

$$y_i = (w^t x_i + w_0) \geq 1 \tag{2.5}$$

2 クラス分類問題の場合式(2.5)の等式を満たす点 x_i は境界から最も近い点のみとなり,このときのマージン最大化問題は,

$$maximize \frac{1}{||w||} \tag{2.6}$$

となる.
また,これまでの議論により式(2.6)は,以下の制約付き最小化問題に置き換えることができる.

$$minimize \frac{1}{2}||w||^2 \qquad (2.7)$$
$$s.t.\, y_i(w^t x_i + w_0) \geq 1 (i = 1, \ldots, l)$$

この問題を解くため,Lagrange 未定乗数ベクトル a≧0 を導入し,w,w_0 については最小化,a_i については最大化する最適化問題に双対化することにより,以下の式が導かれる.

$$L((w, w_0), a) = \frac{1}{2}||w||^2 + \sum_{i=1}^{l} a_i\{1 - y_i(wx_i + w_0)\} \qquad (2.8)$$

式(2.7)を w,w_0 について微分することにより,最適化における条件として以下の式が導かれる.

$$w = \sum_{i=1}^{l} a_i y_i x_i \qquad (2.9)$$

$$\sum_{i=1}^{l} a_i y_i = 0 \qquad (2.10)$$

式(2.8)のwは,すなわち学習データの展開式となり,式(2.7)に式(2.8),式(2.9)を代入することにより,以下の凸最適化問題を得ることができる.

$$maximize \sum_{i=1}^{l} a_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} a_i a_j y_i y_j x_i x_j \qquad (2.11)$$
$$s.t. \sum_{i=1}^{l} a_i y_i = 0$$
$$a_i \geq 0 (i = 1, \ldots, l)$$

2.2.3 ロジスティック回帰について

ロジスティクス回帰モデルは[10],値域が[0,1]で総和が 1 になるような x の線形関数を用いてK 個のクラス事後確率をモデル化したいという欲求から生まれたモデルである.このロジスティック回帰モデルは,

$$log\frac{Pr(G=1|X=x)}{Pr(G=K|X=x)}=\beta_{10}+\beta_1^T x$$
$$log\frac{Pr(G=2|X=x)}{Pr(G=K|X=x)}=\beta_{20}+\beta_2^T x$$
$$\vdots$$
$$log\frac{Pr(G=K-1|X=x)}{Pr(G=K|X=x)}=\beta_{(K-1)0}+\beta_{K-1}^T x \quad (2.12)$$

の形で表せる.これは,K−1 個の対数オッズ(log-odds)もしくは(確率の総和が 1 となる拘束を付けた)ロジット(logit)変換として与えられるモデルである.ここではオッズ比の分母にクラス K の事後確率 Pr(G=K|X=x)を用いているが,分母にどのクラスを選んでも,得られる推定値は変わらない.そのため,任意のクラスを分母に選ぶことが可能である.少し式変形を行うと,

$$Pr(G=k|X=x)=\frac{exp(\beta_{k0}+\beta_k^T x)}{1+\sum_{l=1}^{K-1} exp(\beta_{l0}+\beta_l^T x)}, k=1,\ldots,K-1$$
$$Pr(G=K|X=x)=\frac{1}{1+\sum_{l=1}^{K-1} exp(\beta_{l0}+\beta_l^T x)} \quad (2.13)$$

が得られ,総和が 1 になることがわかる.ここで,それぞれの計算にパラメータ集合 $\theta=\{\beta_{10}, \beta_1^T,\ldots,\beta_{(K-1)0}, \beta_{K-1}^T\}$の全要素が必要である.このことをはっきりと示すために,$Pr(G=k|X=x)=p_k(x;\theta)$と表記する.

クラス数が K=2 の場合,ロジスティック回帰モデルは一つの線形関数のみで表すことができる.生物統計学では,患者の生死,心臓病の有無,条件の有無といった二値の応答(2 クラス)に対してロジスティック回帰モデルが広く利用されている.

2.2.4 ロジスティック回帰のアルゴリズム

ロジスティック回帰は分類モデルであり[9],非常に実装しやすいものの,高い性能が発揮されるのは線形分類可能なクラスに対してのみである.ロジスティック回帰は産業界において最も広く使用されている分類のアルゴリズムの 1 つである.パーセプトロンと同様に,本章で取り上げるロジスティック回帰モデルは二値分類のための線形モデルでもあり,たとえば一対他(OvR)手法に基づいて多クラス分類モデルとして拡張できる.

ロジスティック回帰の概念を理解するために,まずオッズ比(odds ratio)から見ていこう.オッズ比は事情の起こりやすさを表すもので,$\frac{p}{(1-p)}$と書くことができる.この場合,p は正事象の確率を表す.正事象(positive event)は必ずしも「良い」ことを意味するわけではなく,患者に疾患がある確率など,予測したい事象を表す.正事象については,クラスラベル y=1 として考えることができる.その場合は,ロジット(logit)関数を定義できる.この関数は単にオッズ比の対数(対数オッズ)となる.

$$logit(p) = log\frac{p}{(1-p)} \tag{2.14}$$

コンピュータサイエンスの慣例に従い,この場合の「対数」自然対数を表すことに注意しよう.ロジット関数は,0 よりも大きく 1 よりも小さい範囲の入力値を受け取り,それらを実数の全範囲の値に変換する.この関数を使って,特徴量の値と対数オッズとの間の線形関係を表すことができる.

$$logit(p(y = 1|x)) = w_0x_0 + w_1x_1 + \ldots + w_mx_m = \sum_{i=0}^{m} w_ix_i = \boldsymbol{w^Tx} \tag{2.15}$$

ここで p(y=1|x)は,特徴量 x が与えられる場合にサンプルがクラス 1 に属するという条件付き確率である.

ここで実際に関心があるのは,サンプルが特定のクラスに属している確率を予測することである.これはロジット関数の逆関数であり,ロジスティックシグモイド(logistic sigmoid)関数とも呼ばれる.その特徴的な S 字形により,単にシグモイド(sigmoid)関数と呼ばれることもある.

$$\phi(z) = \frac{1}{1+e^{-z}} \tag{2.16}$$

この場合の z は総入力である.つまり,重みとサンプルの特徴量との線形結合であり,次のように計算できる.

$$z = w^Tx = w_0x_0 + w_1x_1 + \cdots + w_mx_m \tag{2.17}$$

e^{-z} は z の値が大きい場合は非常に小さいため,z が無限大に向かう場合(z→∞)は Φ(z)が 1 に近づくことがわかる.同様に,z→−∞では分母が徐々に大きくなるため,Φ(z)は 0 に向かう.よって,このシグモイド関数は,入力として実数値を受け取り,Φ(z)=0.5 を切片として,それらの入力値を[0,1]の範囲の値に変換する.

ロジスティック回帰モデルに対する直観を養うために,以下に図解を示す.

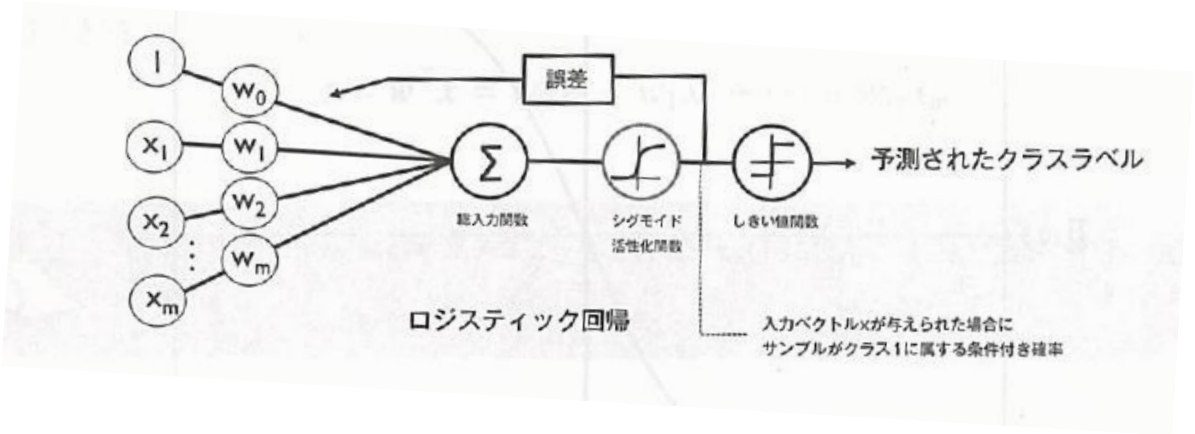


図 2.2　ロジスティック回帰の図解

特徴量 x が重み w でパラメータ化されるとすれば,このシグモイド関数の出力は,サンプルがクラス 1 に属している確率 $\Phi(z)=P(y=1|x; w)$であると解釈される.たとえば,あるサンプルに対して $\Phi(z)=0.8$ が算出される場合は,出力値がクラス 1 である確率が 80%であることを意味する.したがって,このサンプルがクラス 0 である確率は $P(y=0|x; w)=1-P(y=1|x; w)=0.2$,つまり 20%として計算される.あとは,閾値関数を使用することで,予測された確率を二値の成果指標に変換すればよい.

$$\hat{y} = \begin{cases} 1 & \phi(z) \geqq 0.5 \\ 0 & \phi(z) < 0.5 \end{cases} \tag{2.18}$$

先のシグモイド関数のグラフと照合すると,これは以下と等価であることがわかる.

$$\hat{y} = \begin{cases} 1 z \geqq 0.0 \\ 0 z < 0.0 \end{cases} \tag{2.19}$$

実際のところ,多くのアプリケーションでは,予測されるクラスラベルに関心があるだけでなく,クラスの所属関係の確率を見積もることが特に有益となる.たとえば気象予報では,雨が降るかどうかだけでなく,降水確率も発表するためにロジスティック回帰が使用される.同様に,特定の症状に基づいて患者が疾患にかかっている確率を予測する目的でも使用できるため,ロジスティック回帰は医療分野でも広く利用されている.

2.2.5 ランダムフォレストについて

ランダムフォレストは(Random Forest)は,分類性能が高く,スケーラビリティに優れ,使いやすいことから,機械学習の応用において非常に支持されている.直感的には,決定木の「アンサンブル」とみなすことができる.ランダムフォレストの背後にある考え方は,バリアンスが高い複数の決定木を平均化することで,より汎化性能が高い頑健なモデルを構築することである.

2.2.6 決定木について

決定木(Decision Tree)分類器は[9],意味解釈可能性に配慮する場合に用いられる.このモデルは,一連の質問に基づいて決断を下すという方法により,データを分類するモデルである.

2.2.7 ランダムフォレストのアルゴリズム

ランダムフォレストアルゴリズムは次の 4 つの単純な手順にまとめることができる.

1. サイズ n のランダムな「ブートストラップ」標本を復元抽出する(トレーニングデータセットから n 個のサンプルをランダムに選択する).
2. ブートストラップ標本から決定木を成長させる.各ノードで以下の作業を行う.
    a. d 個の特徴量をランダムに非復元抽出する.
    b. たとえば情報利得を最大化することにより,目的関数に従って最適な分割となる特徴量を使ってノードを分割する.
3. 手順 1〜2 を k 回繰り返す.
4. 決定木ごとの予測をまとめて,「多数決」に基づいてクラスラベルを割りあてる.

個々の決定木をトレーニングするときと比べて,手順 2 に若干の変更がる.ノードごとに最適な分割を判断するにあたってすべての特徴量を評価するのではなく,その一部をランダムに検討するだけとなる.

2.2.8 メル周波数ケプストラム係数

メル周波数ケプストラム係数(MFCC)とは[7],音声認識の分野で最も広く利用されている音響特徴量である.

プリエンファシス処理は,音声信号における周波数の偏りを修正するために高周波成分を強調させる目的で行われる.高周波成分が強調された信号 y(t)は次式によって表される.

$$y(t) = s(t) - p \cdot s(t - 1) \tag{2.20}$$

ここで s(t)は時刻 t における音声波形データ,p はプリエンファシス係数で,0.97 を使うことが多い.プリエンファシス処理をしたのり,音声波形に FFT(高速フーリエ変換)を行う.その後周波数軸上に,L 個の三角窓を配置し,窓幅に対応する周波数帯域の信号のパワーを m(l)とする.ここで窓幅は人間の聴覚特性にあわせて低周波ほど間隔が狭く,高周波ほど間隔を広くとる.このメタフィルタバンクによって得られたＬ個の帯域におけるパワーを次式によって DCT(離散コサイン変換)する.

$$c_k = \sqrt{\frac{2}{L}\sum_{l=1}^{L} logm(l)cos\{(l-\frac{1}{2})\frac{k\pi}{L}\}} \tag{2.21}$$

ここで k はケプストラム係数の次式を表しており,低次の成分を取り出したものが MFCC 特徴量である.本研究では 13 次元までを抽出している.

2.2.9 音声認識システム

音声認識をパターン認識によって行うシステムは,一般に図 2.3 に示すようなモジュール構成で実現される.[8]

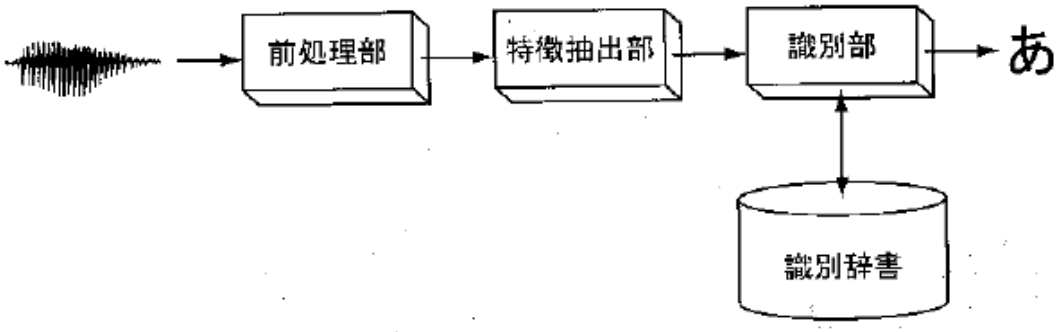


図 2.3 音声認識システムの構成

マイクなどの入力装置から入力されたアナログ信号を前処理によってコンピュータ内部で処理可能なディジタル信号に変換する.このディジタル化したデータを入力データとし,特徴量出を行う.特徴抽出は一般にベクトルの形式で抽出される.この特徴ベクトルを識別辞書中に存在する各クラスの訓練データと比較し識別結果が決定される.

本研究では少数のラベルありデータと類似したデータを訓練データとして用いてパラメータを決定した SVM でテストデータに対する識別を行う.入力データを 3 種類のニュージーランドの野鳥の鳴き声の音声データであり,前項の MFCC によって特徴抽出を行う.

3 提案手法

藤岡らの手法は,少数ラベル付きデータの平均ベクトルを中心点として一定距離以内のラベルなしデータを訓練事例集合として取得している.これは,特徴空間内でラベル付きデータと同種の鳴き声データがラベルなしデータの中に等方向的に分布していることを暗に仮定している.筆者らは,鳴き声の種類が少ない場合にはこの仮定は妥当だが, 鳴き声が多くなると鳴き声データは特徴空間内に鳴き声ごとに偏在し,等方性が減少して中心から等距離では不適切なデータが含まれやすいのではないかと考えた.

そこで,これに対応する手法として,少数のラベル付きデータのそれぞれを中心点として中心点の個数回ラベルなしデータの取得を行う方法を提案する.これにより,ランダムに選択されたラベル付きデータの分布を利用してラベルなしデータの偏在にある程度対応できるのではないかと考えた.

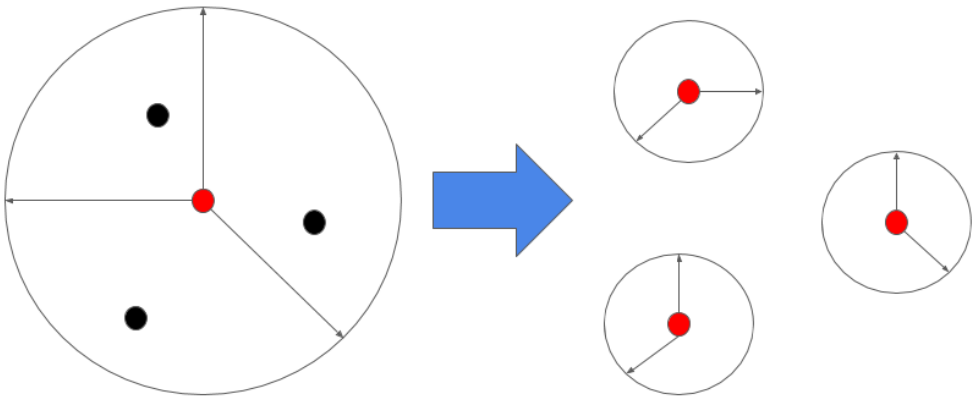


図 3.1 提案手法の略図

4 実験と考察

4.1 実験概要

提案手法によって決定した訓練データを用いて検証実験を行った.テストデータは訓練データとの重複を避けるためあらかじめデータセットからランダムで 20 個抜き出して用いた.中心点には 1 から 10,距離には 5 段階の基準を設けた.具体的には 1000,1200,1400,1600,1800,2000 であり,基準値以下であれば正例,上回れば負例のラベルをつけて訓練データとして用いる.識別率の測定は各距離ごと,各鳥ごと,中心データの数ごと行い, 30 回繰り返した場合の識別率の平均値を測定値とする.使用するデータセットは野鳥の鳴き声データで,ミヤマオウム,キジカッコウ,ニュージーランドアオバズクの 3 種類それぞれ 60 セグメントのデータセットにした.なお,ミヤマオウムについては 3 種類以上の鳴き声,キジカッコウについては 3 種類程度の鳴き声,ニュージーランドアオバズクについては 1 種類の鳴き声データで構成されている.これらの鳴き声データを MFCC による特徴抽出を行い,910 次元ベクトルを作成し,特徴ベクトルとした.また,機械学習と認識精度の測定にはサポートベクターマシン(SVM)を用い,手法比較としてロジスティック回帰,ランダムフォレスト(RF)を用いる.各々の手法にはグリッドサーチをかけ,チューニングを行った.本実験にはそのパラメータを入力した手法を用いた.

4.2 実験環境

本研究の実験では,多くのデータサイエンスアプリケーションの共通語となっている Python(ver3.7)を使用した[11].使用音源は Long-tailed cuckoo・Kea・Morepork の鳴き声 1 回分(2 秒から 5 秒)が収録された.WAVE 形式のファイルである[12].それらに対して,MFCC で算出し,ロジスティック回帰,SVC 及びランダムフォレストで 3 クラスに分類した.その他実験環境を表 1 に示す.

表 1.計算環境

CPU	Intel(R) Core™ i7-8750 2.20GHz
OS	Ubuntu 18.04
メモリ	8 GB
numpy	1.15.4
pandas	0.25.3
scipy	1.4.1
matplotlib	3.1.1
scikit-learn	0.22.1

4.3 実験方法

基準値を設けたラベリング処理のフローを図 4.1 に示す.機械学習手法を用いた識別率の測定は各距離ごとに行い,30 回繰り返した場合の識別率の平均値を測定値とする.この全体の処理を図 4.2 に示し,実験の詳細を以下に示す.

使用する生データを用意する.この際データ形式は wave 形式(拡張子.wav)とする.本実験では,野鳥の鳴き声を MFCC を用いて変換し,識別器に入力しやすい csv 形式データにする.そのため MFCC を Python を用いて wave 形式データを csv 形式データに変換する.その際,1 つのデータファイルにつきサンプル数 20,特徴量 910 次元となる.csv 形式にしたデータファイルを展開すると縦に 20 行,横に 910 列になる.

変換後の csv ファイルを中心点や距離ごとに振り分けるために訓練・テスト用データセット作成プログラム[Make_Dataset_(鳥名).py]に入力する.このプログラムによって中心点・距離に応じてデータが振り分けられる.中心点が 1 から 10,距離が 1000,1200,1400,1600,1800,2000 あるため,(中心点の数の種類×距離の種類)の数だけ振り分けのパターンが存在する.

次に,データセット作成プログラムによって振り分けられたデータセットを入力データとして識別プログラム(main_classifier.py)に入力する.識別プログラム内では選択した学習手法によって学習を行う.識別した結果の評価方法は正解率を用いている.正解率の出力方法は今回はライブラリを用いているが,自作でも可能である.その後,識別した結果である正解率を csv 形式データとして出力し保存する.なお,ここまでの過程を中心点 1 から 10 と距離 1000,1200,1400,1600,1800,2000 ごとにデータセット作成と識別器による学習を 30 回行う.

30 回学習した結果である正解率をグラフ可視化プログラム(graph.py)を用いて可視化する.この際,横軸は訓練数,縦軸は Accuracy(正解率)となる.訓練数がデータセット作成プログラムによって多少増減しているため,横軸の最大値はばらつきがある.出力されたグラフを png 形式で野鳥ごとのディレクトリに保存する.

上記過程を野鳥ごとに行う.これらの過程を効率的に行うためにシェルスクリプト[2class_(鳥名).sh]を組んだ.シェルスクリプトに渡す引数を変化させれば,プログラムの実行方法を変化させることができる.引数に 1 を渡せば全パターン(中心手の数の種類×距離の種類)実行し,2 を渡せば[中心点(1,3,10)×距離(1000,1600,2000)]のパターンのみ実行できる.

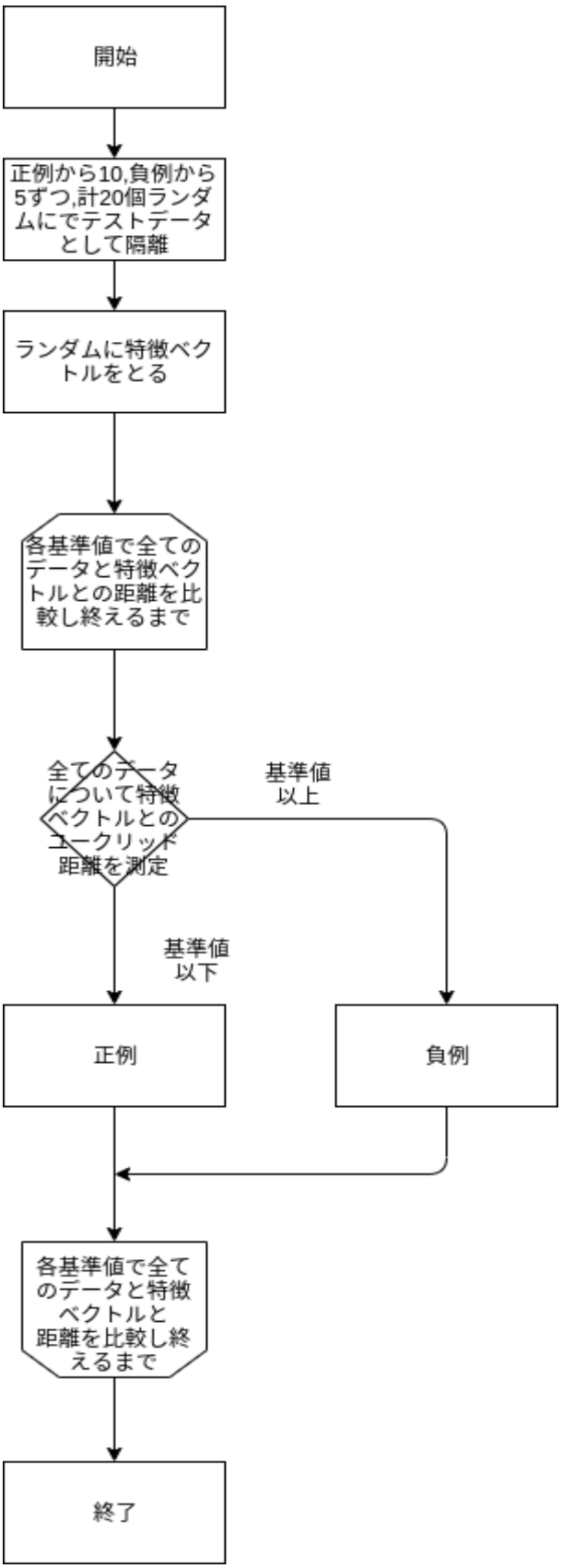

開始
正例から10,負例から5ずつ,計20個ランダムにでテストデータとして隔離
ランダムに特徴ベクトルをとる
各基準値で全てのデータと特徴ベクトルとの距離を比較し終えるまで
全てのデータについて特徴ベクトルとのユークリッド距離を測定
基準値以上
基準値以下
正例
負例
各基準値で全てのデータと特徴ベクトルと距離を比較し終えるまで
終了

図 4.1　実験のフロー

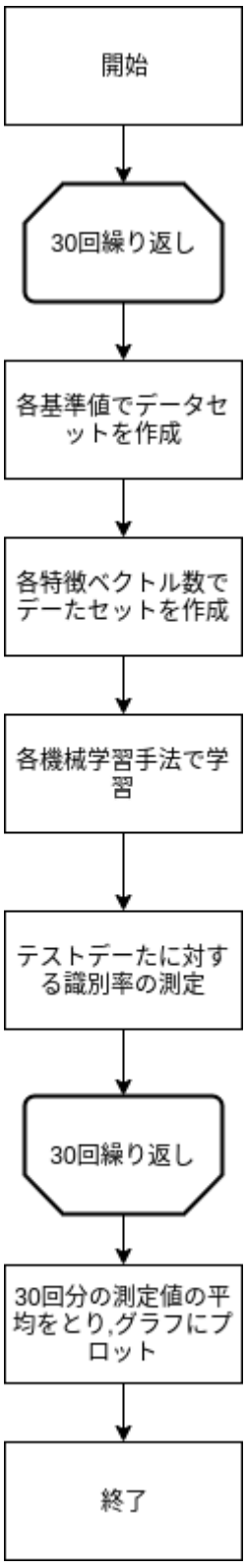

開始
30回繰り返し
各基準値でデータセットを作成
各特徴ベクトル数でデーたセットを作成
各機械学習手法で学習
テストデーたに対する識別率の測定
30回繰り返し
30回分の測定値の平均をとり,グラフにプロット
終了

図 4.2　全体のフロー

4.4 実験結果

実験の結果の一部を図 4.3 から図 4.12 に示す.横軸は訓練事例数,縦軸は認識精度を示している.ミヤマオウム,キジカッコウ,ニュージーランドアオバズク(以下 NZ アオバズク)の順に 3×3 形式に図を示す.中心点の数が 1 のグラフは従来手法を用いている.

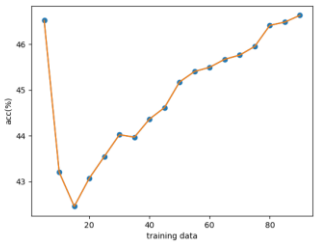


図 4.3
正例ミヤマオウム
距離 2000
中心点 1
ロジスティック回帰

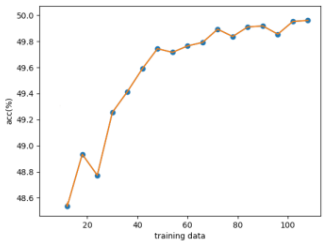


図 4.4
正例ミヤマオウム
距離 2000
中心点 3
ロジスティック回帰

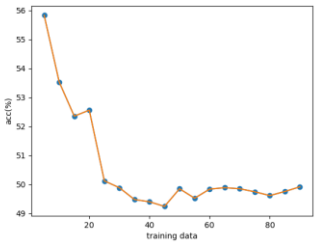


図 4.5
正例ミヤマオウム
距離 2000
中心点 10
ロジスティック回帰

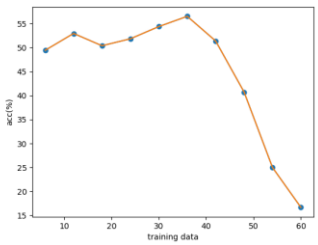


図 4.6
正例ミヤマオウム
距離 2000
中心点 1
SVM

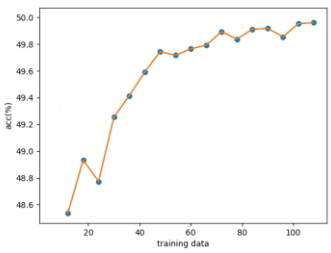


図 4.7
正例ミヤマオウム
距離 1600
中心点 3
SVM

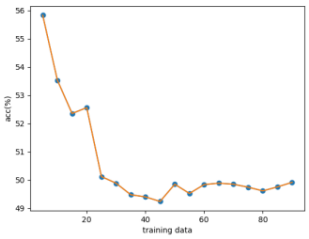


図 4.8
正例ミヤマオウム
距離 2000
中心点 10
SVM

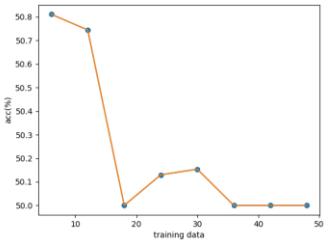


図 4.9
正例ミヤマオウム
距離 1000
中心点 1
ランダムフォレスト

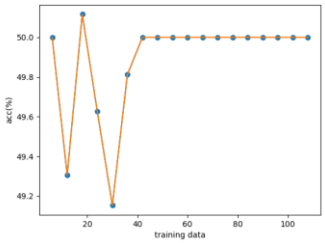


図 4.10
正例ミヤマオウム
距離 1800
中心点 3
ランダムフォレスト

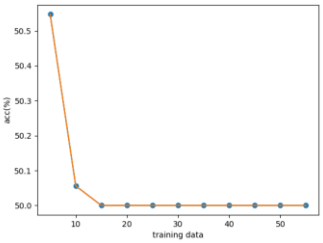


図 4.11
正例ミヤマオウム
距離 1400
中心点 10
ランダムフォレスト

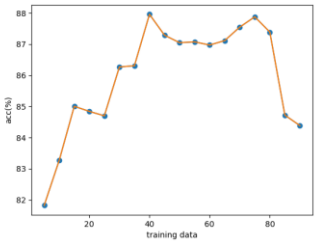


図 4.12
正例キジカッコウ
距離 1500
中心点 1
ロジスティック回帰

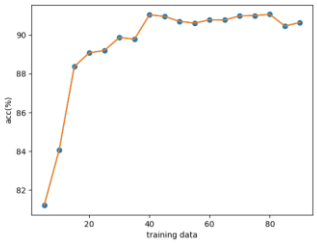


図 4.13
正例キジカッコウ
距離 1500
中心点 3
ロジスティック回帰

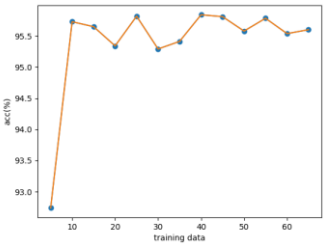


図 4.14
正例キジカッコウ
距離 1800
中心点 10
ロジスティック回帰

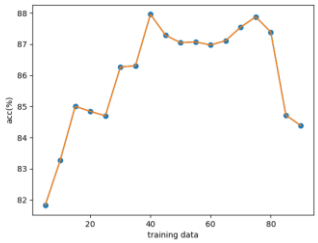


図 4.15
正例キジカッコウ
距離 1800
中心点 1
SVM

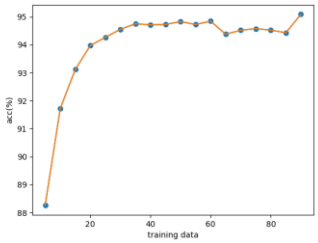


図 4.16
正例キジカッコウ
距離 1800
中心点 3
SVM

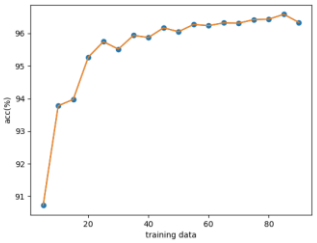


図 4.17
正例キジカッコウ
距離 1800
中心点 10
SVM

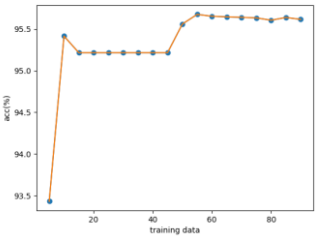


図 4.18
正例キジカッコウ
距離 1500
中心点 3
ランダムフォレスト

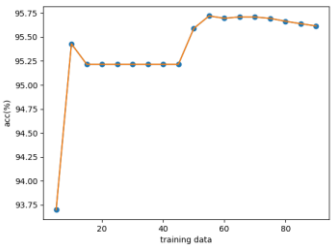


図 4.19
正例キジカッコウ
距離 1500
中心点 3
ランダムフォレスト

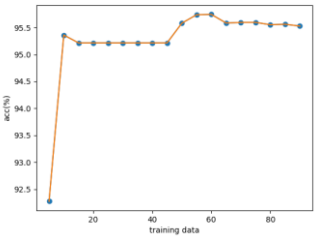


図 4.20
正例キジカッコウ
距離 1500
中心点 10
ランダムフォレスト

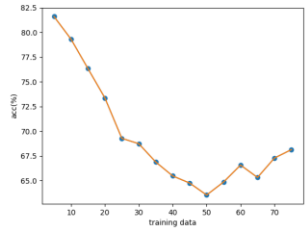


図 4.21
正例 NZ アオバズク
距離 1500
中心点 1
ロジスティック回帰

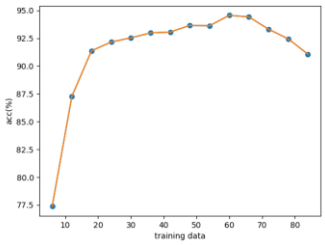


図 4.22
正例 NZ アオバズク
距離 2000
中心点 3
ロジスティック回帰

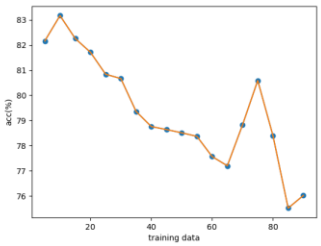


図 4.23
正例 NZ アオバズク
距離 1500
中心点 10
ロジスティック回帰

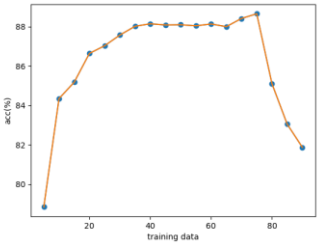


図 4.24
正例 NZ アオバズク
距離 1200
中心点 1
SVM

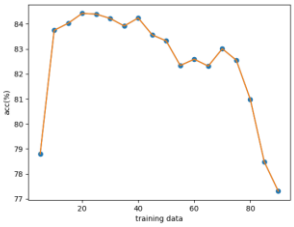


図 4.25
正例 NZ
アオバズク
距離 1500
中心点 3
SVM

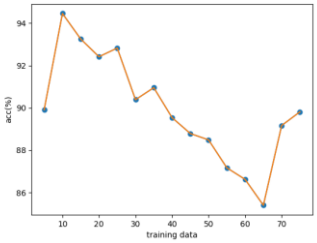


図 4.26
正例 NZ アオバズク
距離 1800
中心点 10
SVM

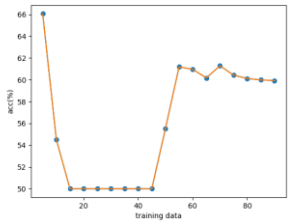


図 4.27
正例 NZ アオバズク
距離 1000
中心点 1
ランダムフォレスト

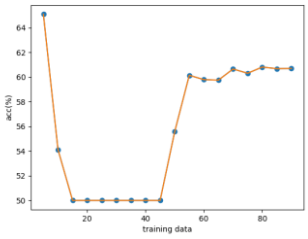


図 4.28
正例 NZ アオバズク
距離 1800
中心点 3
ランダムフォレスト

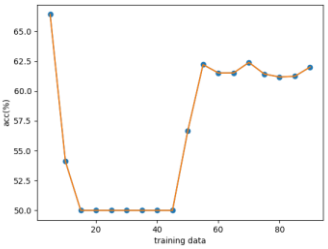


図 4.29
正例 NZ アオバズク
距離 1800
中心点 10
ランダムフォレスト

4.5 考察

3 つの機械学習手法を用いて識別率を測定した.ミヤマオウム,キジカッコウ,NZ アオバズクと順に見ていくと,手法による違いが大きいことがわかる.

ミヤマオウムに関してはデータの複雑性が 3 羽の中で最も高いこともあり,全体的に芳しい結果は得られなかった.図 4.8 からは中心点を多く配置するほど良好な結果が得られるという予測に反して,正解率の停滞が確認できる.ミヤマオウムの場合,データの偏在性が高いこともあり負例のデータも中心点からの距離内に含まれてしまったと考えれる.そのため,訓練数を増やしていくに従い正解率が停滞していったと考えれる.しかし,SVM かつ中心て 3 個で行った図 4.7 に関しては,正解率の上昇傾向が高いことから訓練数を増やしていくとさらに正解率が伸びていくと考えれる.

キジカッコウ場合,他の 3 羽のデータと比較してデータの偏在性は 3 羽の中で平均的である.SVM を実行した行である図 4.15,16,17 を見ると,高い正解率かつ正解率の上昇傾向が確認できる.中心手の数を増やしていくほどに正解率が上昇しているため,キジカッコウにおいては提案手法が適切であったと考えられる.ロジスティック回帰で実行した際にも上記と同じ傾向が確認できる.対して,ランダムフォレストの場合は他の手法と同じような毛校は確認できたない.ランダムフォレストは複雑性の高い手法であり,複雑なデータに適切なパラメータを指定した場合に高い識別率を獲得できると考えれる.データの複雑性という観点においてキジカッコウは低い複雑性であったため,あまり変化を確認できない正解率となったと考えられる.

NZ アオバズクにおいては,データの種類が単一であるということから,単純な手法での正解率の上昇傾向が高いことが確認できる.ロジスティック回帰の場合,他 2 つ手法と比較しても複雑性の低い手法であるため,NZ アオバズクのデータに適切であったと考えれる.距離が小さい場合は正解率は下降傾向にあるが,中心点を多くすれば高い正解率であることが確認できる(図 4.22,23).SVM を用いて実行した場合,エポック数が少ないときには高い正解率であったが,数が多くなるに従い下降傾向になる.しかし,中心点の数が多いときに学習回数を増やすと正解率が上昇している傾向が確認できる.このことから学習数と中心点の数を増やしていくほど正解率が高くなることが確認できる.ランダムフォレストに関しても,多少の変化はあっても,一定の学習回数を過ぎるとあまり変化は確認できない.50%以上の正解率を保ってはいるが,これも適切なパラメータの設定が必要であると考える.

従来手法の場合は,全体的に下降傾向にあることが確認できる.特に SVM においては,どの鳥と比較してもそれは顕著である.しかし,ランダムフォレストを用いた場合とミヤマオウムにロジスティック回帰を用いた場合は,あまりその変化がみられない.これらに関しては,中心点の数だけでなく,パラメータや前処理過程での処理を適切にする必要があると考えられる.

上記のことから,比較的単純な学習手法では正例データの偏在性を高めること,中心点の数を多くすることで正解率が上昇することが確認できた.しかし,ランダムフォレストのような複雑な学習手法では,従来手法や提案手法との大きな差は確認できなかった.これらのことから,比較的単純な学習手法では中心点の数を増やすことで従来手法より提案手法の方が正解率が安定していることがわかった.

5 おわりに

教師あり学習において,学習に多数の訓練データを用いるほど識別率が高くなることが知られていることから,少数のラベル付きデータからの距離が近いものをラベル付きデータと類似したデータとみなして訓練データとして用いる手法の改良手法を提案し,比較実験を行った.このとき手法はデータ偏在性に対応した最適な手法があることが確認できた.訓練データや中心点数を変化させて行った実験から,ミヤマオウムなどの複数の種類の鳴き声が混ざっている鳴き声をする鳥に関しては,有用な結果を得ることができなかったが,単一な鳴き声をするニュージーランドアオバズクの場合は他の結果と比較して有用な結果を得た.また,各々のデータの偏在性に適した手法を用いることで識別率が向上した.実験結果より,従来手法では識別率が不安定であった鳴き声の種類が多い鳥に対しても,提案手法では比較的安定した識別率が得られることがわかった.

今後の課題として,中心点の数と分散や標準偏差を加味した有効なデータが含まれる距離との相関関係及び他の野鳥との比較した結果を実験により確認する.

謝辞

本論文を作成するにあたり，多くのご指摘，ご助言を頂きました三好　力　教授に厚く御礼申し上げます．また，議論に協力してくださった三好研究室の皆様や学友の皆様に心から感謝致します．

参考文献

[1] 藤岡 優也, 三好 力 :“機械学習における特徴量類似性と認識精度に関する検討,” 第 17 回情報科学技術フォーラム論文集,F-038, 福岡県,福岡工業大学(2018.09).
[2]“ サポートベクターマシン (SVM)”, http://www.sist.ac.jp/~kanakubo/research/neuro/supportvectormachine.html, 2019/10/30
[3]“ サポートベクターマシンを手計算で理解する ”, http://s0sem0y.hatenablog.com/entry/2017/01/24/201702, 2019/10/30
[4] 中川 祐志 (2018),“ サポートベクターマシン ”, http://www.r.dl.itc.u-tokyo.ac.jp/~nakagawa/SML1/kernel1.pdf ,2019/10/30
[5] 阿部重夫, パターン認識のためのサポートベクトルマシン入門, 森北出版株式会社, 2011
[6] 高村大也, “言語処理のための機械学習入門”, コロナ社, 2010
[7]“ メル周波数ケプストラム係数 (MFCC)”, http://aidiary.hatenablog.com/entry/20120225/1330179868 ,2019/10/30
[8] 荒木 雅弘, “フリーソフトでつくる音声認識システム”, 森北出版株式会社, 2007
[9] Sebastian Raschka, Vahid Mirjalili, “Python 機械学習プログラミング 達人データサイエンティストによる理論と実装(第 2 版)”, 株式会社インプレス, 2019
[10] Trevor Hastie, Robert Tibshirani, Jerome Friedman, “統計的学習の基礎 データマイニング・推論・予測”, 共立出版株式会社, 2014
[11] Andreas C. Muller, “Python ではじめる機械学習 scikit-learn で学ぶ特徴量エンジニアリングと機械学習の基礎”, 株式会社オライリージャパン, 2017
[12] 前川 志帆, 三好 力 :”鳴き声を用いた鳥の自動分類における最適パラメータの検討,” 龍谷大学 理工学部 情報メディア学科 平成 29 年度特別研究報告書, 2017

付録 A

識別プログラム

```
#!/usr/bin/env python
# coding: utf-8

# 必須モジュール
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
import matplotlib.pyplot as plt
import os
import re
import math
import random
import sys
# データセット
from sklearn.datasets import make_classification
# 前処理モジュール
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.preprocessing import PolynomialFeatures
# モデルの性能評価
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold, cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import learning_curve
# チューニング
from sklearn.pipeline import make_pipeline
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
# 学習モデル
from sklearn.dummy import DummyClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.ensemble import AdaBoostClassifier
# 評価モジュール
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.metrics import roc_curve, auc, roc_auc_score

# segments(renamed)
# Kea 60
# LTCuckoo 60
# Moreporrk 60
# total 180 segments
#
# 4 クラス分類ラベル
# label_ = "Kea"
# label_2 = "LTCuckoo"
# label_3 = "Morepork"
# label_4 = "Noise"

num = range(0, 200)

# 2 クラス分類のラベル
label_pos = 0.0 # 正例
label_neg = 1.0 # 負例
# 実験番号
# exp_num = 1800
exp_num = int(sys.argv[1] + "00")

# train は学習データ, val は検証データ, test はテストデータ
# X は入力特徴量, y は出力ベクトル
X_train = [] # 訓練入力データ
y_train = [] # 訓練出力ラベル
X_test = [] # テスト入力データ
y_test = [] # テスト出力ラベル

def create_classifier():
    """学習モデル構築する関数
        返り値:
            classifier:
                使用する手法
                パラメータは事前にグリッドサーチをして出力されたものを使用する
    """
    classifier = LogisticRegression(penalty='l2', random_state=1)
    # classifier = SVC(random_state=1, C=1.0, kernel='linear')
    # classifier = LinearSVC(random_state=1, C=1.0)

    return classifier

def create_data():
    """訓練データを DataFrame に加工して返す関数

        返り値:
            X_train_val_df:
                入力訓練事例集合
            y_train_val_df:
                出力訓練事例集合
            X_train_df:
                入力訓練事例集合(検証用抜き)
            X_val_df:
                入力検証事例集合
            y_train_df:
                出力訓練事例集合(検証用抜き)
            y_val_df:
                出力検証事例集合

    """
    # 訓練データのデータフレーム化
    X_train_val_df = pd.DataFrame(X_train)
    y_train_val_df = pd.DataFrame(y_train)
    X_train_df, X_val_df, y_train_df, y_val_df = train_test_split(X_train_val_df,
y_train_val_df, random_state=1, train_size=0.9)
    X_train_df = pd.DataFrame(X_train_df)
    X_val_df = pd.DataFrame(X_val_df)
    y_train_df = pd.DataFrame(y_train_df)
    y_val_df = pd.DataFrame(y_val_df)
    print("X_train_val_df: {}".format(X_train_val_df.shape))
    print("X_train_df: {}".format(X_train_df.shape))
    print("X_val_df: {}".format(X_val_df.shape))
    print("X_test_df: {}".format(X_test_df.shape))
    print("y_train_val_df: {}".format(y_train_val_df.shape))
    print("y_train_df: {}".format(y_train_df.shape))
    print("y_val_df: {}".format(y_val_df.shape))
    print("y_test_df: {}".format(y_test_df.shape))

    return X_train_val_df, y_train_val_df, X_train_df, X_val_df, y_train_df, y_val_df

def create_pos_test():
    """正例テストデータ作成する関数
        返り値:
            X_test_df_pos:
                入力正例試験集合
            y_test_df_pos:
                出力正例試験集合
    """
    # 正例テスト
    dir = 'pos_test' # ディレクトリのパスを取得
    files = os.listdir(dir) # ファイルのリストを取得
    count = 0 # カウンタの初期化
    X_test_pos = []
    y_test_pos = []

    for file in files: # ファイルの数だけループ
        index = re.search('.csv', file) # 拡張子が csv のものを検出
        if index: # カウンタをカウントアップ
            count += 1

    for i in num[1: count+1]: # ファイルの数だけループ
        pos = np.loadtxt("pos_test/pos_test_%d.csv" % (i), delimiter=',')
        X_test_pos.append(pos)
        y_test_pos.append(label_pos) # label_pos に正例テストデータ格納

    # DataFrame 化
    X_test_df_pos = pd.DataFrame(X_test_pos)
    y_test_df_pos = pd.DataFrame(y_test_pos)
    print("X_test_df_pos: {}".format(X_test_df_pos.shape))
    print("y_test_df_pos: {}".format(y_test_df_pos.shape))

    return X_test_df_pos, y_test_df_pos

def create_neg_test():
    """負例テストデータ作成

    戻り値
    ----------
        X_test_df_neg:
            負例 X_test の DataFrame
        y_test_df_neg:
            正例 y_test の DataFrame
```

```
    負例テストデータ数:
        10
    ディレクトリ名:
        neg_test
    """
    # 負例テスト
    dir = 'neg_test' # ディレクトリのパスを取得
    files = os.listdir(dir) # ファイルのリストを取得
    count = 0 # カウンタの初期化
    X_test_neg = []
    y_test_neg = []

    for file in files: # ファイルの数だけループ
        index = re.search('.csv', file) # 拡張子が csv のものを検出
        if index: # カウンタのカウントアップ
            count += 1

    for i in num[1: count+1]:
        neg = np.loadtxt("neg_test/neg_test_%d.csv" % (i), delimiter=',')
        X_test_neg.append(neg)
        y_test_neg.append(label_neg) # label_neg に負例テストデータ格納

    # DataFrame 化
    X_test_df_neg = pd.DataFrame(X_test_neg)
    y_test_df_neg = pd.DataFrame(y_test_neg)
    print("X_test_df_neg: {}".format(X_test_df_neg.shape))
    print("y_test_df_neg: {}".format(y_test_df_neg.shape))

    return X_test_df_neg, y_test_df_neg

# テストデータセット受け取り
X_test_df_pos, y_test_df_pos = create_pos_test()
X_test_df_neg, y_test_df_neg = create_neg_test()

# テストデータセット連結
X_test_df = pd.concat([X_test_df_pos, X_test_df_neg], axis=0)
y_test_df = pd.concat([y_test_df_pos, y_test_df_neg], axis=0)
print("X_test_df: {}".format(X_test_df.shape))
print("y_test_df: {}".format(y_test_df.shape))


# In[72]:


def count_pos_train():
    """正例訓練事例集合のカウント数を返す関数

        返り値:
            pos_cnt:
                正例訓練事例集合の数

    """
    # ディレクトリパス取得
    dir_ = "pos_train" + str(exp_num)
    # ファイルのリスト取得
    files = os.listdir(dir_)
    # カウンタフォーマット
    pos_cnt = 0
    for file in files: # ファイルの数だけループ
        if re.search('.csv', file):
            pos_cnt += 1
    print("pos_cnt: {}".format(pos_cnt))

    return pos_cnt

def count_neg_train():
    """負例訓練事例集合のカウント数を返す関数

        返り値:
            neg_cnt:
                負例訓練事例集合の数

    """
    # ディレクトリパスの取得
    dir_ = "neg_train" + str(exp_num)
    # ファイルのリスト取得
    files = os.listdir(dir_)
    # カウンタフォーマット
    neg_cnt = 0
    for file in files: # ファイルの数だけループ
        if re.search('.csv', file):
            neg_cnt += 1
    print("neg_cnt: {}".format(neg_cnt))

    return neg_cnt

pos_cnt = count_pos_train()
neg_cnt = count_neg_train()

def nazo():
    """ループ回数の最大値を決める関数

        返り値:
            rnum:
                ループ回数の最大値
    """
    rnum = 0
    if pos_cnt < neg_cnt:
        rnum = math.floor(pos_cnt // 3)
    else:
        rnum = math.floor(neg_cnt // 3)
    rnum = int(rnum)
    pos_num = pos_cnt
    neg_num = neg_cnt

#     print("rnum: {}".format(rnum))
    return rnum

rnum = nazo()
list_ = []

classifier = create_classifier()

for i in num[1: rnum + 1]:
    # 正例の数だけ配列に代入
    get_sample = i * 3
    list_ = []
    list_ = random.sample(range(1, pos_cnt+1), get_sample)
    # pos_cnt だけループ
    for j in num[1: pos_cnt + 1]:
        # 選ばれたものを抽出
        if j in list_:
            pos_ = np.loadtxt("pos_train{}/pos_train_{}.csv".format(exp_num, j),
                              delimiter=',')
            X_train.append(pos_)
            y_train.append(label_pos)
            list_ = []
            list_ = random.sample(range(1, neg_cnt+1), get_sample)
            # neg_cnt だけループ
            for k in num[1: neg_cnt + 1]:
                # 選ばれたものを抽出
                if k in list_:
                    neg_ = np.loadtxt("neg_train{}/neg_train_{}.csv"
                                      .format(exp_num, k),
                                      delimiter=',')
                    X_train.append(neg_)
                    y_train.append(label_neg)

            # データ受け取り
            X_train_val_df, y_train_val_df, X_train_df, X_val_df, y_train_df, y_val_df =
create_data()
            # ターゲットを一次元化
            y_train_val_df = y_train_val_df[:].values
            # 訓練
            classifier.fit(X_train_val_df, y_train_val_df)
            # 評価
            prediction_ = classifier.predict(X_test_df)
            cm = confusion_matrix(y_test_df, prediction_)
            # 正解率
            score_ = classifier.score(X_test_df, y_test_df)
            # 適合率
            precision_ = precision_score(y_test_df, prediction_)
            txt_ = np.loadtxt("Score_precision_/precision_{}.csv".format(i), delimiter=',')
            txt_ = np.append(txt_, precision_)
            np.savetxt("Score_precision_/precision_{}.csv".format(i), txt_, delimiter=',')
            # 再現率
            recall_ = recall_score(y_test_df, prediction_)
            txt_ = np.loadtxt("Score_recall_/recall_{}.csv".format(i), delimiter=',')
            txt_ = np.append(txt_, recall_)
            np.savetxt("Score_recall_/recall_{}.csv".format(i), txt_, delimiter=',')
            # F-1 値
            f1_ = f1_score(y_test_df, prediction_)
            txt_ = np.loadtxt("Score_f1_/f1_{}.csv".format(i), delimiter=',')
            txt_ = np.append(txt_, f1_)
            np.savetxt("Score_f1_/f1_{}.csv".format(i), txt_, delimiter=',')
            # 評価指標をまとめて算出
            print(classification_report(y_test_df, prediction_))
            # 前スコアを取り出し,当スコアを格納
            txt_ = np.loadtxt("score %d.csv" % (i), delimiter=',')
            txt_ = np.append(txt_, score_)
            np.savetxt("score %d.csv" % (i), txt_, delimiter=',')

            #フォーマット化
```

```
        X_train = []
        y_train = []

        print(cm)
```

データセット作成プログラム

```
#!/usr/bin/env python
# coding: utf-8

# 必要モジュール
import random
import numpy as np
import shutil
import os
import sys

# コマンドライン第一引数から中心点数を受け取る
point_num = int(sys.argv[1])
# point_num = 5

# コマンドライン第二引数から距離を受け取る
distance = int(sys.argv[2])
# distance = 20

def mkdir_supervised(first_, last_):
    """正負分離ディレクトリ削除・生成"""

    if os.path.exists('pos_test'): # pos_test がある場合
        shutil.rmtree("pos_test")
    if os.path.exists('pos_train'):
        shutil.rmtree("pos_train")
    if os.path.exists('neg_test'):
        shutil.rmtree("neg_test")
    if os.path.exists('neg_train'):
        shutil.rmtree("neg_train")

    #尺度内教師ディレクトリ削除
    for p in range(first_, last_ + 1):
        if os.path.exists("pos_train" + str(p) + "00"):
            shutil.rmtree("pos_train" + str(p) + "00")

    #尺度外ディレクトリ削除
    for n in range(first_, last_ + 1):
        if os.path.exists("neg_train" + str(n) + "00"):
            shutil.rmtree("neg_train" + str(n) + "00")

    #正負分離ディレクトリ作成
    os.makedirs("pos_test")
    os.makedirs("pos_train")
    os.makedirs("neg_test")
    os.makedirs("neg_train")

    #尺度内教師ディレクトリ作成
    for p in range(first_, last_ + 1):
        os.makedirs("pos_train" + str(p) + "00")

    #尺度外教師ディレクトリ作成
    for n in range(first_, last_ + 1):
        os.makedirs("neg_train" + str(n) + "00")

def random_train_test_split(range_num, pos_csv_cnt=1, neg_csv_cnt=1):
    """正例と負例に分離"""

    # 1 から 60 までの数字をランダムで 10 個配列に入れる。
    pos_list = []
    # 10 個取り出す
    pos_list = random.sample(range(1,60), 10)# 抽出する添字を取得

    # テストデータと教師データの分離
    # ランダム配列にある要素の番号の鳥の名前はテストデータフォルダへ、その他は教師データへ

    # 正例
    for mn in range_num[1: 61]:
        if mn in pos_list:
            shutil.copyfile("Morepork/Morepork {}.csv".format(mn),
                            "pos_test/pos_test_{}.csv".format(pos_csv_cnt))
            pos_csv_cnt += 1
        else:
            shutil.copyfile("Morepork/Morepork {}.csv".format(mn),
                            "pos_train/pos_train_{}.csv".format(neg_csv_cnt))
            neg_csv_cnt += 1

    # 初期化
    pos_csv_cnt = 1
    neg_csv_cnt = 1

    # 5 つ取り出す
    neg_list = random.sample(range(1, 60), 5)

    # 負例
    for mn in range_num[1: 61]:
        if mn in neg_list:
            shutil.copyfile("Kea/Kea {}.csv".format(mn),
                            "neg_test/neg_test_{}.csv".format(pos_csv_cnt))
            pos_csv_cnt += 1
        else:
            shutil.copyfile("Kea/Kea {}.csv".format(mn),
                            "neg_train/neg_train_{}.csv".format(neg_csv_cnt))
            neg_csv_cnt += 1

    # 負例
    for mn in range_num[1: 61]:
        if mn in neg_list:
            shutil.copyfile("LTCuckoo/LTCuckoo {}.csv".format(mn),
                            "neg_test/neg_test_{}.csv".format(pos_csv_cnt))
            pos_csv_cnt += 1
        else:
            shutil.copyfile("LTCuckoo/LTCuckoo {}.csv".format(mn),
                            "neg_train/neg_train_{}.csv".format(neg_csv_cnt))
            neg_csv_cnt += 1

def euclidean_distance(range_num, x, distance=10, pos_csv_cnt=1, neg_csv_cnt=1):
    """正例フォルダ、負例ディレクトリそれぞれで 3 平均ベクトルとユークリッド距離を比較"""

    # 文字列用距離
    str_dist = str(distance)
    # 判別用距離
    discrimination = int(str_dist + "00")

    #正例教師から比較
    for mn in range_num[1:51]:
        y = np.loadtxt("pos_train/pos_train_{}.csv".format(mn), delimiter=',')
        D = np.linalg.norm(x-y)
        if D < discrimination:
            shutil.copyfile("pos_train/pos_train_{}.csv".format(mn),
                            "pos_train" + str_dist + "00/pos_train_{}.csv".format(pos_csv_cnt))
            pos_csv_cnt += 1
        else:
            shutil.copyfile("pos_train/pos_train_{}.csv".format(mn),
                            "neg_train" + str_dist + "00/neg_train_{}.csv".format(neg_csv_cnt))
            neg_csv_cnt += 1

    #負例教師から比較
    for mn in range_num[1:101]:
        y = np.loadtxt('neg_train/neg_train_{}.csv'.format(mn), delimiter=',')
        D = np.linalg.norm(x-y)
        if D < discrimination:
            shutil.copyfile("neg_train/neg_train_{}.csv".format(mn),
                            "pos_train" + str_dist + "00/pos_train_{}.csv".format(pos_csv_cnt))
            pos_csv_cnt += 1
        else:
            shutil.copyfile("neg_train/neg_train_{}.csv".format(mn),
                            "neg_train" + str_dist + "00/neg_train_{}.csv".format(neg_csv_cnt))
            neg_csv_cnt += 1


def calc_point(point_num=3):
    """正例教師フォルダから、3つベクトルを選び出し、平均をとり x に入れる。"""
    # 中心点を指定個数分取り出す
    point_list = random.sample(range(1,50), point_num)
    # numpy 配列宣言
    pos_sum = 0

    # 中心点個数分ループ
    for i in point_list:
        pos_sum += np.loadtxt("pos_train/pos_train_{}.csv".format(i), delimiter=',')
        print(pos_sum)
    print(pos_sum.size)
    # 平均値格納
    x = pos_sum/point_num

    return x

def multiple_points_distance(range_num, point_num, distance=10, func_dis=euclidean_distance):
    """複数の中心点を用いてデータセットを作成する窓口関数

        引数:
```

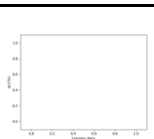
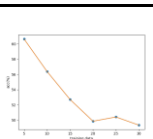
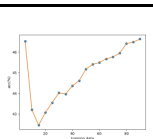
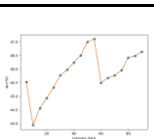
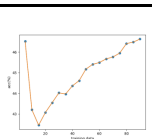
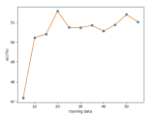
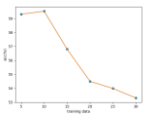
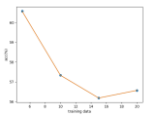
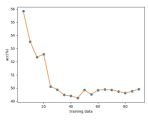
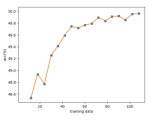
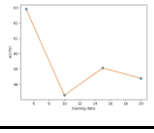
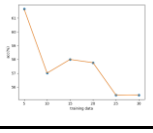
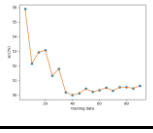
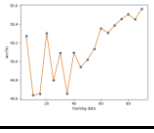
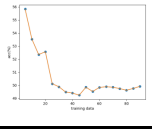
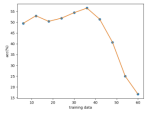
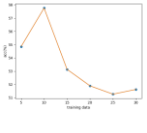
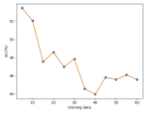
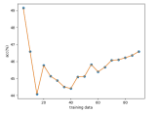
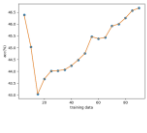
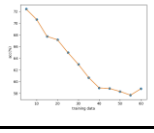
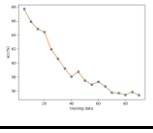
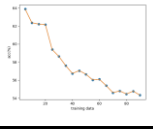
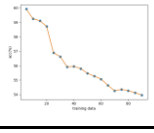
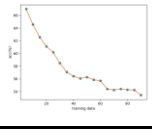
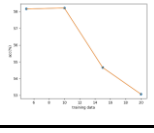
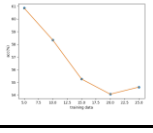
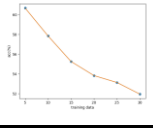
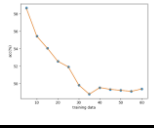
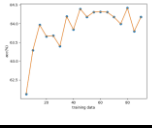
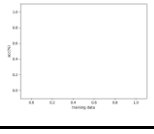
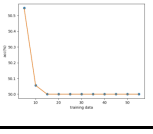
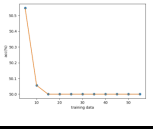
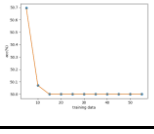
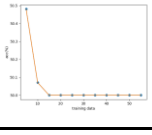
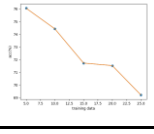
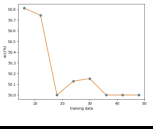
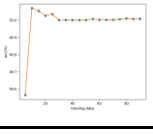
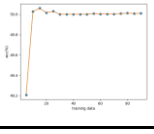
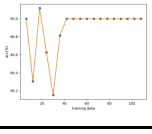
```
        range_num:
            トレーニングデータ数だけ繰り返すため用意したイテレータ
        point_num:
            中心点の数
        distance:
            距離
        func_dis:
            距離比較し,分類する関数
    """

    # 中心点を変数 center_point の数だけ非復元抽出
    point_list = random.sample(range(1, 50), point_num)
    print(point_list)

    # 中心点の数だけ繰り返す
    for center_point in point_list:
        x = np.loadtxt("pos_train/pos_train_{}.csv".format(center_point), delimiter=',')
        # 関数 euclidean_distance コール
        func_dis(range_num=range_num, x=x, distance=distance)

random.seed(1)
# 距離上 2 桁指定
first_ = 10
last_ = 20
# ディレクトリ作成
mkdir_supervised(first_, last_)
# 諸手順
range_num = range(0,200)
# ディレクトリ分離
random_train_test_split(range_num)

# 距離測定・識別
# print("距離:{}".format(distance))
multiple_points_distance(range_num,point_num, distance, func_dis=euclidean_distance)

print("OK")
```

可視化プログラム

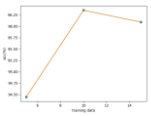
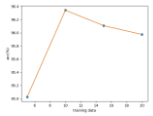
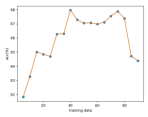
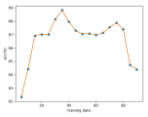
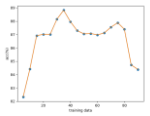
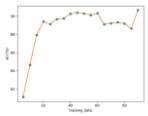
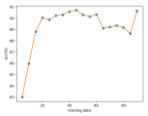
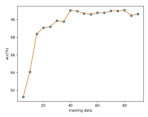
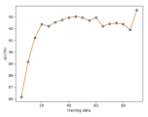
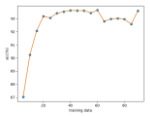
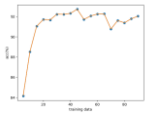
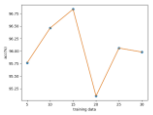
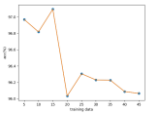
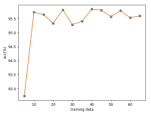
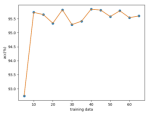
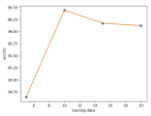
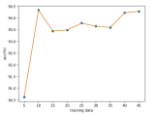
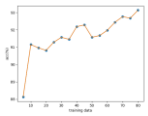
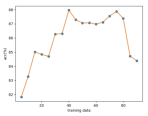
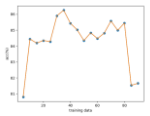
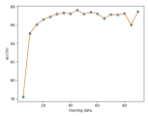
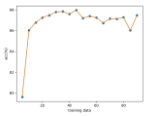
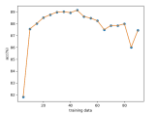
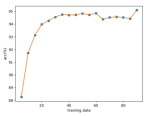
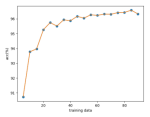
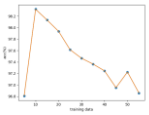
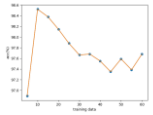
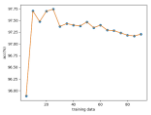
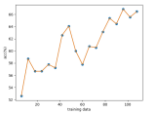
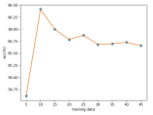
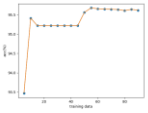
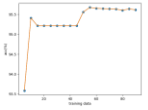
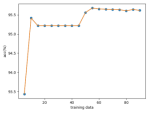
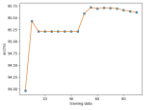
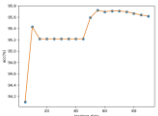
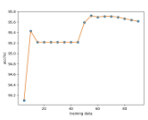
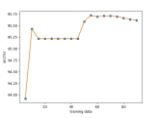
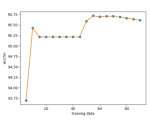
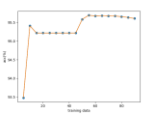
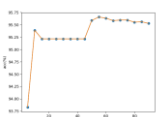
```
# coding: utf-8

import sys
import shutil
import os
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
from matplotlib import pylab

num = range(0,200)

y = []

x = np.array([])

for i in num[1:19]:#横軸の広さ
    c = i * 5
    x = np.append(x, c)

for i in num[1:19]:
    score = np.loadtxt( "score %d.csv" % (i), delimiter=',')
    a = np.average(score)
    a = a * 100
    y = np.append(y, a)

print(y)

pylab.xlabel('training data')
pylab.ylabel('acc(%)')
plt.plot(x, y, marker="o")
plt.plot(x, y)
plt.savefig( 'Img_mor_/acc_{}_{}_{}00.png'.format(sys.argv[3],
                                                  sys.argv[1], sys.argv[2]))
plt.show()
np.savetxt('result_accuracy.txt' , y, delimiter = ',')
```

付録 B

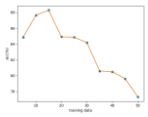
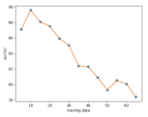
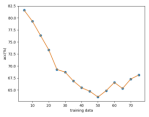
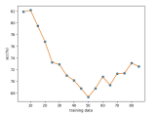
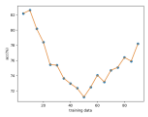
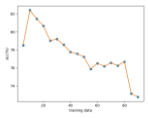
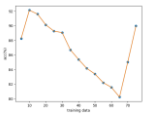
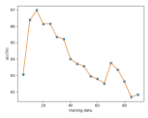
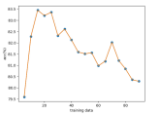
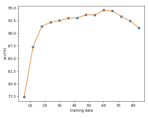
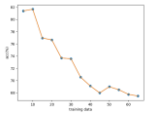
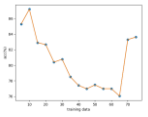
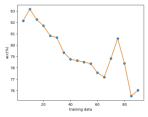
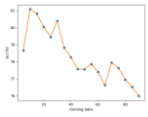
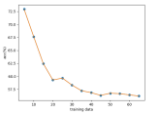
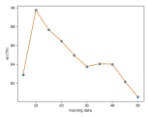
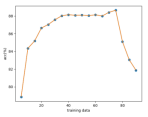
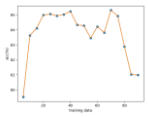
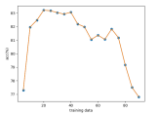
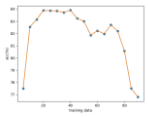
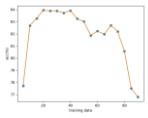
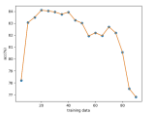
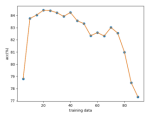
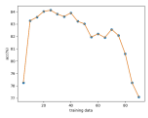
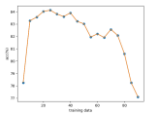
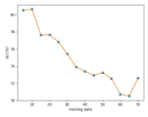
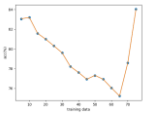
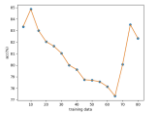
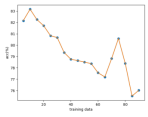
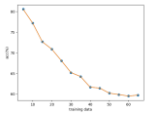
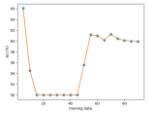
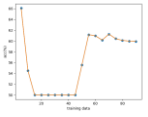
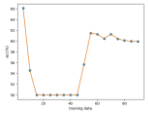
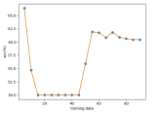
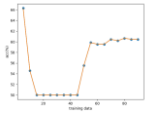
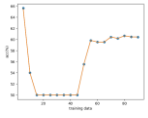
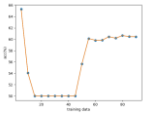
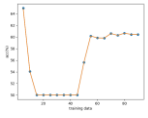
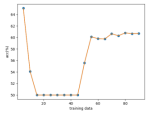
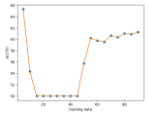
ミヤマオウム

		1000	1200	1500	1800	2000
ロジスティック回帰	中心点 1					
	中心点 3					
	中心点 10					
SVM	中心点 1					
	中心点 3					
	中心点 10					
ランダムフォレスト	中心点 1					
	中心点 3					
	中心点 10					

キジカッコウ

		1000	1200	1500	1800	2000
ロジスティック回帰	中心点 1					
	中心点 3					
	中心点 10					
SVM	中心点 1					
	中心点 3					
	中心点 10					
ランダムフォレスト	中心点 1					
	中心点 3					
	中心点 10					

ニュージーランドアオバズク

		1000	1200	1500	1800	2000
ロジスティック回帰	中心点 1					
	中心点 3					
	中心点 10					
SVM	中心点 1					
	中心点 3					
	中心点 10					
ランダムフォレスト	中心点 1					
	中心点 3					
	中心点 10					