平成26年度茨城大学工学部情報工学科卒業論文

Stacked Denoising Autoencoderを利用した語義曖昧性解消の領域適応

平成27年2月9日

茨城大学工学部情報工学科

河野和平(11T4077H)

指導教員 新納浩幸 准教授

平成 26 年度茨城大学工学部情報工学科卒業研究

Stacked Denoising Autoencoder を利用した 語義曖昧性解消の領域適応

著者： 河野 和平 (11T4077H)
指導教員：新納 浩幸 准教授

論文要旨

本論文では深層学習 (Deep Learning) のモデルのひとつである Stacked Denoising Autoencoder(SdA) を利用し，語義曖昧性解消 (Word Sense Disambiguation, WSD) の領域適応の問題解決を図る．

WSD は文章中に出現する多義語の語義を一意に識別するタスクである．一般に WSD は分類問題として処理され，教師有り学習などで語義を識別する．自然言語処理のタスクにおける分類問題において機械学習を用いる際，訓練データとテストデータは同じ領域のコーパスから得られたデータであることが前提であった．しかし，WSD のタスクにおいて，これらは異なる領域から得られる場合がある．この場合，ある領域 (ソース領域) のコーパス S から得た訓練データによって学習した分類器では，それとは別の領域 (ターゲット領域) のコーパス T から得たテストデータを精度良く識別することが困難である．そこで，ソース領域の訓練データで学習した分類器をターゲット領域のテストデータに適用できるようチューニングすることを領域適応と呼ぶ．

WSD の領域適応の問題点として，異なる領域から得られた訓練データとテストデータでは同じ単語に関するデータであっても，その素性に大きな差が生じてしまうことが挙げられる．これは，異なる領域の文章はその話題や内容に差があり，共起語が大きく異なることによる影響であると考えられる．そこで，深層学習によって当該問題の解決を試みる．深層学習はニューラルネットワーク (Neural network，NN) を用いて，データの抽象的表現を獲得する手法である．ターゲット領域に頻出しない共起語などから獲得したデータで分類器を学習する場合も，深層学習によって獲得した抽象的なデータによって学習することで，上述した問題を緩和できると考えた．本論文では複数存在する深層学習モデルのうち，SdA を利用した．

実験では，現代日本語書き言葉均衡コーパス (BCCWJ) の Yahoo!知恵袋 (OC) 及び書籍 (PB) のデータのうち 16 単語を対象とし，領域適応は OC → PB と PB → OC の双方向を行った．はじめに，特異値分解を利用した次元削減法 (Latent Semantic Indexing, LSI) と SdA により獲得した素性とでサポートベクターマシン (Support Vector Machine, SVM) の識別精度を比較した．実験の結果，SdA を利用したデータは，各領域適応で LSI よりも良い精度が得られた．次に，SdA で獲得するデータの次元数に関する実験も行った．その結果，抽出する素性の次元数により，識別精度に大きな変化が見られた．

今後，SdA で獲得する素性の次元数に関する追実験に加え，SdA の各パラメータに関する実験を行い，適切な値を考察する必要がある．

目 次

表 目 次

図 目 次

第1章　序論

1.1　概要

本論文では深層学習 (Deep Learning) のモデルのひとつである Stacked Denoising Autoencoder(SdA) を利用し，語義曖昧性解消 (Word Sense Disambiguation, WSD) の領域適応の問題解決を図る．

WSD は文章中に出現する多義語の語義を一意に識別するタスクである．一般に WSD は分類問題として処理され，教師有り学習などで語義を識別する．自然言語処理のタスクにおける分類問題において機械学習を用いる際，訓練データとテストデータは同じ領域のコーパスから得られたデータであることが前提であった．しかし，WSD のタスクにおいて，これらは異なる領域から得られる場合がある．この場合，ある領域 (ソース領域) のコーパス S から得た訓練データによって学習した分類器では，それとは別の領域 (ターゲット領域) のコーパス T から得たテストデータを精度良く識別することが困難である．そこで，ソース領域の訓練データで学習した分類器をターゲット領域のテストデータに適用できるようチューニングすることを領域適応 [1] と呼び，本研究室においても関連研究が行われている [2][3].

WSD の領域適応の問題点として，異なる領域から得られた訓練データとテストデータでは同じ単語に関するデータであっても，その素性に大きな差が生じてしまうことが挙げられる．これは，異なる領域の文章はその話題や内容に差があり，共起語が大きく異なることによる影響であると考えられる．そこで本論文では，深層学習によって当該問題の解決を試みる．深層学習はニューラルネットワーク (Neural Network，NN) を用いて，データの抽象的な表現を獲得する手法である．近年活発に研究されており，画像認識や音声認識の分野でよい結果が得られている [4]．ターゲット領域に頻出しない共起語などから獲得したデータで分類器を学習する場合も，深層学習によって獲得した抽象的なデータによって学習することで，上述した問題を緩和できると考えた．本論文では複数存在する深層学習モデルのうち，SdA を利用した．

実験では，現代日本語書き言葉均衡コーパス (BCCWJ) の Yahoo!知恵袋 (OC) 及び書籍 (PB) のデータのうち 16 単語を対象とし，領域適応は OC → PB と PB → OC の双方向を行った．はじめに，特異値分解を利用した次元削減法 (Latent Semantic Indexing, LSI) と SdA により獲得した素性とでサポートベクターマシン (Support Vector Machine, SVM) の識別精度を比較した．実験の結果，SdA を利用したデータは，各領域適応で LSI で獲得したデータよりも良い精度が得られた．次に，SdA で獲得するデータの次元数に関する実験も行った．その結果，抽出する素性の次元数により，識別精度に大きな変化が見られた [5].

1.2 本論文の構成

本論文では，はじめに理論とその手法について紹介する．第 2 章で語義曖昧性解消 (Word Sense Disambiguation，WSD) について，第 3 章で領域適応 (Domain Adaptation) の概要を述べる．第 4 章では深層学習 (Deep Learning) について概説する．

そして，WSD の領域適応に対する深層学習の利用について述べ，その実験と結果の考察を行う．第 5 章で WSD の領域適応に対する SdA の利用について概説し，第 6 章では SVM を用いた実験の内容とその結果を述べる．第 7 章ではその実験結果について考察し，第 8 章で結論と今後の課題について述べる．

第2章　語義曖昧性解消

2.1　概要

単語には，一般に複数の意味が存在する．語義曖昧性解消 (Word Sense Disambiguation, WSD) は，文中に出現するこのような単語の語義を一意に識別するタスクである．

例えば,「のむ」という単語には少なくとも表 2.1 のような 3 つの意味が存在する．

表 2.1:「のむ」の語義

単語	語義
のむ	(1) 飲食物を口から体内に送り込む． (2) 受け入れる．妥協する． (3) 外に出さないで抑える．こらえる．

ここで,「のむ」に関する次のような例文が与えられたとする．

- 息をのむ．

この例文における「のむ」の語義は，表 2.1 のうち 3 番目の「外に出さないで抑える．こらえる.」に該当する．このようにして，与えられた文中に出現する単語の語義を一意に識別することを語義曖昧性解消と呼ぶ．

2.2　一般的な手法

一般に，WSD の識別には教師有り学習，半教師有り学習，教師無し学習などが利用される．教師有り学習はラベル付データを用いて学習を行う手法である．一方，教師無し学習はラベル無しデータを用いて学習を行う．また，半教師有り学習はラベル付きデータとラベル無しデータの混在データを利用して学習する．代表的な例として，サポートベクターマシン (Support Vector Machine，SVM) は，訓練データを用いて教師有り学習により分類器を学習し，テストデータの分類を行う．

学習にはコーパスと呼ばれる大量に集めた文書データを利用する．WSD のタスクの場合，このコーパスの一部のデータに語義や構文構造，品詞などの付加情報を含む自然言語処理に特化した注釈付コーパスが存在する．一般にはこのようなコーパスのデータを利用して学習を行う．

第3章　領域適応

従来，自然言語処理のタスクにおいて機械学習を用いる際，前提として訓練データとテストデータは同じ領域のコーパスから得られたデータである必要があった．しかし，WSD のタスクにおいて，訓練データとテストデータは異なる領域から得られていることも多い．例えば，訓練データとして書籍から得た文章を利用して分類器を学習し，新聞から得た文章中の単語の語義を識別したいような場合である．このような場合，書籍から得たデータで学習した分類器では，新聞から得たデータを精度良く識別することは困難である．

そこで，ある領域 S の訓練データによって学習した分類器を別の領域 T のデータに合うようにチューニングすることを領域適応と呼ぶ．ここで，訓練データを獲得した領域 S をソース領域，テストデータを獲得した領域 T をターゲット領域域と呼ぶ．例として，新聞から得た訓練データを雑誌から得たテストデータに適用する流れを図 3.1 に示す．

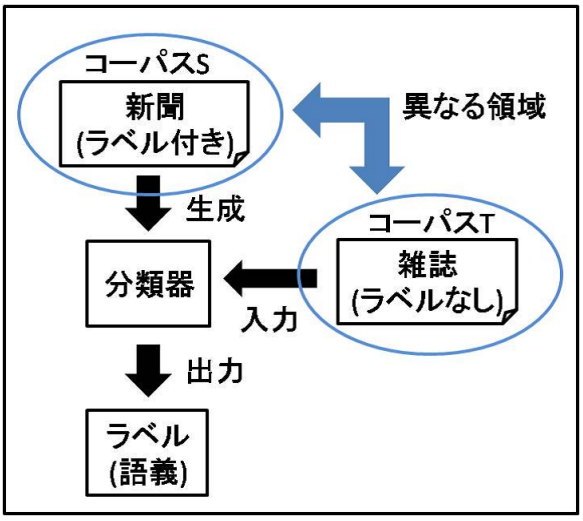

図 3.1: DomainAdaptation

第4章　深層学習

深層学習 (Deep Learning) は，ニューラルネットワーク (Neural network,NN) を用いた教師なし学習法で，NN を多層に重ねることにより抽象的なデータ表現を獲得する手法である．近年活発に研究されており，画像認識や音声認識の分野で有意な結果が得られている [4]．深層学習のモデルには確率論的モデルである RBM(Restricted Boltzmann Machines) や決定論的モデルの SdA(Stacked Denoising Autoencoder)，RBM の積み重ねからなる DBN(Deep Belief Nets) などが存在する．本論文では，これらのモデルのうち SdA を利用し，WSD の領域適応の問題解決を図る．SdA は dA(Denoising Autoencoder) と呼ばれる学習モデルを複数回実施することで学習を積み重ねるモデルである．本章では SdA 及びその基盤となる Autoencoder，Denoising Autoencoder について概説する．

4.1　Autoencoder

Autoencoder(AE) は，図 4.1 のような 3 層構造からなり，出力が入力を再現するようなモデルの学習を行う．

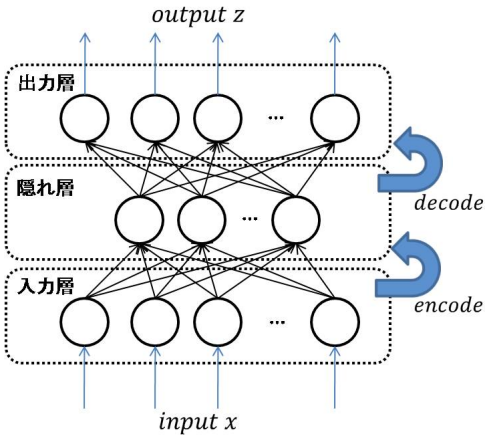

図 4.1: Autoencoder

まずはじめに，入力層 $\boldsymbol{x}$ に与えたデータに対して符号化 (encode) を行い，隠れ層 $\boldsymbol{y}$ が得られる．次に，隠れ層 $\boldsymbol{y}$ を複合化し，出力層 $\boldsymbol{z}$ を得る．AE は出力層 $\boldsymbol{z}$ と入力層 $\boldsymbol{x}$ の差が小さくなるようなモデルの学習を行う．このようにして得られた隠れ層 $\boldsymbol{y}$ は，より少ないノード数で入力層 (≒出力層) と同等の情報を表現していると言える．

4.2 Denoising Autoencoder

Denoising Autoencoder(dA) は図 4.2 のように AE の入力データに対して確率的にノイズを付加し，AE と同様の処理を行う．

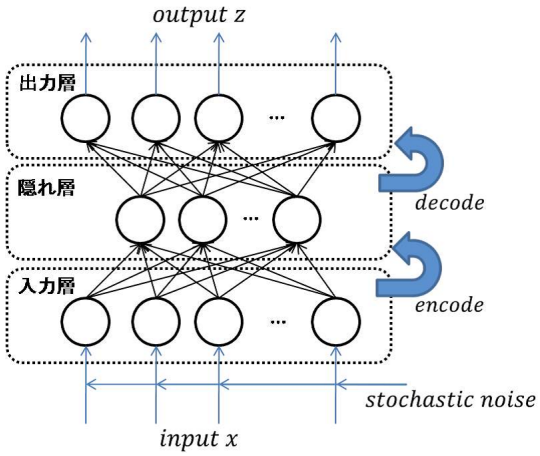

図 4.2: Denoising Autoencoder

AE の入力ベクトル x 対して確率的にノイズを付与したベクトル $\boldsymbol{x'}$ を入力層に与え，ノイズを付与する前の入力ベクトル $\boldsymbol{x}$ と出力層の差が小さくなるようなモデルを学習する．従って dA では，付与したノイズを除去するようなモデルを学習する．入力層 $\boldsymbol{x'}$ から隠れ層 $\boldsymbol{y}$，隠れ層 $\boldsymbol{y}$ から出力層 $\boldsymbol{z}$ の写像はそれぞれ以下の様に表される．

$$\begin{aligned} \boldsymbol{y} &= sigmoid(\boldsymbol{W}\boldsymbol{x'} + \boldsymbol{b}) \\ \boldsymbol{z} &= sigmoid(\boldsymbol{W^T}\boldsymbol{y} + \boldsymbol{b'}) \end{aligned}$$

ここで，$\boldsymbol{b}$，$\boldsymbol{b'}$ はバイアス，$\boldsymbol{W}$ は重み行列を示し，$\boldsymbol{W^T}$ はその転置行列である．また，$sigmoid()$ はシグモイド関数を表す．確率的勾配降下法 (Stochastic Gradient Descent, SGD) により，次の式で表される平均二乗誤差関数が最小となる $\boldsymbol{W}(\boldsymbol{W^T})$，$\boldsymbol{b}$，$\boldsymbol{b'}$ を求める．

$$E_N = \|\boldsymbol{z} - \boldsymbol{x}\|^2$$

このようにして得られた隠れ層 $\boldsymbol{y}$ は入力データ $\boldsymbol{x}$ の抽象的表現となる．

4.3 パラメータの更新式

dA では平均二乗誤差が最小となるパラメータ $\boldsymbol{W}(\boldsymbol{W^T})$，$\boldsymbol{b}$，$\boldsymbol{b'}$ を求める．ここでは，確率的勾配降下法による各パラメータの更新式を導出する．

入力層 $\boldsymbol{x'}$ から隠れ層 $\boldsymbol{y}$，隠れ層 $\boldsymbol{y}$ から出力層 $\boldsymbol{z}$ の写像はそれぞれ次の様に表された．

$$
\begin{aligned}
\boldsymbol{y} &= sigmoid(\boldsymbol{W}\boldsymbol{x}' + \boldsymbol{b}) \\
\boldsymbol{z} &= sigmoid(\boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{y} + \boldsymbol{b}')
\end{aligned}
$$

また，二乗誤差は次の式で表される．

$$
E(x, z) = (\boldsymbol{z} - \boldsymbol{x})^2
$$

ここで,

$$
\begin{aligned}
h_1(\boldsymbol{x}') &= \boldsymbol{W}\boldsymbol{x}' + \boldsymbol{b} \\
h_2(\boldsymbol{y}) &= \boldsymbol{W}^{\boldsymbol{T}}\boldsymbol{y} + \boldsymbol{b}'
\end{aligned}
$$

と置けば,

$$
\begin{aligned}
\boldsymbol{y} &= sigmoid(h_1) \\
\boldsymbol{z} &= sigmoid(h_2)
\end{aligned}
$$

と表せる．E の各パラメータにおける勾配は

$$
\begin{aligned}
\frac{\partial E}{\partial \boldsymbol{W}} &= \frac{\partial E}{\partial h_1}\frac{\partial h_1}{\partial \boldsymbol{W}} + \frac{\partial E}{\partial h_2}\frac{\partial h_2}{\partial \boldsymbol{W}} \\
\frac{\partial E}{\partial \boldsymbol{b}} &= \frac{\partial E}{\partial h_1}\frac{\partial h_1}{\partial \boldsymbol{b}} + \frac{\partial E}{\partial h_2}\frac{\partial h_2}{\partial \boldsymbol{b}} \\
&= \frac{\partial E}{\partial h_1}\frac{\partial h_1}{\partial \boldsymbol{b}} \\
&= \frac{\partial E}{\partial h_1} \\
\frac{\partial E}{\partial \boldsymbol{b}'} &= \frac{\partial E}{\partial h_1}\frac{\partial h_1}{\partial \boldsymbol{b}'} + \frac{\partial E}{\partial h_2}\frac{\partial h_2}{\partial \boldsymbol{b}'} \\
&= \frac{\partial E}{\partial h_2}\frac{\partial h_2}{\partial \boldsymbol{b}'} \\
&= \frac{\partial E}{\partial h_2}
\end{aligned}
$$

である．シグモイド関数の微分公式を用いれば,

$$
\begin{aligned}
\frac{\partial E}{\partial h_1} &= \frac{\partial E}{\partial \boldsymbol{y}}\frac{\partial \boldsymbol{y}}{\partial h_1} \\
&= \frac{\partial E}{\partial \boldsymbol{y}} * \boldsymbol{y} * (1 - \boldsymbol{y}) \\
\frac{\partial E}{\partial h_2} &= \frac{\partial E}{\partial \boldsymbol{z}}\frac{\partial \boldsymbol{z}}{\partial h_2} \\
&= 2(\boldsymbol{z} - \boldsymbol{x}) * \boldsymbol{z} * (1 - \boldsymbol{z})
\end{aligned}
$$

となる．従って，各パラメータにおける勾配は

$$\frac{\partial E}{\partial \boldsymbol{W}} = (\boldsymbol{W}(2(\boldsymbol{z}-\boldsymbol{x}) * \boldsymbol{z} * (1-\boldsymbol{z})) * \boldsymbol{y} * (1-\boldsymbol{y}))\boldsymbol{x'}^{T} + ((2(\boldsymbol{z}-\boldsymbol{x}) * \boldsymbol{z} * (1-\boldsymbol{z})) * \boldsymbol{y}^{T})^{T}$$

$$\frac{\partial E}{\partial \boldsymbol{b}} = \boldsymbol{W}(2(\boldsymbol{z}-\boldsymbol{x}) * \boldsymbol{z} * (1-\boldsymbol{z})) * \boldsymbol{y} * (1-\boldsymbol{y})$$

$$\frac{\partial E}{\partial \boldsymbol{b'}} = 2(\boldsymbol{z}-\boldsymbol{x}) * \boldsymbol{z} * (1-\boldsymbol{z})$$

であり，各パラメータの更新式は以下のようになる．

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_{t} - \frac{\epsilon}{N}\sum_{n=1}^{N}\frac{\partial E}{\partial \boldsymbol{W}_{t}}$$

$$\boldsymbol{b}_{t+1} = \boldsymbol{b}_{t} - \frac{\epsilon}{N}\sum_{n=1}^{N}\frac{\partial E}{\partial \boldsymbol{b}_{t}}$$

$$\boldsymbol{b'}_{t+1} = \boldsymbol{b'}^{t} - \frac{\epsilon}{N}\sum_{n=1}^{N}\frac{\partial E}{\partial \boldsymbol{b'}_{t}}$$

4.4 Stacked Denoising Autoencoder

Stacked Denoising Autoencoder(SdA) は，dA を複数回積み重ねたものである．まず，dA によって学習を行う．次に，得られた隠れ層を入力として再度 dA によって学習する．これを繰り返し行うことで，dA による学習を積み重ね，生のデータからデータの抽象的な表現の素性を得る．従って，dA によって得られた出力層は誤差関数を求めるためだけに使用し，SdA ではこの過程で得られた隠れ層を利用する．

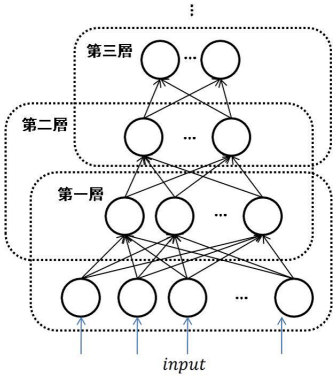

図 4.3: Stacked Denoising Autoencoder

第5章　SdAを利用した語義曖昧性解消の領域適応

WSDにおいて教師有り学習などで識別を行う場合，注釈付コーパスから素性と呼ばれる特徴量を抽出し，これを用いて学習を行う．素性は対象単語と共起する単語の有無やその単語の品詞などを数値化するのが一般的である．WSDのタスクにおいて，訓練データとテストデータが異なる領域から得られているようなケースが存在し，このような場合，識別精度の低下が危惧されることは第3章で既に述べた．これは，異なる領域から得た文章はその話題や内容に差があり，共起語が大きく異なるため，その素性に大きな差が生じることによる影響と考えられる．例として第2章で挙げた「のむ」という単語について考える．この単語が新聞から得たデータに出現した場合，次のような文が考えられる．

- 条件をのんだ.
- 契約をのんだ.

一方，この単語を得たデータがブログや書籍などであった場合，次のような文が頻出する．

- 水をのむ.
- 薬をのませた.
- 息をのんだ.

新聞から得たデータにおける素性は「条件」や「契約」といった単語の特徴がよく表れている．一方，ブログや書籍などから得たデータの素性は「水」や「薬」,「息」といった単語の特徴が反映される．このようにして，例えば新聞から得た素性で分類器を学習し，ブログや書籍などから得た素性を識別しようとした場合，分類器には「水」や「薬」,「息」といった単語に関する情報がほとんどなく，識別が困難になってしまう．

本論文では，深層学習を利用しこの問題解決を図る．第4章で述べたように，深層学習は与えられたデータから抽象的な表現を獲得する機械学習手法である．深層学習によって，データの抽象的表現をうまく獲得することができれば,「水」や「契約」といった具体的な単語に関する素性ではなく,「のむ」という単語の全体像を表現した素性となり，上述した問題を緩和させることができる．

本論文では，SdAを用いて獲得したデータの抽象的な表現を入力データに付加することで，当該問題の解決を図る．図5.1にSdAを利用したWSDの領域適応の流れを示す.

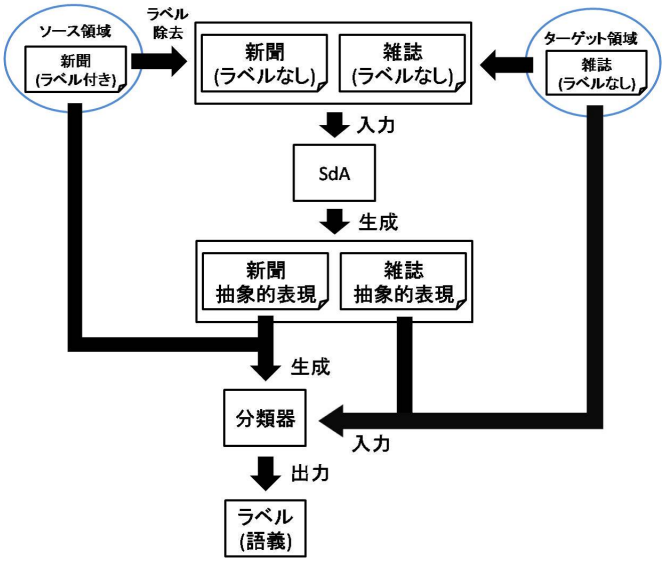

図 5.1: SdA を利用した WSD の領域適応

具体的には，SdA を利用して n_x 次元の入力データ $\boldsymbol{x}$ から n_{abst} 次元の抽象的表現 $\boldsymbol{x_{abst}}$ を抽出する．そして，入力データ $\boldsymbol{x}$ 及び抽象的表現 $\boldsymbol{x_{abst}}$ を結合した $n_x + n_{abst}$ 次元のデータを得る．これを Support Vector Machines(SVM) によって識別する．SdA による抽象的表現 $\boldsymbol{x_{abst}}$ 獲得及び入力データ $\boldsymbol{x}$ との結合の流れを図 5.2 に示す.

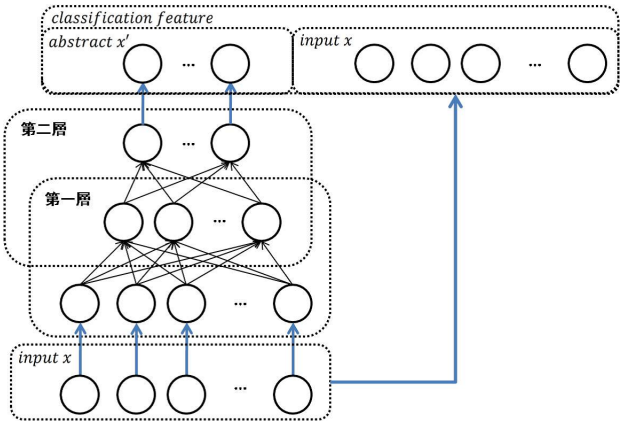

図 5.2: 識別素性獲得の流れ

第6章　実験

6.1　実験概要

WSD の領域適応のタスクに SdA のモデルを用いた深層学習を利用することの有効性を検証する．SdA によるデータの抽象的な表現の獲得は入力データの素性に対して少ない次元数の素性を獲得するという観点から見れば，一種の次元縮約とみなすことができる．そこで，本実験では代表的な次元縮約法のひとつである特異値分解を利用した次元削減法 (Late Semamtic Indexing，LSI) と比較する．

識別するデータは現代日本語書き言葉均衡コーパス (BCCWJ)[6] における Yahoo!知恵袋 (OC) 及び書籍 (PB) のデータを利用する．SemEval-2 の日本語 WSD タスクではこれらの領域のコーパスの一部に語義タグを付けたデータを公開しており，そのデータを利用する．このデータのうち，それぞれのデータである程度の頻度で存在する多義語 16 単語を対象とする．領域適応は OC 及び PB を各領域とし，双方向 (OC → PB 及び PB → OC) を行った．

6.1.1　SdA

SdA の学習には，Deep Learning ライブラリとして公開されている Pylearn2[1]を利用する．SdA による dA の積み重ね回数は 2 回とした．ここで，始めに学習を行う dA を dA_1，二番目に学習を行う dA を dA_2 とする．入力データの次元数を N_x 次元とした場合の dA_1，dA_2 における入力層，隠れ層，出力層のノード数はそれぞれ表 6.1，表 6.2 のようである．ただし，計算結果が少数を含む場合は，四捨五入による端数処理を行う．

表 6.1: dA_1 の各層のノード数

層	ノード数
入力層	N_x
隠れ層	$2/3 * N_x$
出力層	N_x

[1] http://deeplearning.net/software/pylearn2/

表 6.2: dA_2 の各層のノード数

層	ノード数
入力層	$2/3 * N_x$
隠れ層	$2/3 * N_x$
出力層	$2/3 * N_x$

従って，SdA の各層のノード数は表 6.3 のようであり，例えば入力データの次元数を 5 とした場合の SdA は図 6.1 のようである．

表 6.3: SdA の各層のノード数

層	ノード数
入力層	N_x
第一層	$2/3 * N_x$
第二層	$2/3 * N_x$

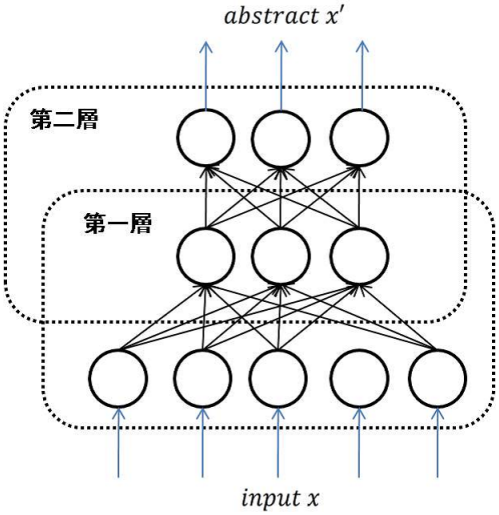

図 6.1: SdA($inputDim = 5$)

6.1.2 LSI

Latent Semantic Indexing(LSI) は特異値分解 (singular value decomposition，SVD) を利用した次元削減法である．特異値分解は m 行 n 列の行列 A を次のように分解する．

$$A = U \Sigma V$$

ここで，U は m 行 m 列，V は n 行 n 列の行列であり，Σ は対角成分に特異値を持つ対角行列である．LSI ではここで得られた行列 V から任意の列数 k だけ列ベクトルを

削除する．このようにして得られた n 行 $n-k$ 列の行列 V_k と行列 A の積 $A_k = A \times V_k$ は行列 A の情報をできるだけ保存しつつ次元数を圧縮した行列となる．

WSD のタスクにおいて，m 行 n 列の行列 A はデータ数 m，次元数 n の入力データであり，LSI によって獲得される行列 A_k はデータ数 m，次元数 $n-k$ のデータとなる．

本実験では，比較のため LSI による次元縮約と SdA による特徴抽出の次元数が同等となるよう，行列 V から入力データの次元数に対して 1/3 次元分の列ベクトルを削除し，2/3 次元のデータを得る．

6.1.3 SVM によるデータの識別

それぞれの方法で獲得したデータをサポートベクターマシン (Support Vector Machine, SVM) によって識別する．SVM による識別を行うデータは SdA，LSI のそれぞれの方法によって獲得した素性 (縮約素性) と入力データの素性 (入力素性) とを結合したものを利用する．ただし，そのまま素性を結合した場合，縮約素性は入力素性と比べて次元数が少ないことは明らかであり，入力素性の情報がより強く結果に表れてしまう．そこで，入力素性と縮約素性の重みを等しくするため，それぞれの素性を正規化したものを結合する．SVM による識別には libsvm[2]を用いる．

6.2 実験結果

それぞれの方法によって獲得した素性の SVM による識別結果を表 6.4(OC → PB) 及び表 6.5(PB → OC) に示す．ここで，Normal は入力データの素性に対して何も処理をせずに SVM によって識別した結果を表す．ただし，Total の正解率はマイクロ平均により算出する．

LSI 及び SdA は Normal と比べて識別精度が改善されている．また，LSI と SdA を比較すると，単語により精度の良し悪しが異なるが，平均では各領域適応で SdA の識別精度が良い結果となった．

[2] http://www.csie.ntu.edu.tw/~cjlin/libsvm/

表 6.4: 実験結果 (OC → PB(%))

Word	Normal	LSI	SdA
言う	82.05	81.78	82.14
入れる	62.5	66.07	66.07
書く	83.87	77.42	75.81
聞く	66.67	68.29	67.48
子供	24.73	30.11	27.96
時間	90.54	87.84	87.84
自分	97.40	94.81	95.13
出る	55.92	57.89	58.55
取る	23.46	25.93	24.69
場合	84.67	84.67	84.67
入る	43.22	45.76	47.46
前	69.38	77.50	77.50
見る	83.88	83.88	83.88
持つ	77.12	79.09	77.78
やる	92.95	92.95	92.95
行く	88.72	88.72	88.72
Total	77.20	77.70	77.73

表 6.5: 実験結果 (PB → OC(%))

Word	Normal	LSI	SdA
言う	83.78	83.48	83.63
入れる	73.97	71.23	76.71
書く	73.74	73.74	73.74
聞く	66.13	65.32	70.16
子供	12.99	25.97	23.38
時間	83.02	83.02	83.02
自分	87.50	87.50	87.50
出る	64.89	66.41	66.41
取る	24.59	31.15	32.79
場合	91.27	90.48	90.48
入る	63.24	58.82	58.82
前	89.52	90.48	89.52
見る	56.11	56.49	56.49
持つ	64.52	82.26	82.26
やる	94.02	94.02	94.02
行く	68.95	68.95	68.95
Total	73.09	73.94	74.31

第7章　考察

本論文では，WSD の領域適応のタスクの主問題がソース領域とターゲット領域のデータにおける素性の相違にあると考え，深層学習のモデルのうち SdA を利用し，識別精度の改善を試みた．

SdA の核であるモデル dA は入力層や出力層と比べて隠れ層が少ないノード数で同等の情報を表現できているという観点から，隠れ層は入力層を抽象的に表現していると言える．従って，これをうまくデータに付加できれば，ソース-ターゲット間の素性の相違を緩和できると考えた．

実験ではそのままの入力データや LSI によって獲得したデータと比べて平均で良い結果が得られていることから，意図したように SdA によってデータの抽象的表現を獲得し，うまく当該問題を緩和できたと言える．しかしながら，識別精度が悪化した単語も存在する．以下ではこのような事例の改善や更なる識別精度改善のための主要因について考察する．

7.1　各層のノード数

SdA による学習の精度は，各 dA における隠れ層のノード数に大きく依存すると考えられる．そこで，隠れ層のノード数による識別精度の比較を行う．

前述の実験において，SdA は入力データの次元数に対して 2/3 次元の素性を得た．隠れ層のノード数の変化による識別精度の比較を行うため，これを 1/3 次元に変更し，再実験を行った．すなわち，本実験では，n_x 次元の入力データ x に対して dA_1，dA_2 の隠れ層のノード数を $n_x/3$ 個とした．実験の結果を表 7.1 及び表 7.2 に示す．

表 7.1: 各ノード数による識別精度 (OC → PB(%))

Word	2/3 次元	1/3 次元
言う	82.14	81.51
入れる	66.07	64.29
書く	75.81	74.19
聞く	67.48	63.41
子供	27.96	26.88
時間	87.84	86.49
自分	95.13	94.16
出る	58.55	59.21
取る	24.69	23.46
場合	84.67	84.67
入る	47.46	47.46
前	77.50	76.88
見る	83.88	83.88
持つ	77.78	75.82
やる	92.95	92.95
行く	88.72	89.47
Total	77.73	77.04

表 7.2: 各ノード数による識別精度 (PB → OC(%))

Word	2/3 次元	1/3 次元
言う	83.63	82.88
入れる	76.71	72.60
書く	73.74	73.74
聞く	70.16	66.94
子供	23.38	40.26
時間	83.02	83.02
自分	87.50	87.50
出る	66.41	62.60
取る	32.79	26.23
場合	90.48	91.27
入る	58.82	54.41
前	89.52	87.62
見る	56.49	56.87
持つ	82.26	74.19
やる	94.02	94.02
行く	68.95	70.78
Total	74.31	73.81

SdA によって入力データに対して 2/3 次元の素性を獲得する場合と比べ，一部の単語で僅かに精度が改善されたケースも存在するが，大半の単語において精度が低下或いは変化が見られず，大幅に悪化してしまうケースも存在した．また，OC → PB の領域

適応において，入力データに対して 1/3 次元の素性を得る SdA では，そのままのデータを SVM によって識別する場合よりも精度が悪くなった．本実験によって SdA の学習では各層のノード数を適切に設定できなければ識別精度を改善できないことが分かった．このことから，各層のノード数をうまく設定できれば更なる精度改善が見込めると考える．

7.2 積み重ね回数

各層のノード数と同様に，SdA における dA の積み重ね回数も重要である．SdA では dA を複数回積み重ね学習を繰り返すことでデータの抽象的な表現を獲得する．本論文において，積み重ね回数はすべて 2 回に設定して実験を行った．これは，積み重ね回数としては最も少ない回数である．dA の積み重ね回数を増やせば，SdA はより抽象的な表現を獲得できると言える．しかしながら，必要以上に積み重ね回数を増やした場合，獲得する素性は抽象的すぎる表現となってしまい，どのような入力データから獲得した素性でも同様な素性となり，識別結果になんら影響を与えない余計な素性となってしまうと考えられる．今後，dA の積み重ね回数に関する実験を行い適切な積み重ね回数を設定する必要がある．

7.3 素性の重み

本論文では，入力素性と縮約素性が識別結果に与える影響を同等にするために，それぞれの素性を正規化したものを結合した．入力素性の識別に対する役割はソース領域及びターゲット領域で頻出する共起語に関する具体的な情報による細分類である．一方，縮約素性の識別に対する役割はターゲット領域の共起語がソース領域で頻出しない場合における抽象的表現によるデータの分類である．従って，ソース領域とターゲット領域の共起語が似通っていれば入力素性の重みを大きく，そうでなければ縮約素性の重みを大きくすることで識別精度の改善が見込めると考える．2 領域の共起語の類似性は与えられたデータから判別可能である．そこで今後，何らかの方法でこれを数値化し，入力素性と縮約素性の重みを適宜変更する実験を行う必要がある．

第8章　結論

本論文ではWSDの領域適応のタスクにおいてSdAを用いてデータの抽象的表現を抽出し，素性に加えることで識別精度の改善を試みた．領域適応の主問題として，同じ単語に関するデータであってもソース-ターゲット間には素性に大きな差があり，識別精度が低下してしまうことが挙げられる．SdAによってデータの抽象的な表現を獲得することができれば，この素性の相違をうまく緩和させることができると考えた．

実験では，なにも施さないそのままのデータやLSIによって獲得したデータと比べて，SdAによって獲得したデータに対するSVMの識別精度の方が良い結果が得られた．これは，SdAによってデータの抽象的な表現をうまく獲得でき，これを素性に加えることで，意図したようにソース領域とターゲット領域の素性の相違を緩和できたと考える．

次に，dAの隠れ層のノード数を変更し，精度の比較を行った．結果として，ノード数をより少なくした場合は，そうでない場合よりも精度が悪化した．更には，領域によっては，SdAを実施せず，そのままのデータをSVMによって識別した方が良い結果となった．dAにおいて，隠れ層のノード数を少なく設定すると，各ノードはより抽象的な情報を保持していると言える．しかし，極端に少ないノード数の場合，正確な情報を表現しきれなくなってしまい，識別精度が低下してしまうと考えられる．今後，このdAの隠れ層のノード数に関する追実験を行い，適切なノード数を考察する予定である．

dAの積み重ね回数に関しても実験が必要である．本論文では，dAの積み重ね回数を2回としたが，これを増やし，より抽象的な表現を獲得した場合，識別精度がどのように変化するか考察を行う必要がある．また，SdAによって獲得した素性の重みに関する実験も必要である．本論文では，SdAによって獲得した素性と入力データの素性とを1:1の割合で結合した．これを単に変更，或いは訓練データとテストデータの類似性によって適宜変更した場合，識別精度にどのような変化が見られるか実験を行い，適切な比率を考察する必要がある．

謝辞

本研究を進めるにあたり，多くのご指導，ご協力を頂いた指導教員の新納浩幸准教授に感謝致します．また，日常の議論を通じて多くの知識や示唆を頂いた新納研究室の皆様に感謝します.

参考文献

[1] Anders Sogaard. *Semi-Supervised Learning and Domain Adaptation in Natural Language Processing*. Morgan & Claypool, 2013.

[2] 新納浩幸, 佐々木稔. ”共変量シフト下の学習による語義曖昧性解消の教師なし領域適応”. 自然言語処理, Vol. 21, No. 5, pp. 1011–1035, 2014.

[3] 新納浩幸, 佐々木稔. ”k 近傍法とトピックモデルを利用した語義曖昧性解消の領域適応”. 自然言語処理, Vol. 20, No. 5, pp. 707–726, 2013.

[4] Quoc Le, Marc’Aurelio Ranzato, Rajat Monga, Matthieu Devin, Kai Chen, Greg Corrado, Jeff Dean, and Andrew Ng. ”Building high-level features using large scale unsupervised learning”. In *ICML-2012*, 2012.

[5] 河野和平, 新納浩幸, 佐々木稔, 古宮嘉那子. ”Stacked Denoising Autoencoder を利用した語義曖昧性解消の領域適応”. 言語処理学会第 21 回年次大会, to appear, (2015).

[6] Kikuo Maekawa. ”Design of a Balanced Corpus of Contemporary Written Japanese”. In *LKR-2007*, pp. 55–58, 2007.

付 録A　Pylearn2

本論文では SdA の学習に Deep Learning ライブラリの Pylearn2 を利用した．以下ではスクリプト言語 Python の習得と Pylearn2 のインストール及びチュートリアルが動作済みであることを前提に Pylearn2 における SdA の利用方法について紹介する．なお，Pylearn2 のインストールなどについては詳細が既に多数のウェブサイトなどで解説されているため，そちらを参照されたい．

Pylearn2 で SdA を利用するために編集が必要となるのは，以下の 3 つのファイルである．

表 A.1: 編集が必要なファイル

ファイル	Path
test_dae.py	/pylearn2/scripts/tutorials/stacked_autoencoders/tests/
mnist.py	/pylearn2/datasets/
dae_l1.yaml	/pylearn2/scripts/tutorials/stacked_autoencoders/

以下では各ファイルの要点についてまとめる

test_dae.py

SdA による学習の一連の処理を行う実行ファイルである．初期状態では手書き数字認識用のデータ (mninst) に対して dA による学習を 2 回繰り返し，その後多層パーセプトロン (Multi Layer Perceptron，MLP) によって学習し，識別を行うような処理が施されている．ただし，MLP に関する記述は本論文と関係がないため，記述を省く．図 A.1 に修正が必要な箇所を，図 A.2 に省略が必要な箇所を示す．①は dA_1 の入力層のノード数であり，②はそれぞれ dA_2 の入力層と隠れ層のノード数である．

```
"""
This module tests stacked_autoencoders.ipynb
"""

import os

from pylearn2.testing import skip
from pylearn2.testing import no_debug_mode
from pylearn2.config import yaml_parse


@no_debug_mode
def train_yaml(yaml_file):

    train = yaml_parse.load(yaml_file)
    train.main_loop()


def train_layer1(yaml_file_path, save_path):
    print "-----------------------------------l1"
    yaml = open("{0}/dae_l1.yaml".format(yaml_file_path), 'r').read()
    hyper_params = {'train_stop': 50,
                    'batch_size': 50,
                    'monitoring_batches': 1,
                    'nhid': 484,
                    'max_epochs': 1,
                    'save_path': save_path}
    yaml = yaml % (hyper_params)
    train_yaml(yaml)

def train_layer2(yaml_file_path, save_path):
    print "-----------------------------------l2"
    yaml = open("{0}/dae_l2.yaml".format(yaml_file_path), 'r').read()
    hyper_params = {'train_stop': 50,
                    'batch_size': 50,
                    'monitoring_batches': 1,
                    'nvis': 484,
                    'nhid': 484,
                    'max_epochs': 1,
                    'save_path': save_path}
    yaml = yaml % (hyper_params)
    train_yaml(yaml)
```

① ②

図 A.1: test_dae.py(1)

```
def train_mlp(yaml_file_path, save_path):
    print "-----------------------------------mlp"
    yaml = open("{0}/dae_mlp.yaml".format(yaml_file_path), 'r').read()
    hyper_params = {'train_stop': 50,
                    'valid_stop': 50050,
                    'batch_size': 10,
                    'max_epochs': 1,
                    'save_path': save_path}
    yaml = yaml % (hyper_params)
    train_yaml(yaml)                                                    ③

def test_sda():

    skip.skip_if_no_data()

    yaml_file_path = os.path.abspath(os.path.join(os.path.dirname(__file__),
                                                  '..'))
    save_path = os.path.dirname(os.path.realpath(__file__))

    train_layer1(yaml_file_path, save_path)
    train_layer2(yaml_file_path, save_path)
    train_mlp(yaml_file_path, save_path)                                ④

    try:
        os.remove("{}/dae_l1.pkl".format(save_path))
        os.remove("{}/dae_l2.pkl".format(save_path))
    except OSError:
        pass                                                            ⑤

if __name__ == '__main__':
    test_sda()
```

図 A.2: test_dae.py(2)

mnist.py

入力データを読み込むファイルである．WSD のタスクにおいて通常は入力データは libsvm 形式で与えられる．しかし，初期状態では mnist のファイルを読み込むため，csv 形式で指定されている．そこで，WSD の領域適応のタスクに適用するためには，libsvm 形式に書き換える必要がある．

mnist.py では csv 形式のファイルを読み込み，特徴ベクトルを配列 X にラベルを配列 y に格納している．そこで，当該記述を libsvm 形式のファイルを読み込み，配列 X，y に格納するよう書き換えればよい．minist.py を libsvm 形式のファイルを読み込むよう書き換えたプログラムは付録 B を参照のこと．

dae_l1.yaml

yaml ファイルは，学習に必要なパラメータや学習法などを記述するファイルである．各学習で得られた重み行列 $\boldsymbol{W}$，バイアス $\boldsymbol{b}$，$\boldsymbol{b}'$ はここで指定した pkl ファイルに記述される．

初期状態では，dae_l1.yaml，dae_l2.yaml，dae_mlp.yaml の 3 つが存在し，それぞれ，dA_1，dA_2，MLP の学習に関するファイルである．ただし，MLP に関しては実施しないため，該当ファイルも不要である．また，dA_2 の入力データは dA_1 の隠れ層となり，既にそのように記述されているため編集は不要である．そこでここでは，dae_l1.yaml に

ついて述べるが，基本的には dae_l2.yaml も同様である．

図 A.3 に dae_l1.yaml ファイルを示す．当該ファイルには dataset，model，algorithm の主に 3 つの項目が存在する．

dataset では入力データの指定を行う．

model では学習モデル (本論文では SdA) の選択を行う．model 中の nvis，nhid はそれぞれ入力層，隠れ層のノード数である．また，corruptor ではノイズの選択を行っている．主に修正が必要な箇所は nvis である．

algorithm では学習アルゴリズムの選択を行う．本論文では SGD による平均二乗誤差の最小化を行った．

```
!obj:pylearn2.train.Train {
    dataset: &train !obj:pylearn2.datasets.mnist.MNIST {
        which_set: 'train',
        # TODO: the one_hot: 1 is only necessary because one_hot: 0 is
        # broken, remove it after one_hot: 0 is fixed.
        one_hot: 1,
        start: 0,
        stop: %(train_stop)i                                              ①
    },
    model: !obj:pylearn2.models.autoencoder.DenoisingAutoencoder {
        nvis : 784,
        nhid : %(nhid)i,
        irange : 0.05,
        corruptor: !obj:pylearn2.corruption.BinomialCorruptor {
            corruption_level: .2,
        },
        act_enc: "tanh",
        act_dec: null,    # Linear activation on the decoder side.      ②
    },
    algorithm: !obj:pylearn2.training_algorithms.sgd.SGD {
        learning_rate : 1e-3,
        batch_size : %(batch_size)i,
        monitoring_batches : %(monitoring_batches)i,
        monitoring_dataset : *train,
        cost : !obj:pylearn2.costs.autoencoder.MeanSquaredReconstructionError {},
        termination_criterion : !obj:pylearn2.termination_criteria.EpochCounter {
            max_epochs: %(max_epochs)i,
        },                                                                ③
    },
    save_path: "%(save_path)s/dae_l1.pkl",
    save_freq: 1
}
```

図 A.3: dae_l1.yaml

付 録B　ソースリスト

```
"""
adult dataset wrapper for pylearn2
"""

import csv
import numpy as np
import os

from pylearn2.datasets.dense_design_matrix import DenseDesignMatrix
from pylearn2.utils import serial
from pylearn2.utils.string_utils import preprocess

class AdultDataset( DenseDesignMatrix ):

        def __init__(self,
                        path = 'train.csv',
                        one_hot = False,
                        with_labels = True,
                        start = None,
                        stop = None,
                        preprocessor = None,
                        fit_preprocessor = False,
                        fit_test_preprocessor = False):
                """
                which_set: A string specifying which portion of the dataset
                        to load. Valid values are 'train' or 'public_test'
                base_path: The directory containing the .csv files from kaggle.com.
                                This directory should be writable; if the .csv files haven't
                                already been converted to npy, this class will convert them
                                to save memory the next time they are loaded.
                fit_preprocessor: True if the preprocessor is allowed to fit the
                                   data.
                fit_test_preprocessor: If we construct a test set based on this
                                        dataset, should it be allowed to fit the test set?
                """
                self.no_classes = 2

                self.test_args = locals()
                self.test_args['which_set'] = 'test'
                self.test_args['fit_preprocessor'] = fit_test_preprocessor
                del self.test_args['start']
                del self.test_args['stop']
                del self.test_args['self']

                path = preprocess(path)
                X, y = self._load_data( path, with_labels )


                if start is not None:
                        assert which_set != 'test'
                        assert isinstance(start, int)
                        assert isinstance(stop, int)
                        assert start >= 0
                        assert start < stop
                        assert stop <= X.shape[0]
                        X = X[start:stop, :]
                        if y is not None:
                                y = y[start:stop, :]

```

```
61                 super(AdultDataset, self).__init__(X=X, y=y)
62
63                 if preprocessor:
64                         preprocessor.apply(self, can_fit=fit_preprocessor)
65
66         def _load_data(self, path, expect_labels):
67
68                 assert path.endswith('.dat')
69
70                 libsvmformat = open(path,'r')
71
72                 l = []
73
74                 for line in libsvmformat:
75                         s = line.split()
76                         m = []
77                         for mtos in s:
78                                 tmp = mtos.split(':')
79                                 m.append(tmp)
80
81                         l.append(m)
82
83                 numpy_label = np.empty(len(l),dtype=str)
84                 dim = 0
85
86                 for i in range(len(l)):
87                         if dim < int(l[i][-1][0]):
88                                 dim = int(l[i][-1][0])
89
90
91                 numpy_feature = np.zeros((len(l),dim))
92
93                 for i in range(len(l)):
94                         numpy_label[i] = str(l[i][0][0])
95
96                 for i in range(len(l)):
97                         for j in range(len(l[i])-1):
98                                 numpy_feature[i][int(l[i][j+1][0])-1] = l[i][j+1][1]
99
100                data = np.zeros((len(numpy_label),dim+1))
101
102                for i in range(len(numpy_label)):
103                        data[i][0] = int(0)
104                        for j in range(dim):
105                                data[i][j+1] = int(numpy_feature[i][j])
106
107                if expect_labels:
108                        y = numpy_label
109                        X = data[:,1:]
110
111                        one_hot = np.zeros((y.shape[0], self.no_classes ),dtype='float32')
112                        for i in xrange( y.shape[0] ):
113                                label = y[i]
114                                if label == 1:
115                                        one_hot[i,1] = 1.
116                                else:
117                                        one_hot[i,0] = 1.
118
119                        y = one_hot
120                else:
121                        X = data
122                        y = None
123
124                return X, y
```

ソースコード 1 adult_dataset.py(libsvm 形式を読み込む mninst.py)

```
#-*- coding: utf-8 -*-
# kouno

import cPickle as pickle
import sys
import numpy as np
import os

filename = 'train_v.dat'
pickle_file = 'dae_l1.pkl'
pickle_file2 = 'dae_l2.pkl'

fp = open(pickle_file)
par = pickle.load(fp)
pv = par.get_param_values()

fp2 = open(pickle_file2)
par2 = pickle.load(fp2)
pv2 = par2.get_param_values()

libsvmformat = open(filename,'r')

l = []

for line in libsvmformat:
        s = line.split()
        m = []
        for mtos in s:
                tmp = mtos.split(':')
                m.append(tmp)

        l.append(m)

numpy_label = np.empty(len(l),dtype=str)
dim = 0

for i in range(len(l)):
        if dim < int(l[i][-1][0]):
                dim = int(l[i][-1][0])

numpy_feature = np.zeros((len(l),dim))

for i in range(len(l)):
        #print "label = ",l[i][0][0]
        numpy_label[i] = str(l[i][0][0])

for i in range(len(l)):
        for j in range(len(l[i])-1):
                numpy_feature[i][int(l[i][j+1][0])-1] = l[i][j+1][1]

data = np.zeros((len(numpy_label),dim+1))

for i in range(len(numpy_label)):
        #data[i][0] = int(numpy_label[i])
        data[i][0] = int(0)
        for j in range(dim):
                data[i][j+1] = int(numpy_feature[i][j])

c = np.dot(numpy_feature,pv[2])

e = np.dot(c,pv2[2])

norm = numpy_feature
for i in norm:
        i /= np.linalg.norm(i)

norm2 = e
for i in norm2:
        i /= np.linalg.norm(i)

print np.shape(norm)
print np.shape(norm2)
```

```
73
74  x_new = np.c_[norm,norm2]
75  x_new_dl_only = norm2
76
77  label_oc = open('label-oc','r')
78  label_pb = open('label-pb','r')
79
80  class_oc = []
81  for line in label_oc:
82          oc = line.split()
83          class_oc.append(oc[-1])
84
85  class_pb = []
86  for line in label_pb:
87          pb = line.split()
88          class_pb.append(pb[-1])
89
90  f_oc = open('OCdl.dat','w')
91  f_pb = open('PBdl.dat','w')
92
93  f_oc_only = open('OCdl_only.dat','w')
94  f_pb_only = open('PBdl_only.dat','w')
95
96  oc = open('OC.dat','w')
97  pb = open('PB.dat','w')
98
99  print len(class_oc)
100
101 for i in range(len(class_oc)):
102         f_oc.write(class_oc[i][-1])
103         f_oc_only.write(class_oc[i][-1])
104         oc.write(class_oc[i][-1])
105         for j in range(len(x_new[i])):
106                 if x_new[i][j] != 0:
107                         f_oc.write(' '+str(j+1)+':'+str(x_new[i][j]))
108         for j in range(len(x_new_dl_only[i])):
109                 if x_new_dl_only[i][j] != 0:
110                         f_oc_only.write(' '+str(j+1)+':'+str(x_new_dl_only[i][j]))
111         for j in range(len(numpy_feature[i])):
112                 if numpy_feature[i][j] != 0:
113                         oc.write(' '+str(j+1)+':'+str(numpy_feature[i][j]))
114         f_oc.write('\n')
115         f_oc_only.write('\n')
116         oc.write('\n')
117
118 for i in range(len(class_oc),len(class_oc)+len(class_pb)):
119         f_pb.write(class_pb[i-len(class_oc)][-1])
120         f_pb_only.write(class_pb[i-len(class_oc)][-1])
121         pb.write(class_pb[i-len(class_oc)][-1])
122         for j in range(len(x_new[i])):
123                 if x_new[i][j] != 0:
124                         f_pb.write(' '+str(j+1)+':'+str(x_new[i][j]))
125         for j in range(len(x_new_dl_only[i])):
126                 if x_new_dl_only[i][j] != 0:
127                         f_pb_only.write(' '+str(j+1)+':'+str(x_new_dl_only[i][j]))
128         for j in range(len(numpy_feature[i])):
129                 if numpy_feature[i][j] != 0:
130                         pb.write(' '+str(j+1)+':'+str(numpy_feature[i][j]))
131         f_pb.write('\n')
132         f_pb_only.write('\n')
133         pb.write('\n')
134
135 f_oc.close()
136 f_pb.close()
137 f_oc_only.close()
138 f_pb_only.close()
139 oc.close()
140 pb.close()
```

ソースコード 2 識別素性 (入力素性+縮約素性) を生成するプログラム