平成 29 年度　特別研究報告書

機械学習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

T140496　藤岡 優也

指導教員　三好 力　教授

内容梗概

機械学習における教師あり学習では,多数のラベルありデータを訓練データとして学習に用いるほど認識精度が高くなることが知られている.しかし,ラベルありデータは人の手間や専門家の知識などが必要であることが多く,獲得にはコストがかかってしまう.それに対してラベルなしデータは低コストで大量に獲得することができる.もし低コストで大量に獲得が可能なラベルなしデータを訓練データとして用いることができるのであれば,低コストで運用可能な識別器となることができる.本研究では特徴量の類似性に着目し,少数のラベルありデータと,それに類似したラベルなしデータを訓練データとして用いる手法を検討し,低コストで運用可能な識別器を目指す.

目次

付録

第1章 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に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の平均ベクトル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.この方法に対して①ラベル付きデータに合成データを加えると識別率は向上するのか②加える合成データの数が多いほど識別率が向上するのか③合成データの平均ベクトルからの距離の閾値が近いほど識別率が向上するのか④識別率が向上する距離尺度はあるのか⑤識別率が向上する特徴抽出法はあるのか等の検討が求められる.④に対しては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なユークリッド距離を特徴量間距離尺度として用いている[1].⑤については事前実験で性能が高かったメル周波数ケプストラム係数(Mel-Frequency Cepstrum Coefficients : 以下 MFCC とする)を用いている.この方法によって決定した訓練データを教師あり学習を用いるサポートベクターマシン (Support Vector Machine : 以下 SVM とする)を用いて学習し,テストデータに対する識別率を求めて検証する.

本研究では類似性を測る尺度とこの手法によって決定した訓練データの数の関係性をニュージーランドに生息する野鳥の鳴き声のデータを例に,特徴量を MFCC,特徴量の類似性を測る距離尺度をベクトル間のユークリッド距離によって定義し,③を調べるため,特徴量の類似性として最適な距離が存在するのかどうかを検討する実験と,①②に対応する本研究で提案する手法によって決定した訓練データの数と識別率の関係を検討するため,距離尺度毎に教師の数を変動させて識別率の推移を測る実験を行った.

第2章 基本事項

2.1 SVM について

Support Vector Machine(SVM)とは[2-4],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,様々な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 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$x_i \in \mathcal{R}^d$が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.

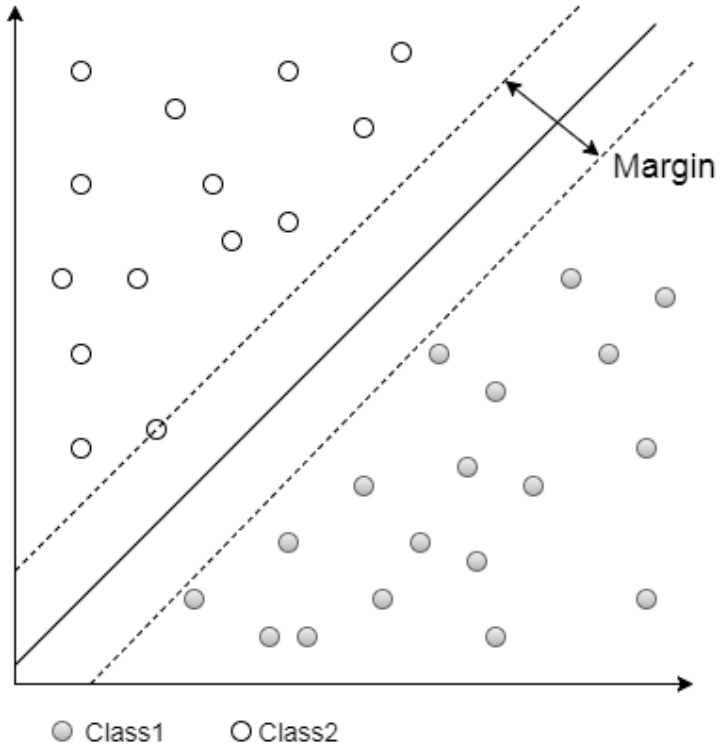


図 2.1 SVM の 2 クラス分類による分離平面

求める境界は以下の式で表される.

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

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 yi 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 yi = 1,もしくは yi = -1 をとるので,f(x)yi>0 が成り立つ.このことから,正しく分類できた場合の点 xi から,境界面までの距離は以下の式で表される.

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

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.

$$y_i(w^t x_i + w_0) = 1 \tag{2.4}$$

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,

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

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.

$$\text{minimize } \frac{1}{2}\|w\|^2$$

$$\text{s.t. } y_i(w^t x_i + w_0) \geq 1 \ (i = 1, \ldots, l) \tag{2.7}$$

この問題を解くため,Lagrange 未定乗数ベクトル$a \geq 0$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.

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

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.

$$w = \sum_{i=1}^{l} a_i y_i x_i \tag{2.9}$$

$$\sum_{i=1}^{l} a_i y_i = 0 \tag{2.10}$$

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.

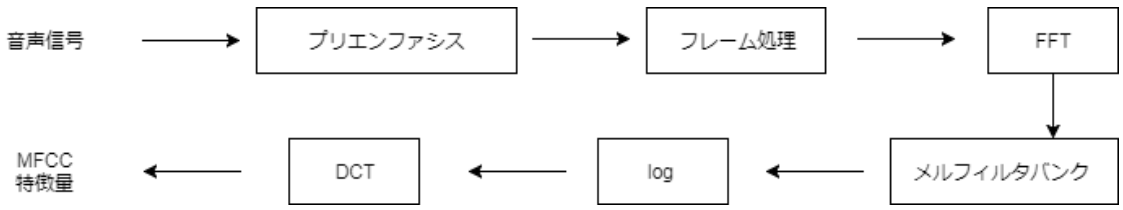
$$\text{maximize} \sum_{i=1}^{l} a_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l} a_i a_j y_i y_j x_i x_j$$

$$s.t. \sum_{i=1}^{l} a_i y_i = 0$$

$$a_i \geq 0 (i = 1, \ldots., \mathrm{l}) \tag{2.11}$$

2.3 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[5],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.MFCC の特徴量抽出の手順を図 2.2 に示す.



,図 2.2:MFCC の特徴量抽出の手順

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.

$$y(\mathrm{t}) = \mathrm{s}(\mathrm{t}) - \mathrm{p} \cdot \mathrm{s}(\mathrm{t}-1) \tag{2.12}$$

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ち,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.このメル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.

$$c_k = \sqrt{\frac{2}{L}}\sum_{l=1}^{L} logm(l)\, cos\left\{\left(l - \frac{1}{2}\right)\frac{k\pi}{L}\right\} \tag{2.13}$$

ここでkはケプストラム係数の次元を表しており,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.

2.4 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.[6]

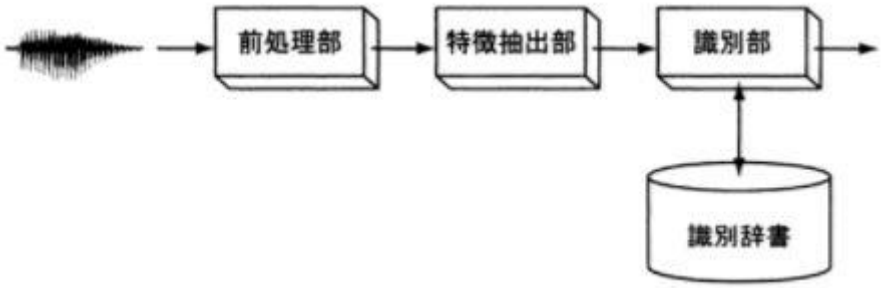


図 2.3:音声認識システムの構成

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,特徴抽出を行う.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.入力データは 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.

第3章 提案手法

本研究では前章の音声認識システムの識別部についての手法を提案する.一般的にSVMではクラスyが既知の特徴ベクトルxを訓練データとして学習を行うため,クラスyが未知であるラベルなしデータを訓練データとして用いることはできない.そのため何らかの方法によってラベルなしデータにクラス y を与える必要がある.そこで本研究ではラベルなしデータのクラス y を決定する手段として特徴ベクトル間の類似性を用いる.類似性を測る尺度は,機械学習では主に特徴ベクトル間距離を用いるアルゴリズムが多く,本研究では類似度を特徴ベクトル間のユークリッド距離によって決定する.

まず訓練用データとして少数のラベル付きデータを用意する.そのラベル付きデータからさらにランダムで少数の特徴ベクトルを選び取り,その平均ベクトル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.本研究ではこの手法によってラベルを定めたデータによる識別率と,訓練データの数を変動させることによる識別率を測定する実験を行った.

第4章 実験と考察

4.1 実験概要

本章では特徴量類似性と認識精度に関する実験と考察について述べる.提案手法に対し,①合成データの平均ベクトルからの距離の閾値が近いほど識別率が向上するのか②訓練データに加える合成データの数が多いほど識別率が向上するのかを検証するために 2 種類の実験を検討する.実験 1 では①について,実験 2 では②について検証する実験を行った.

使用するデータセットはニュージーランドに生息する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を使用した.なお,ミヤマオウムについては 3 種類以上の鳴き声の種類,キジカッコウについては 3 種類程度の鳴き声の種類,ニュージーランドアオバズクについては 1 種類の鳴き声のデータで構成されている. 音声データは全て wave ファイルで構成されており,全てのデータを MFCC によって特徴抽出を行い,得られた特徴ベクトルを全て csv ファイルで保存している.また,MFCC 特徴量は入力音声のサンプリングレート 22050 で 512 フレームごとに抽出している.このとき音声データごとの時間のずれによるベクトルの次元数の違いをなくすために,全ての音声データについて 1.62 秒分の MFCC 特徴量を抽出し,910 次元の特徴ベクトルを用いている.1.62 秒に満たないデータについてはゼロパディングを行い,超過したデータについては値の切り捨てを行った.また,機械学習と認識精度の測定には線形 SVM を用いており,チューニングは行っていない.

4.2 実験方法

4.2.1 実験 1

実験 1 は前項の①について検討するための実験である.3 種類の鳥から正例とする鳥を一種類定め,正例から 10 個,負例から 5 個ずつのデータをテストデータとして隔離する.次にクラスが既知の正例データからランダムで 3 つのデータを抽出し,その 3 つのデータの特徴ベクトルの平均ベクトルをとる.その後テストデータを除く全てのデータの特徴ベクトルと平均ベクトルとのユークリッド距離を測定し,設けた基準値 1000,1100,1200,1300,1400 以下であれば正例,それ以上であれば負例のラベルをつける.

この処理のフローを図 4.1 に示す.次にこの方法によって定めたラベルを付けたデータ全て訓練データとして用いて SVM のパラメータを決定し,テストデータに対する識別率を求める.また,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の値を測定値とする.実験 1 の全体の処理のフローを図 4.2 に示す.なお,ここでは図 4.1 に示した処理を各基準値でデータセットを作成としている.

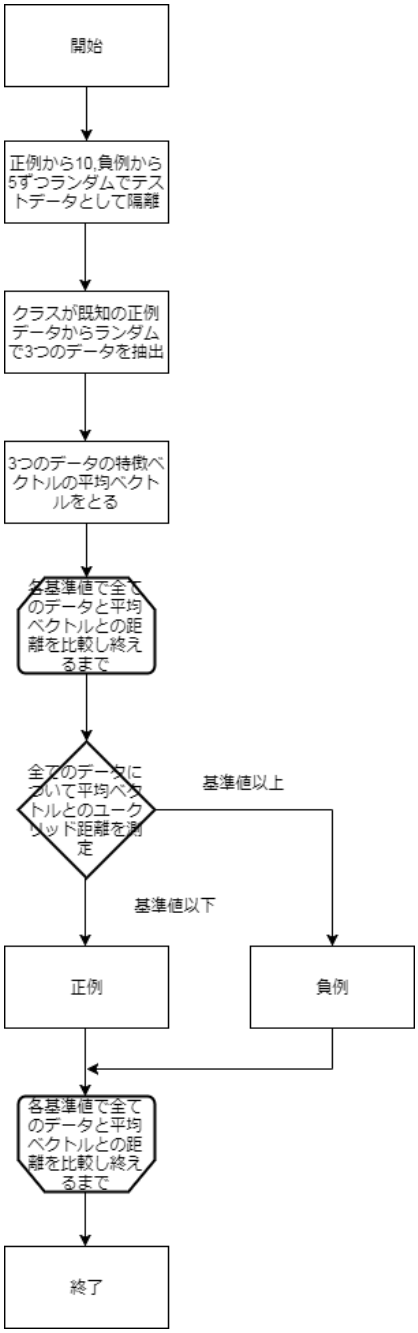

開始
正例から10,負例から5ずつランダムでテストデータとして隔離
クラスが既知の正例データからランダムで3つのデータを抽出
3つのデータの特徴ベクトルの平均ベクトルをとる
各基準値で全てのデータと平均ベクトルとの距離を比較し終えるまで
全てのデータについて平均ベクトルとのユークリッド距離を測定
基準値以上
基準値以下
正例
負例
各基準値で全てのデータと平均ベクトルとの距離を比較し終えるまで
終了

図 4.1:処理のフロー

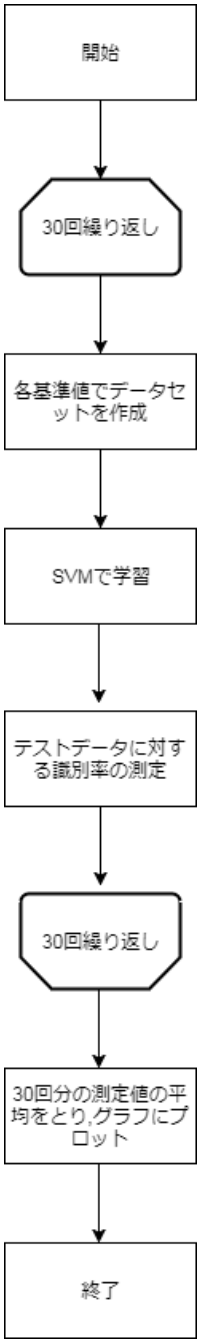


図 4.2:実験 1 の処理のフロー

4.2.2 実験 2

実験 2 は前項の②について検証するための実験である.実験 1 の各ユークリッド距離に対して,正例データと負例データを比べて少ない方を 3 で割った数を測定回数とし,訓練データの数と認識精度の推移を訓練データ 6 個おきに正例データミヤマオウムとニュージーランドアオバズクの場合について測定した.処理のフローを図 4.3 に示す.また,ミヤマオウムの場合については他のデータと比べてデータ間距離が離れる傾向があったため,ユークリッド距離を 200 増やして測定している.データセットの作成方法については実験 1 と同じで,30 回繰り返した場合の平均を測定値とする.

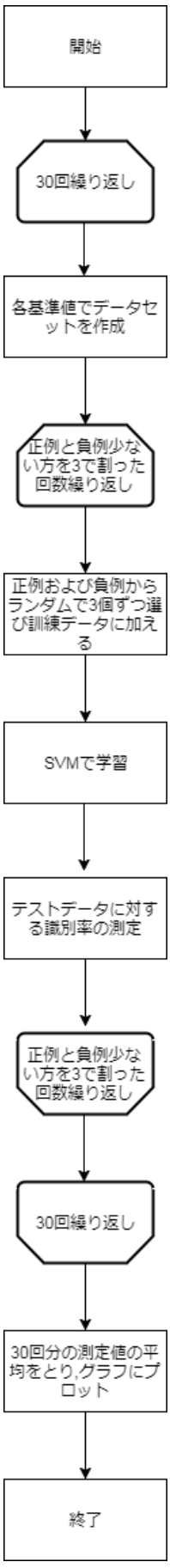


図 4.3:実験 2 の処理のフロー

4.3 実験環境

本実験にて使用した実験環境を以下に示す.実装は Python にて行い,配列処理等に Numpy,グラフ処理に Matplotlib,SVM での学習に Sciket-learn,MFCC による特徴抽出は Librosa によって行った.

Ubuntu 16.04 LTS

Python 3.6.1 Anaconda 4.4.0

Numpy 1.12.1

Matplotlib 2.0.2

Sciket-learn 0.18.1

Librosa 0.5.1

4.4 実験結果

4.4.1 実験 1 の結果

実験 1 の結果を図 4.4,4.5,4.6 に示す.なお,横軸はユークリッド距離を表し,縦軸は認識精度を表している.

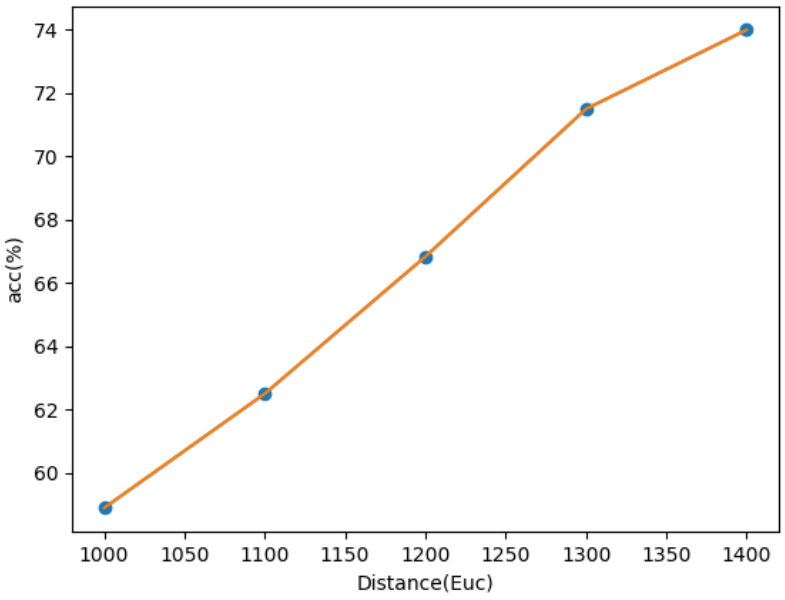


図 4.4 正例データミヤマオウムの場合の距離尺度と認識精度の推移

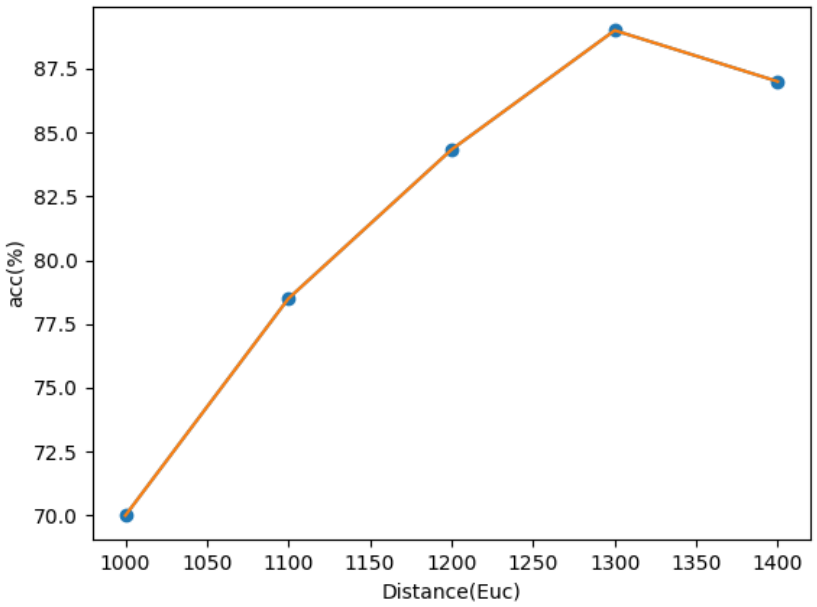


図 4.5 正例データキジカッコウの場合の距離尺度と認識精度の推移

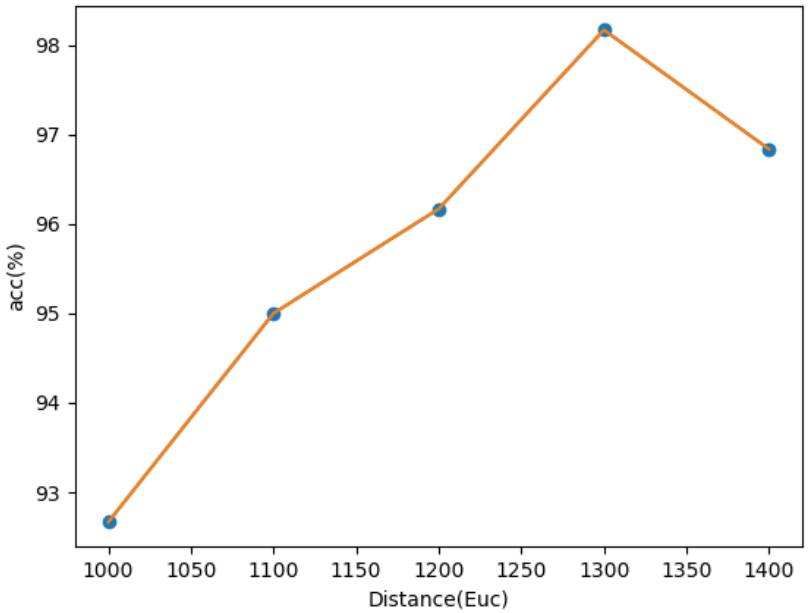


図 4.6 正例データニュージーランドアオバズクの場合の距離尺度と認識精度の推移

4.4.2　実験 2 の結果

実験 2 の結果を図 4.7~図 4.16 に示す.なお,グラフの横軸は学習に用いた訓練データ数,縦軸は認識精度を示している.

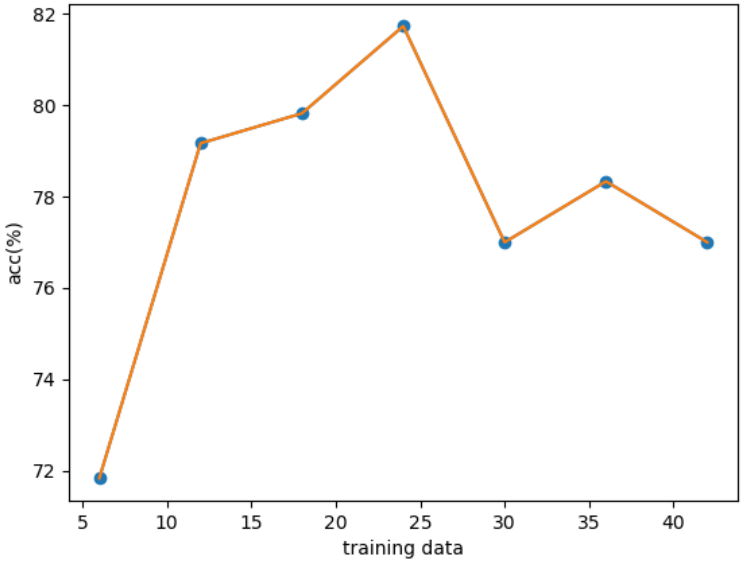


図 4.7:正例データミヤマオウム,ユークリッド距離 1200 での訓練データと認識精度の推移

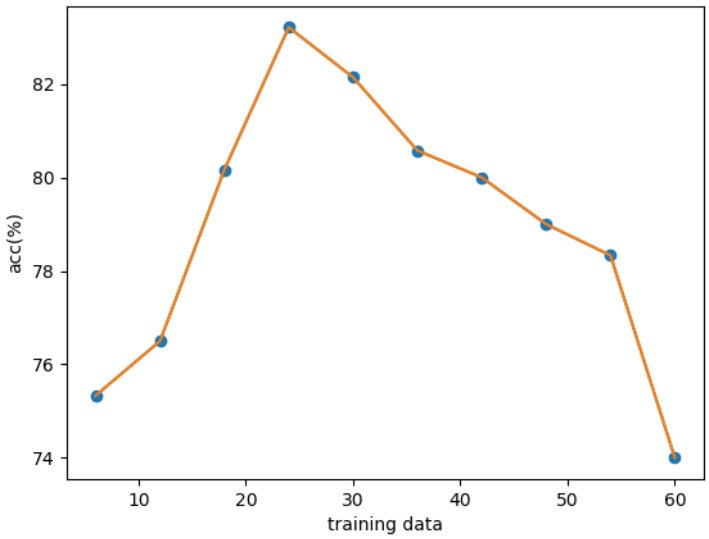


図 4.8:正例データミヤマオウム,ユークリッド距離 1300 での訓練データと認識精度の推移

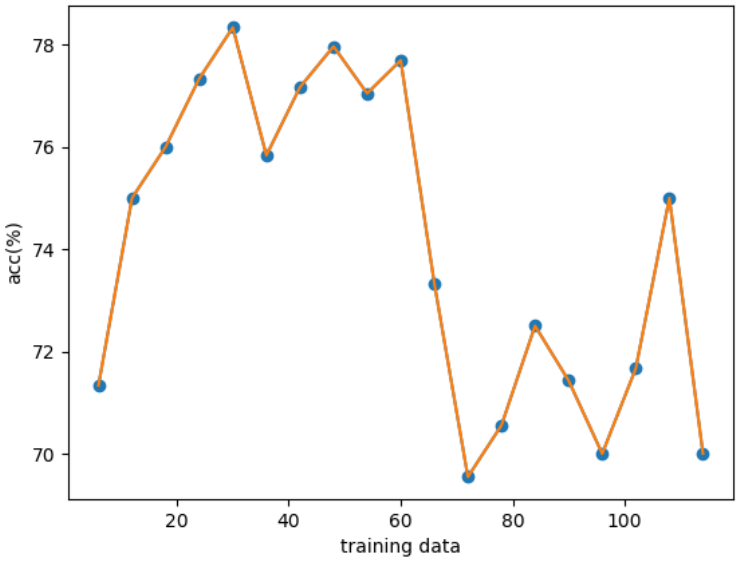


図 4.9:正例データミヤマオウム,ユークリッド距離 1400 での訓練データと認識精度の推移

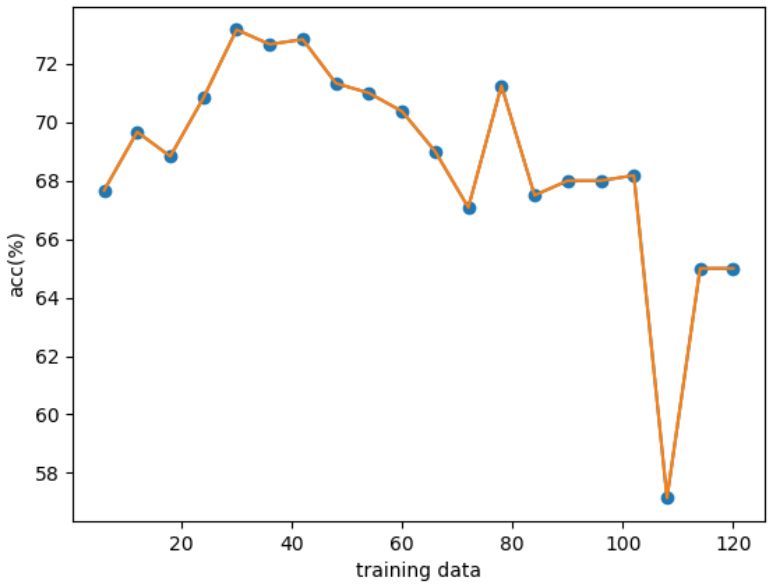


図 4.10:正例データミヤマオウム,ユークリッド距離 1500 での訓練データと認識精度の推移

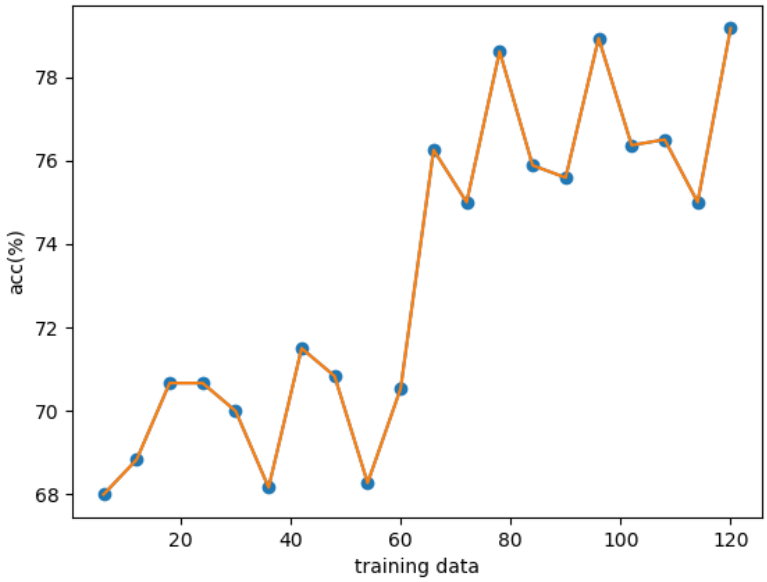


図 4.11:正例データミヤマオウム,ユークリッド距離 1600 での訓練データと認識精度の推移

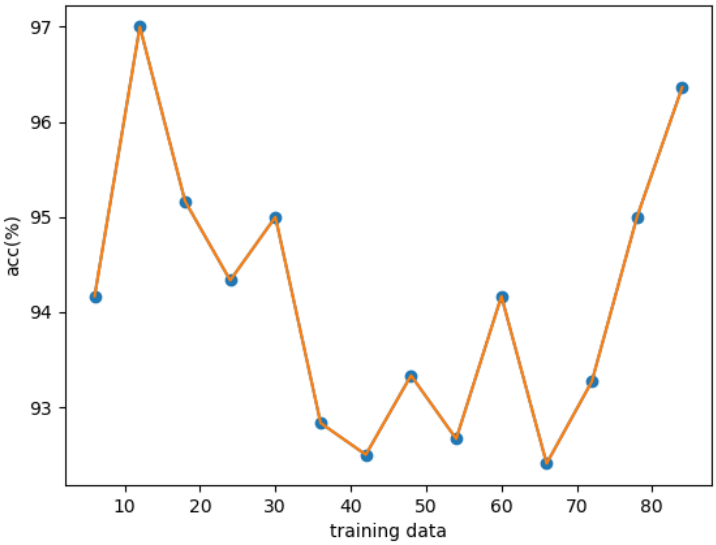


図 4.12:正例データニュージーランドアオバズク,ユークリッド距離 1000 での訓練データと認識精度の推移

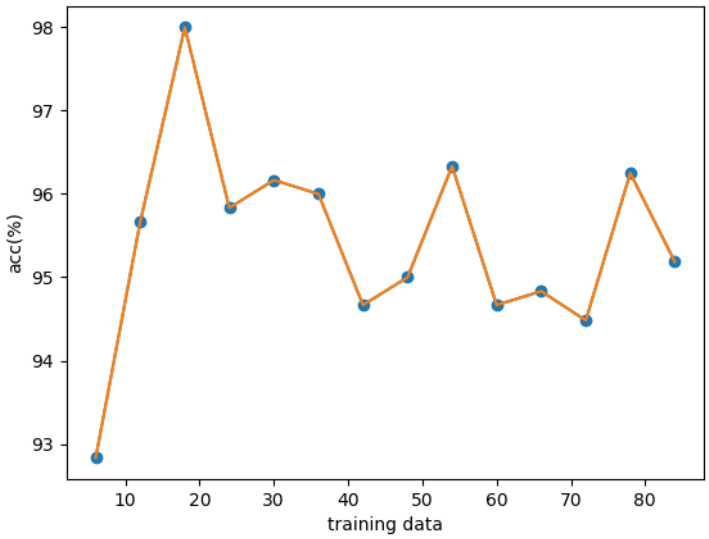


図 4.13:正例データニュージーランドアオバズク,ユークリッド距離 1100 での訓練データと認識精度の推移

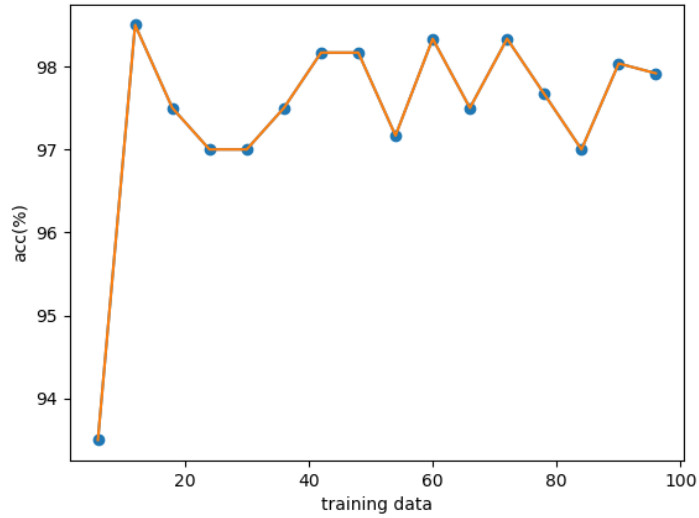


図 4.14:正例データニュージーランドアオバズク,ユークリッド距離 1200 での訓練データと認識精度の推移

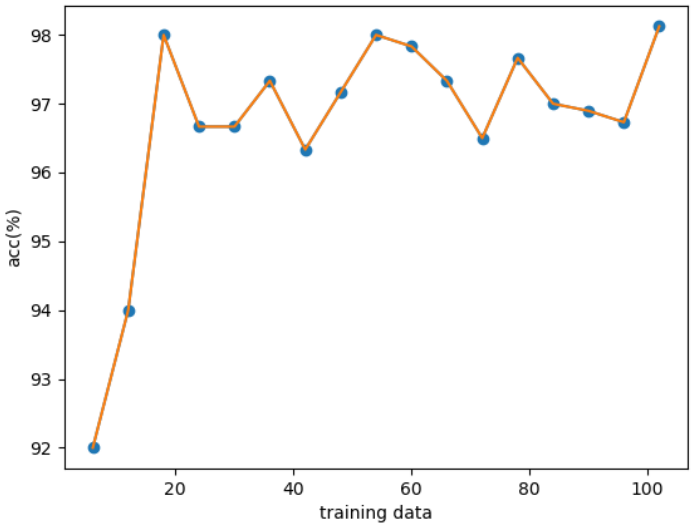


図 4.15:正例データニュージーランドアオバズク,ユークリッド距離 1300 での訓練データと認識精度の推移

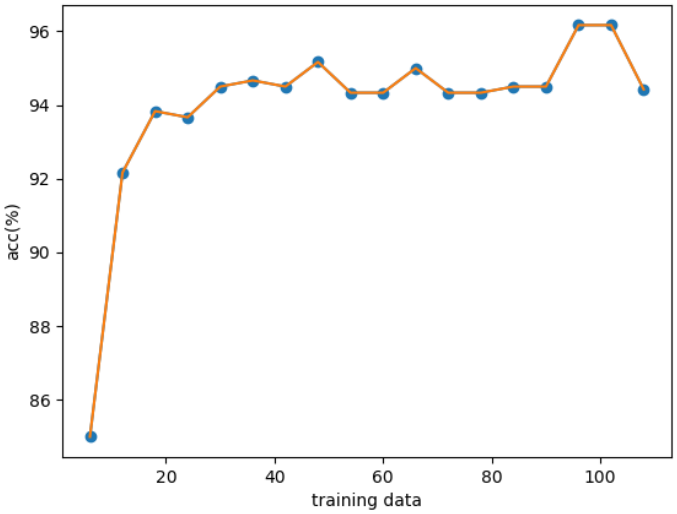


図 4.16:正例データニュージーランドアオバズク,ユークリッド距離 1400 での訓練データと認識精度の推移

4.5 考察

合成データの平均ベクトルからの距離の閾値が近いほど識別率が向上するのかどうかを検討するために行った実験 1 の結果から,距離が近いほど識別率が向上するという予想に反して,距離が近いものに正例のラベルをつけると認識精度が上がるわけではなく,認識精度が最高となる最適な距離が存在することがわかった.これは,距離が近すぎる場合には実際のラベルが正例の場合でも負例であるとして学習させており,距離が遠すぎる場合であっても逆のことが言えるからであると考えられる.本研究では少数のラベル付きデータの平均ベクトルからの距離によって訓練データを決定している.今回の実験ではミヤマオウム,キジカッコウは複数の種類の鳴き方のデータが混ざり合っており,データが広く分布していると考えられることから,平均ではなく分散や標準偏差などを加味した方法も検討することが今後の課題として挙げられる.

実験 2 は訓練データに加える合成データの数が多いほど識別率が向上するのかを検証するために行った.図 4.7~図 4.11 を見ると凸型のグラフに,図 4.12~図 4.16 を見るとある一点から横並びのグラフになっていることがわかる.この実験では加える合成データの数が多いほど識別率が良くなるという結果が推測できるが,この推測に反してミヤマオウム,ニュージーランドアオバズクの両方について距離の大小と訓練データ数による認識精度の向上を確認することはできない.これは,ミヤマオウムの場合鳴き声データは 3 種類以上の鳴き声データで構成されており,距離の大小のみによってラベル付けを行うことは非常に困難であるためだと考えられる.一方でニュージーランドアオバズクの場合は 1 種類のみの鳴き声で構成されており,こちらの場合は距離 1300 以降である一点を境にして訓練データ数を増やしていってもほぼ認識精度はほぼ横並びになっていることがわかる.複数の鳴き方をする鳥に関してはその鳴き方ごとにラベル付けを行うことが望ましいことが考えられることから,提案手法による訓練データの決定によって訓練データ数を無数に増やしていっても識別率の向上は見込めないことが推測できるが,識別率が下がることはないため,訓練データの数は多数あればよいという結論に至る.

第5章 おわりに

本研究では,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認識精度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の平均ベクトル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提案し,実験を行った.このとき距離は近ければ良いというわけではなく,識別率が最高となる最適な距離が存在することが確認できた.また,訓練データの数を変動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方をする鳥に関しては,鳴き声ごとにラベルをつけることが望ましいため,有用な結果を得ることができなかったが,鳴き方の種類が一種類のみのニュージーランドアオバズクの場合はこの手法によって訓練データ数を増やしていくことによる認識精度の向上は見込めなかったが,認識精度が下がるということは起こらなかったため,訓練データの数は多いほど良いという結論に至ることができる.

今後の研究では鳴き方の種類が増えると鳴き声が広く分布していることが考えられることから平均ベクトルではなく分散や標準偏差を加味した距離の決定や,鳴き声の種類ごとにラベルをつけて実験を行う必要があると考えられる.

謝辞

本論文を作成するにあたり,多くのご指導,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]峯松 信明, 志甫 淳, 村上 隆夫, 丸山 和孝, 広瀬 啓吉, “音声の構造的氷像とその距離尺度”, 電子情報通信学会技術研究報告. SP, 音声 105(98), 9-12(2005)

[2] “サポートベクターマシン(SVM)”, http://www.sist.ac.jp/~kanakubo/research/neuro/supportvectormachine.html, 2018/01/08

[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/2017/01/24/201702,2018/01/08

[4]中川裕志(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,2018/01/08

[5]”メル周波数ケプストラム係数(MFCC)”, http://aidiary.hatenablog.com/entry/20120225/1330179868,2018/01/08

[6]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

付録

Mfcc.py

```
import numpy as np
import librosa
nums = range(0,200)
num = np.zeros((13,70))

for i in nums[1:61]:
 y, sr = librosa.load("renamed/Morepork %d.wav" % (i))
 mfcc = librosa.feature.mfcc(y=y, sr=sr , n_mfcc = 13)
 c = mfcc.shape[1]
 if 70 >= c:
#ゼロパディング
  mfccs = np.zeros(num.shape)
  mfccs[:mfcc.shape[0],:mfcc.shape[1]] = mfcc

 if 70<c:
#70 列目以降の要素を全て削除
  mfccs = np.delete(mfcc,np.s_[70:],1)

 mfccs = mfccs.transpose()
 mfccs = mfccs.reshape(-1,)
 np.savetxt('Morepork %d.csv' % (i), mfccs, delimiter = ',')
```

dataset.py

```
import random
import numpy as np
import shutil
import os

#テストデータを隔離
#正例とする鳥の鳴き声は正例テストフォルダ
#それ以外は負例テストフォルダ

list = []
i = 1
j = 1
kea_num = range(0,200)
#正負分離ディレクトリ削除
shutil.rmtree("pos_test")
shutil.rmtree("pos_train")
shutil.rmtree("neg_test")
shutil.rmtree("neg_train")

#尺度内教師ディレクトリ削除
shutil.rmtree("pos_train1000")
shutil.rmtree("pos_train1100")
shutil.rmtree("pos_train1200")
shutil.rmtree("pos_train1300")
shutil.rmtree("pos_train1400")

#尺度外ディレクトリ削除
shutil.rmtree("neg_train1000")
shutil.rmtree("neg_train1100")
shutil.rmtree("neg_train1200")
shutil.rmtree("neg_train1300")
shutil.rmtree("neg_train1400")

#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
os.makedirs("pos_train1000")
os.makedirs("pos_train1100")
os.makedirs("pos_train1200")
os.makedirs("pos_train1300")
os.makedirs("pos_train1400")

#尺度外教師ディレクトリ作成
os.makedirs("neg_train1000")
os.makedirs("neg_train1100")
os.makedirs("neg_train1200")
os.makedirs("neg_train1300")
os.makedirs("neg_train1400")

#1 から 60 までの数字をランダムで 10 個配列に入れる。

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 k_num in kea_num[1:61]:
   if k_num in list:
      shutil.copyfile("Kea/Kea %d.csv" % (k_num), "pos_test/pos_test %d.csv" % (i))
      i = i + 1
   else:
      shutil.copyfile("Kea/Kea %d.csv" % (k_num), "pos_train/pos_train %d.csv" % (j))
      j = j + 1

i = 1
j = 1

list02 = random.sample(range(1,60), 5)

#負例
for k_num in kea_num[1:61]:
   if k_num in list02:
      shutil.copyfile("LTCuckoo/LTCuckoo %d.csv" % (k_num), "neg_test/neg_test
%d.csv" % (i))
      i = i + 1
   else:
      shutil.copyfile("LTCuckoo/LTCuckoo %d.csv" % (k_num), "neg_train/neg_train
%d.csv" % (j))
      j = j + 1

#負例
for k_num in kea_num[1:61]:
   if k_num in list02:
      shutil.copyfile("Morepork/Morepork %d.csv" % (k_num), "neg_test/neg_test
%d.csv" % (i))
      i = i + 1
   else:
      shutil.copyfile("Morepork/Morepork %d.csv" % (k_num), "neg_train/neg_train
%d.csv" % (j))
      j = j + 1

i = 1
j = 1


#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。
list01 = []

list01 = random.sample(range(1,50), 3)

a = list01[0]
b = list01[1]
c = list01[2]

pos_1 = np.loadtxt('pos_train/pos_train %d.csv' % a, delimiter=',')
pos_2 = np.loadtxt('pos_train/pos_train %d.csv' % b, delimiter=',')
pos_3 = np.loadtxt('pos_train/pos_train %d.csv' % c, delimiter=',')

sum = pos_1 + pos_2 + pos_3
x = sum/3


#正例フォルダ、負例フォルダそれぞれで 3 平均ベクトルとユークリッド距離を比較

#正例教師から比較@1000
for k_num in kea_num[1:51]:
   y = np.loadtxt('pos_train/pos_train %d.csv' % (k_num), delimiter=',')
   D = np.linalg.norm(x-y)
   if D < 1200:
      shutil.copyfile("pos_train/pos_train %d.csv" % (k_num), "pos_train1000/pos_train
%d.csv" % (i))
      i = i + 1
   else:
      shutil.copyfile("pos_train/pos_train %d.csv" % (k_num),
"neg_train1000/neg_train %d.csv" % (j))
      j = j + 1
```

```
#負例教師から比較@1000
for k_num in kea_num[1:101]:
    y = np.loadtxt('neg_train/neg_train %d.csv' % (k_num), delimiter=',')
    D = np.linalg.norm(x-y)
    if D < 1200:
        shutil.copyfile("neg_train/neg_train %d.csv" % (k_num),
"pos_train1000/pos_train %d.csv" % (i))
        i = i + 1
    else:
        shutil.copyfile("neg_train/neg_train %d.csv" % (k_num),
"neg_train1000/neg_train %d.csv" % (j))
        j = j + 1

i = 1
j = 1

#正例教師から比較@1100
for k_num in kea_num[1:51]:
    y = np.loadtxt('pos_train/pos_train %d.csv' % (k_num), delimiter=',')
    D = np.linalg.norm(x-y)
    if D < 1300:
        shutil.copyfile("pos_train/pos_train %d.csv" % (k_num), "pos_train1100/pos_train
%d.csv" % (i))
        i = i + 1
    else:
        shutil.copyfile("pos_train/pos_train %d.csv" % (k_num), "neg_train1100/neg_train
%d.csv" % (j))
        j = j + 1

#負例教師から比較@1100
for k_num in kea_num[1:101]:
    y = np.loadtxt('neg_train/neg_train %d.csv' % (k_num), delimiter=',')
    D = np.linalg.norm(x-y)
    if D < 1300:
        shutil.copyfile("neg_train/neg_train %d.csv" % (k_num), "pos_train1100/pos_train
%d.csv" % (i))
        i = i + 1
    else:
        shutil.copyfile("neg_train/neg_train %d.csv" % (k_num),
"neg_train1100/neg_train %d.csv" % (j))
        j = j + 1

i = 1
j = 1

#正例教師から比較@1200
for k_num in kea_num[1:51]:
    y = np.loadtxt('pos_train/pos_train %d.csv' % (k_num), delimiter=',')
    D = np.linalg.norm(x-y)
    if D < 1400:
        shutil.copyfile("pos_train/pos_train %d.csv" % (k_num), "pos_train1200/pos_train
%d.csv" % (i))
        i = i + 1
    else:
        shutil.copyfile("pos_train/pos_train %d.csv" % (k_num),
"neg_train1200/neg_train %d.csv" % (j))
        j = j + 1

#負例教師から比較@1200
for k_num in kea_num[1:101]:
    y = np.loadtxt('neg_train/neg_train %d.csv' % (k_num), delimiter=',')
    D = np.linalg.norm(x-y)
    if D < 1400:
        shutil.copyfile("neg_train/neg_train %d.csv" % (k_num),
"pos_train1200/pos_train %d.csv" % (i))
        i = i + 1
    else:
        shutil.copyfile("neg_train/neg_train %d.csv" % (k_num),
"neg_train1200/neg_train %d.csv" % (j))
        j = j + 1

i = 1
j = 1

#正例教師から比較@1300
for k_num in kea_num[1:51]:
    y = np.loadtxt('pos_train/pos_train %d.csv' % (k_num), delimiter=',')
    D = np.linalg.norm(x-y)
    if D < 1500:
        shutil.copyfile("pos_train/pos_train %d.csv" % (k_num), "pos_train1300/pos_train
%d.csv" % (i))
        i = i + 1
    else:
        shutil.copyfile("pos_train/pos_train %d.csv" % (k_num),
"neg_train1300/neg_train %d.csv" % (j))
        j = j + 1

#負例教師から比較@1300
for k_num in kea_num[1:101]:
    y = np.loadtxt('neg_train/neg_train %d.csv' % (k_num), delimiter=',')
    D = np.linalg.norm(x-y)
    if D < 1500:
        shutil.copyfile("neg_train/neg_train %d.csv" % (k_num),
"pos_train1300/pos_train %d.csv" % (i))
        i = i + 1
    else:
        shutil.copyfile("neg_train/neg_train %d.csv" % (k_num),
"neg_train1300/neg_train %d.csv" % (j))
        j = j + 1

i = 1
j = 1

#正例教師から比較@1400
for k_num in kea_num[1:51]:
    y = np.loadtxt('pos_train/pos_train %d.csv' % (k_num), delimiter=',')
    D = np.linalg.norm(x-y)
    if D < 1600:
        shutil.copyfile("pos_train/pos_train %d.csv" % (k_num), "pos_train1400/pos_train
%d.csv" % (i))
        i = i + 1
    else:
        shutil.copyfile("pos_train/pos_train %d.csv" % (k_num),
"neg_train1400/neg_train %d.csv" % (j))
        j = j + 1

#負例教師から比較@1400
for k_num in kea_num[1:101]:
    y = np.loadtxt('neg_train/neg_train %d.csv' % (k_num), delimiter=',')
    D = np.linalg.norm(x-y)
    if D < 1600:
        shutil.copyfile("neg_train/neg_train %d.csv" % (k_num),
"pos_train1400/pos_train %d.csv" % (i))
        i = i + 1
    else:
        shutil.copyfile("neg_train/neg_train %d.csv" % (k_num),
"neg_train1400/neg_train %d.csv" % (j))
        j = j + 1
```

SVM1.py

```
import numpy as np
from sklearn.metrics import confusion_matrix
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
import os
import re

num = range(0,200)

#2 クラス分類のラベル
label = 0 #正例
label2 = 1 #負例

n = 1000#ユークリッド距離

X_train = []
y_train = []
X_test = []
y_test = []


#正例教師
dir = 'pos_train%d' % (n)#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

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 = count + 1

count = count + 1

for i in num[1:count]:#ファイルの数だけループ
 pos = np.loadtxt( "pos_train%d/pos_train %d.csv" % (n , i), delimiter=',')
 X_train.append(pos)
 y_train.append(label)

#負例教師

dir = 'neg_train%d' % (n)#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

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:
        count = count + 1

#print(count)

count = count + 1

for i in num[1:count]:#ファイルの数だけループ
 neg = np.loadtxt( "neg_train%d/neg_train %d.csv" % (n , i), delimiter=',')
 X_train.append(neg)
 y_train.append(label2)

#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

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 = count + 1


count = count + 1#for 文のループ回数に合わせるために+1 としている

for i in num[1:count]:#ファイルの数だけループ
 pos = np.loadtxt( "pos_test/pos_test %d.csv" % (i), delimiter=',')
 X_test.append(pos)
 y_test.append(label)

#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

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 = count + 1

count = count + 1#for 文のループ回数に合わせるために+1 としている

for i in num[1:count]:#ファイルの数だけループ
 neg = np.loadtxt( "neg_test/neg_test %d.csv" % (i), delimiter=',')
 X_test.append(neg)
 y_test.append(label2)

svc = LinearSVC(C=1.0)


svc.fit(X_train,y_train)
#訓練データとラベルを入れる。第一変数が訓練データ、第二変数がラベル

print('Test score: {:.3f}'.format(svc.score(X_test, y_test)))

score = svc.score(X_test, y_test)

t = np.loadtxt( "score %d.csv" % (n), delimiter=',')

t = np.append(t, score)

np.savetxt('score %d.csv' % (n), t, delimiter = ',')
```

SVM2.py

```
import numpy as np
from sklearn.metrics import confusion_matrix
from sklearn.svm import LinearSVC
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
import os
import re
import math
import random

num = range(0,200)

#2 クラス分類のラベル
label = 0 #正例
label2 = 1 #負例

n = 1000#ユークリッド距離

X_train = []#訓練教師
y_train = []#訓練ラベル
X_test = []#テスト教師
y_test = []#テストラベル

#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

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 = count + 1

count = count + 1#for 文のループ回数に合わせるために+1 としている

for i in num[1:count]:#ファイルの数だけループ
 pos = np.loadtxt( "pos_test/pos_test %d.csv" % (i), delimiter=',')
 X_test.append(pos)
 y_test.append(label)

#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

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 = count + 1


count = count + 1#for 文のループ回数に合わせるために+1 としている

for i in num[1:count]:#ファイルの数だけループ
 neg = np.loadtxt( "neg_test/neg_test %d.csv" % (i), delimiter=',')
 X_test.append(neg)
 y_test.append(label2)



#正例教師フォルダのファイルの数を獲得
dir = 'pos_train%d' % (n)#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
pos_count = 0# カウンタの初期化

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
        pos_count = pos_count + 1

print(pos_count)
```

```
#pos_count = pos_count + 1#for 文のループ回数に合わせるために+1 としている

#負例教師フォルダのファイルの数を獲得
dir = 'neg_train%d' % (n)#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
neg_count = 0# カウンタの初期化

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:
      neg_count = neg_count + 1

print(neg_count)

#neg_count = neg_count + 1

rnum = 0
#pos_count と neg_count の大きい方/3 をループの回数とする
if pos_count < neg_count:
   rnum = math.floor(pos_count//3)
else:
   rnum = math.floor(neg_count//3)

print(rnum)

rnum = int(rnum)

print(rnum)

pos_num = pos_count

neg_num = neg_count

#for 分ループに合わせるための処理
neg_count = neg_count + 1

pos_count = pos_count + 1

rnum = rnum + 1

list = []


for j in num[1:rnum]:

#正例数分だけ配列に代入
  k = j * 3

  list = random.sample(range(1,pos_count), k)


  for i in num[1:pos_count]:#ファイルの数だけループ
       if i in list :
          pos = np.loadtxt( "pos_train%d/pos_train %d.csv" % (n , i), delimiter=',')
          X_train.append(pos)
          y_train.append(label)
#負例教師分だけ配列に代入
  list = random.sample(range(1,neg_count), k)
#   print(list)
  for i in num[1:neg_count]:#ファイルの数だけループ
       if i in list :
          neg = np.loadtxt( "neg_train%d/neg_train %d.csv" % (n , i), delimiter=',')
          X_train.append(neg)
          y_train.append(label2)
  svc = LinearSVC(C=1.0)

  svc.fit(X_train,y_train)
#訓練データとラベルを入れる。第一変数が訓練データ、第二変数がラベル

  score = svc.score(X_test, y_test)

  print(score)
  t = np.loadtxt( "score %d.csv" % (j), delimiter=',')

  t = np.append(t, score)
  np.savetxt('score %d.csv' % (j), t, delimiter = ',')
  X_train = []
  y_train = []#初期化
```

```
prediction = svc.predict(X_test)
cm = confusion_matrix(y_test,prediction)#混合行列を出力
```