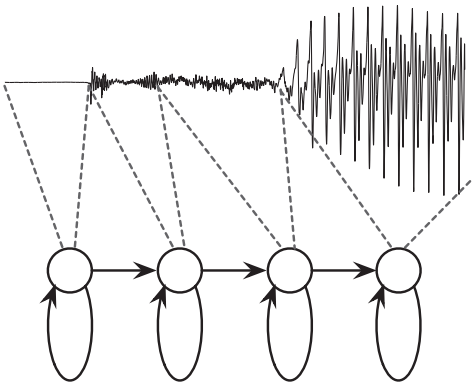
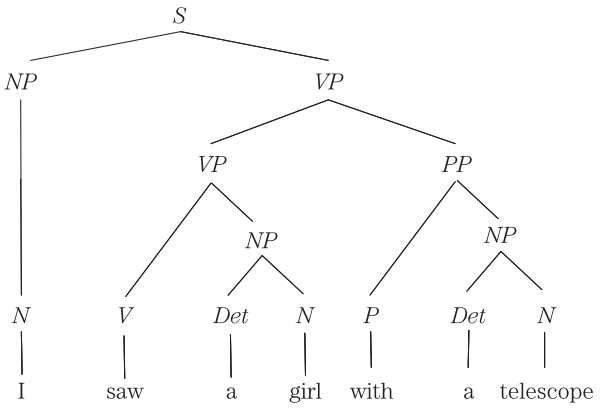
「音声言語処理と自然言語処理」演習

土屋 雅稔，山本 一公 共著





このテキストについて

このテキストは，教科書「音声言語処理と自然言語処理」（中川聖一 編著，コロナ社，2013 年）の読者を対象として，音声言語処理および自然言語処理のフリーソフトウェアを用いた演習手順について説明し，教科書の理解を深めることを目的としています。そのため，音声言語処理および自然言語処理の手法やアルゴリズムについての説明は省略し，演習を実際に実行する手順を主に説明します。手法やアルゴリズムの詳細については，教科書「音声言語処理と自然言語処理」を参照してください。

このテキストでは，以下のような演習を用意しています。

- Praat を用いて，音声のスペクトログラム，ピッチ，フォルマントを観察します（第 2 章，演習時間の目安: 1 時間）。
- HTK を用いて，音声データから MFCC を抽出します（第 3 章，演習時間の目安: 2 時間）。
- HTK を用いて，MFCC から音響モデルを学習し，得られた音響モデルに基づいて音声認識します（第 4 章，演習時間の目安: 4 時間）。
- HTK を用いて，数字単位の HMM を学習し，連続数字音声認識を行います（第 4 章，演習時間の目安: 4 時間）。
- MeCab を用いて形態素解析し，SRILM を用いて言語モデルを作成します（第 5 章，演習時間の目安: 4 時間）。
- CaboCha を用いて係り受け解析と固有表現抽出を行います（第 6 章，演習時間の目安: 2 時間）。
- Minise を用いて全文検索を行います（第 7 章，演習時間の目安: 4 時間）。
- Moses と GIZA++ を用いて統計的機械翻訳モデルを行います（第 8 章，演習時間の目安: 4 時間）。

第 2 章から第 4 章は音声処理に関する演習，第 5 章から第 8 章は言語処理に関する演習です。音声処理に関する演習と言語処理に関する演習は，相互に独立していますので，独立して演習を行うことができます。第 2 章から第 4 章までの演習は，内容に依存関係がありますので，まとめて実施する必要があります。第 6 章，第 7 章および第 8 章の演習は，第 5 章の内容に依存しています。第 5 章から第 8 章までの演習をまとめて実施する時間が確保できない場合は，第 5 章と第 6 章のようにペアで実施してください。

このテキストの演習では，音声データとして CENSREC–1 を，言語データとして Wikipedia 日英京都関連文書対訳コーパスを用いています。このテキストの演習は，これらのデータおよび各種フリーソフトウェアなくしては，成立し得ませんでした。データお

よび各種フリーソフトウェアの作成者の皆様に，深い感謝を捧げます。第 8 章の記述にあたっては，筑波大学で 2008 年に開催された統計的機械翻訳に関する講習会の実習マニュアルを参考にさせて頂きました。実習マニュアルを執筆された山本幹雄先生，藤井敦先生，内山将夫先生，宇津呂武仁先生に深く感謝します。また，統計的機械翻訳の実験条件については，京都フリー翻訳タスクの設定を参考にさせて頂きました。有用なデータを公開してくださっている Graham Neubig 先生に感謝します。

2013 年 3 月　著者記す

目次

第 1 章

演習環境の準備

本テキストでは，さまざまなフリーソフトウェアを用いて音声言語処理および自然言語処理についての演習を行う。この演習を行うための計算機環境として，筆者らは，Debian GNU/Linux 6.0[1] 上に必要なフリーソフトウェアとデータをインストールした仮想マシンイメージを用意した。本テキストは，この仮想マシンを各自の計算機上で実行して演習を行うことを前提として記述している。さらに，読者が，UNIX の操作方法についての基本的な知識（コマンドの実行，ディレクトリ・ファイルの操作，パイプ・リダイレクトの使い方，エディタの使い方）を有していることも前提としている。UNIX に触れた経験がない読者は，「Linux 標準教科書」[2] などの適当な参考書を参照して予習する必要がある。

まず，以下の手順にしたがって演習を行う環境を準備しなさい。

（1）各自の計算機に，仮想マシンを実行するためのソフトウェア VirtualBox[3] をインストール。

（2）教科書サポートサイト [4] から，仮想マシンイメージファイルを圧縮した zip ファイルをダウンロード。

（3）ダウンロードした zip ファイルから，仮想マシンイメージファイルを取り出す。

（4）VirtualBox を起動（図 1.1）し，「新規」をクリックする。仮想マシン作成ウィザードが開くので，その指示に従って以下のように入力していく。

（i）仮想マシン名を指定するダイアログ（図 1.2）では，オペレーティングシステムは「Linux」，バージョンは「Debian (64bit)」を選択する。なお，仮想マシン名は適当に決めて良い。

（ii）メモリ割当て量を指定するダイアログ（図 1.3）では，メインメモリのサイズとして，少なくとも 1GB 以上を割り当てる。

（iii）仮想ハードディスクについてのダイアログ（図 1.4）では，「すでにある仮想ハードドライブファイルを使用する」にチェックを入れて，取り出した仮想マシンイメージファイルを選択する。「作成」ボタンを押すと，図 1.5 のように表示されるはずである。

（5）第 2 章および第 3 章の演習では，各自の計算機上で Praat を用いて録音した音声データを，仮想マシン上のプログラムから参照する必要が生じる。以下の手順にしたがって，仮想マシンのホスト側での共有フォルダの設定を行う。

[1] http://www.debian.org/

[2] https://www.lpi.or.jp/linuxtext/file/linuxtext_ver1.1.0.pdf

[3] https://www.virtualbox.org/

[4] http://www.coronasha.co.jp/support/slp-and-nlp

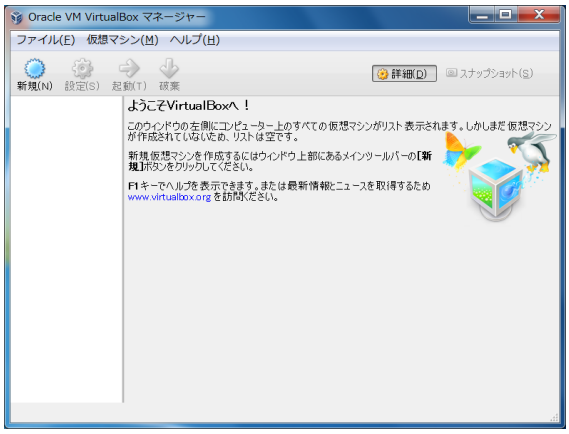


図 1.1　VirtualBox 初期画面

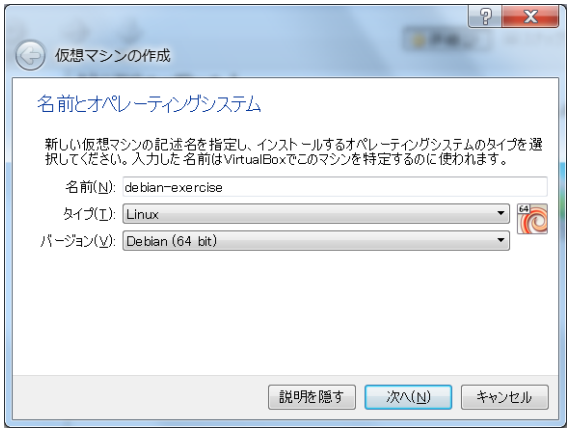


図 1.2　VirtualBox 「名前とオペレーティングシステム」ダイアログ

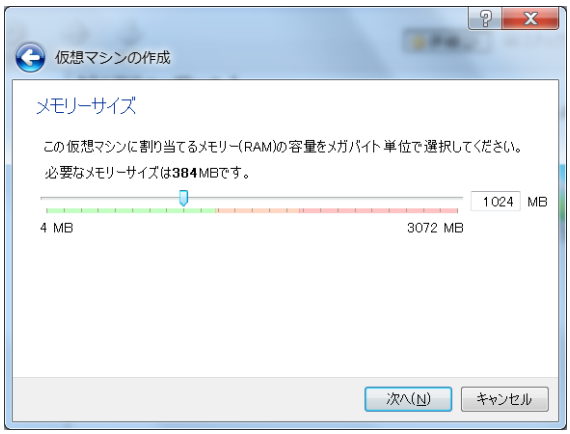


図 1.3　VirtualBox 「メモリーサイズ」ダイアログ

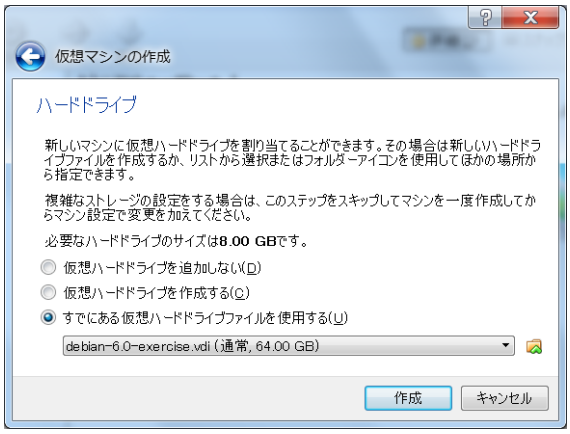


図 1.4　VirtualBox 「ハードドライブ」ダイアログ

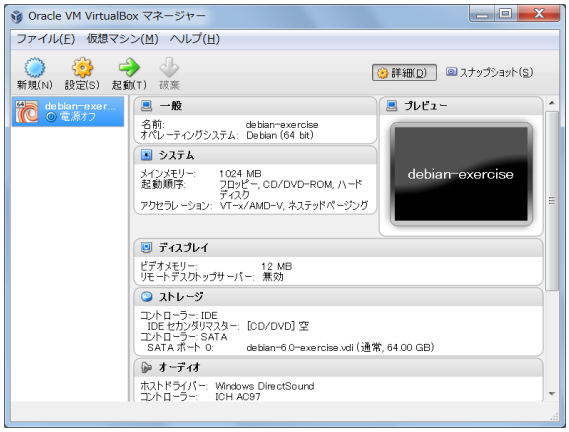


図 1.5　VirtualBox 仮想マシン作成後の画面

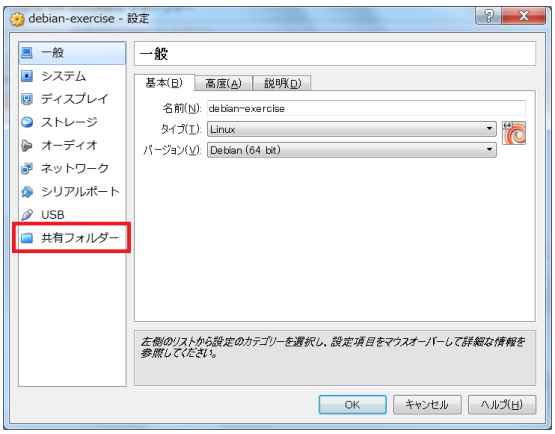


図 1.6　VirtualBox 「設定」ダイアログ

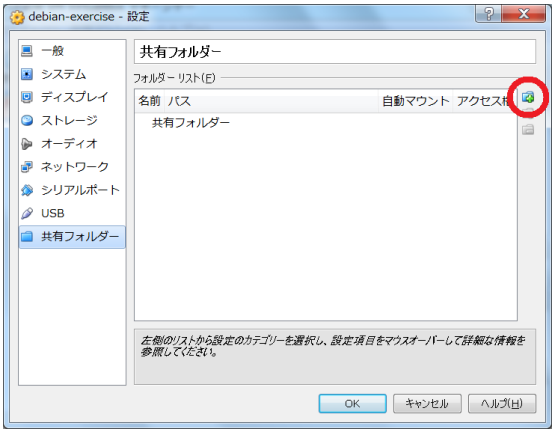


図 1.7　VirtualBox 「共有フォルダ」ダイアログ

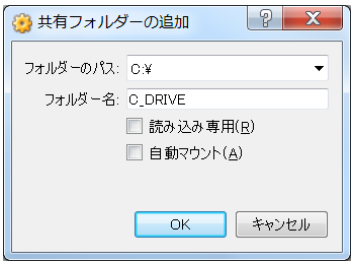


図 1.8　VirtualBox 「共有フォルダの追加」ダイアログ

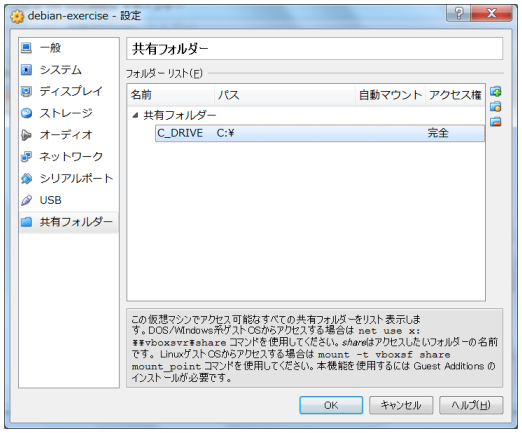


図 1.9　VirtualBox 「共有フォルダ」追加後のダイアログ

（i）VirtualBox（図 1.5）で「設定」をクリックする。
（ii）各種設定を行うダイアログ（図 1.6）で，「共有フォルダ」を選択する。
（iii）共有フォルダの設定を行うダイアログ（図 1.7）で，新たな共有フォルダを作成するアイコン（図 1.7 右上の赤丸で強調しているアイコン）をクリックする。
（iv）共有フォルダの追加を行うダイアログ（図 1.8）で，フォルダのパスとして「C:」を，フォルダ名として「C_DRIVE」と入力する。Praat を用いて録音した音声データを C: ドライブ以外に保存する場合は，適当に変更する。Mac OS または Linux 上に VirtualBox をインストールして使用する場合は，まず，Praat で録音する音声ファイルを保存しておくための適当なフォルダ（ディレクトリ）を作成しておく。フォルダのパスとして，作成したフォルダ（ディレクトリ）を指定し，フォルダ名として適当な分かりやすい名前をつける。共有フォルダの設定が正しく完了していれば，図 1.9 のように表示されるはずである。

（6）図 1.5 で「起動」をクリックして，仮想マシンを起動する。しばらく待つと，ログイン画面（図 1.10）が表示されるので，この Exercise User というユーザでログインする。パスワードは，初回ログイン時に，以下の手順にしたがって設定する。
（i）警告（図 1.11）が表示されるので，OK ボタンを押す。
（ii）パスワードを入力するダイアログ（図 1.12）が開くので，自分が使いたいパスワードを入力する。2 回目以後のログイン時には，ここで設定したパスワードが必要になるので，忘れないように注意すること。
（iii）パスワードを確認のために再入力するダイアログ（図 1.13）が開くので，先に入力したパスワードを再入力する。
ログイン画面（図 1.10）が表示されずに，エラー画面（図 1.14）が表示された場合は，VirtualBox のウェブサイト [5] などを参考に，BIOS の設定を変更すること。

[5] https://www.virtualbox.org/

（7）ログインしたら，図 1.15 のようにメニューをたどって端末（図 1.16）を開く。本テキストの演習は，基本的に，この端末上で行うことを想定している。

（8）第 2 章および第 3 章の演習では，各自の計算機上で Praat を用いて録音した音声データを，仮想マシン上のプログラムから参照する必要が生じる。以下のコマンドを実行して，仮想マシンのゲスト側で共有フォルダが参照できるように設定する。

```
% sudo mount -t vboxsf FOLDER_NAME /mnt ↵
```

FOLDER_NAME の部分には，共有フォルダの追加を行うダイアログ（図 1.8）で，フォルダ名として指定した文字列（上記の例では C_DRIVE）を指定する。この設定が完了すると，共有フォルダが /mnt 以下で参照できるようになっているはずである。

（9）筆者らが用意した仮想マシンでは，root ユーザのパスワードを無効化している。root 特権が必要な操作を行う場合は，sudo コマンドを用いて実行することができる。例えば，以下のコマンドを実行すると，Debian から公開されるセキュリティフィックスを適用することができる。

```
% sudo apt-get update ↵
% sudo apt-get upgrade ↵
```

または，以下のコマンドを実行して，root ユーザのパスワードを設定し，管理することもできる。

```
% sudo passwd root ↵
```

（10）仮想マシンを停止するときは，図 1.17 のようにメニューをたどれば良い。

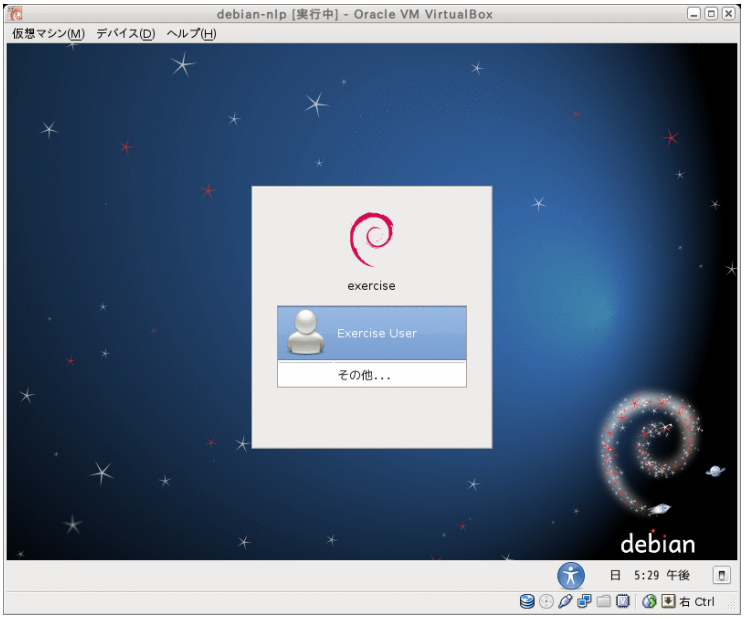


図 1.10　ログイン画面

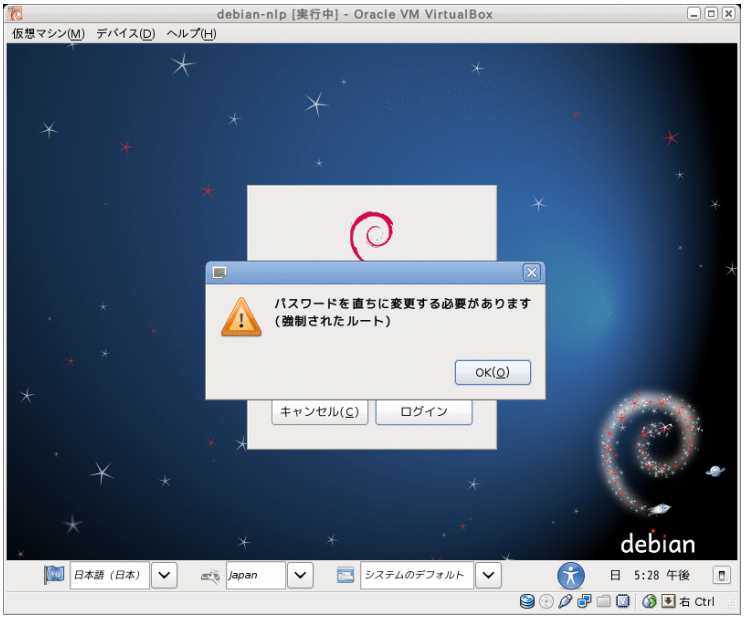


図 1.11　パスワード変更指示（初回ログイン時のみ表示）

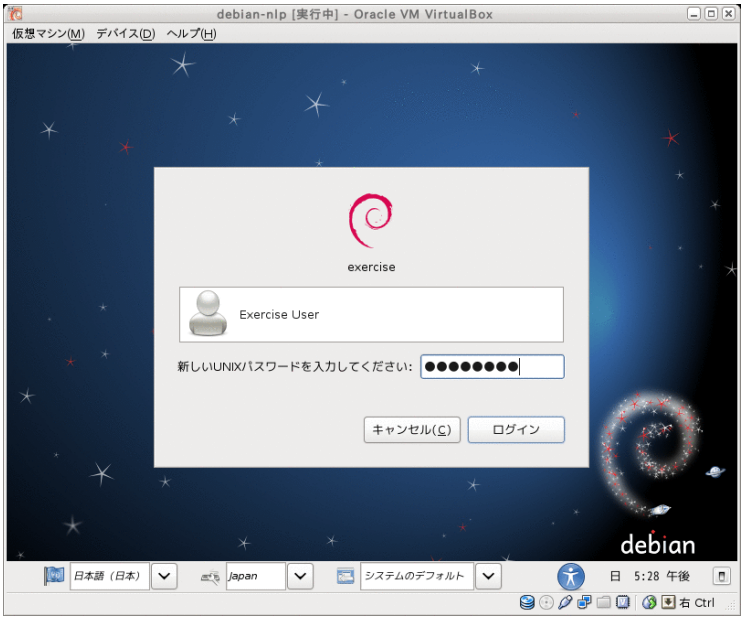


図 1.12　パスワード入力（初回ログイン時のみ表示）

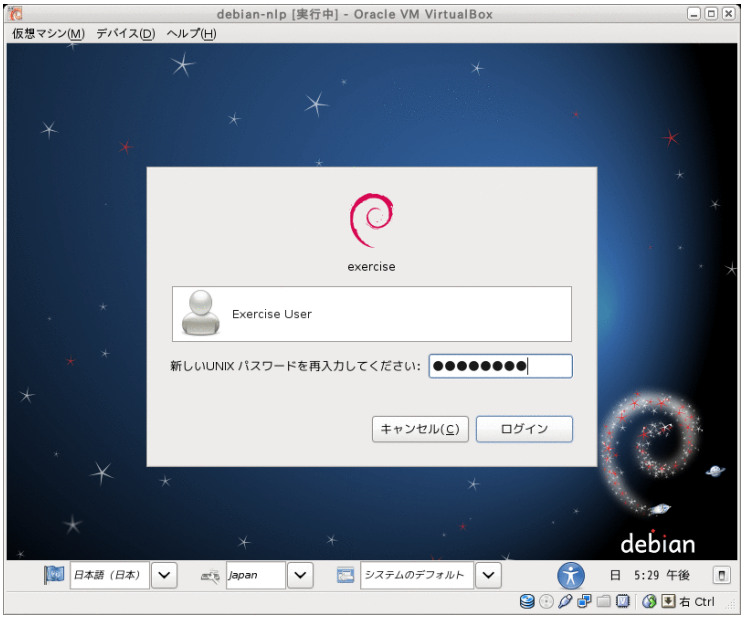


図 1.13　パスワード再入力（初回ログイン時のみ表示）

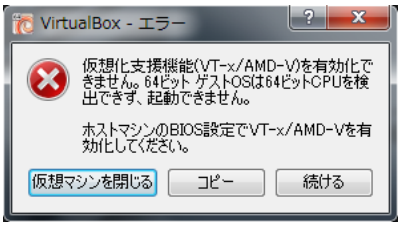


図 1.14　仮想マシン起動失敗時のエラー画面

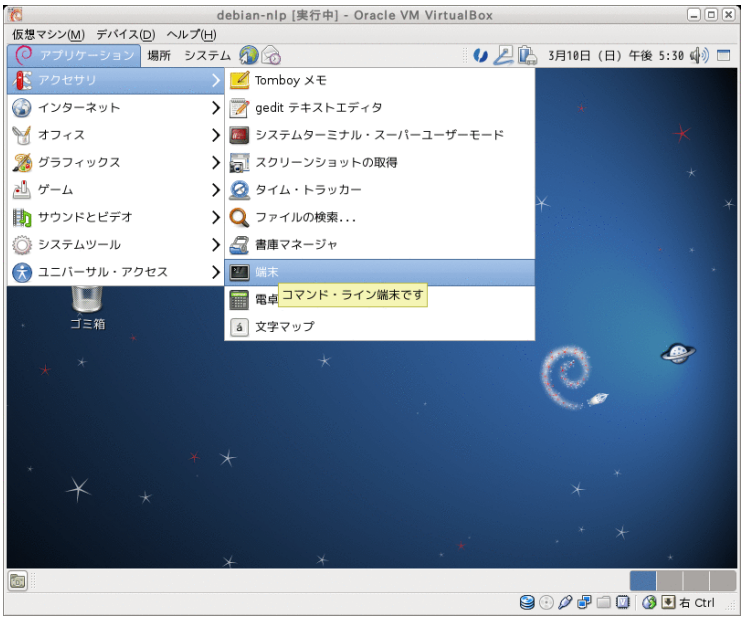


図 1.15　端末ウインドウを開くメニュー

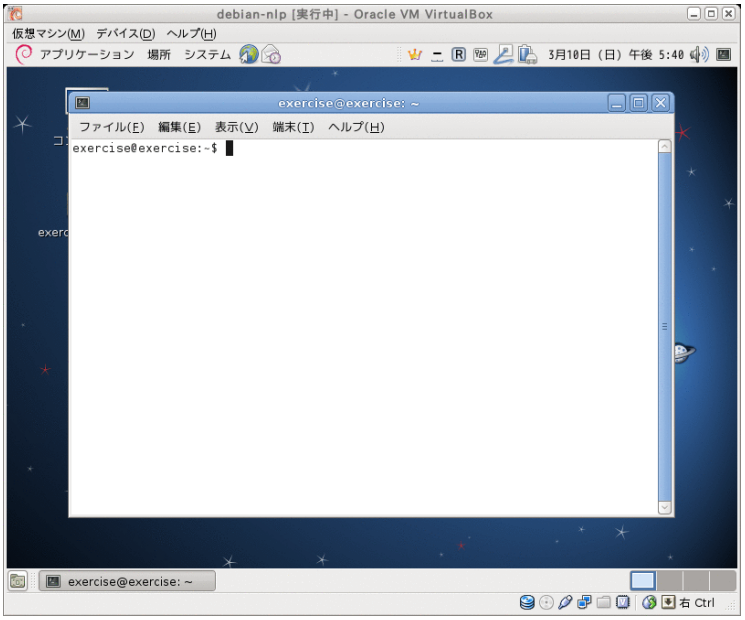


図 1.16　端末ウインドウ

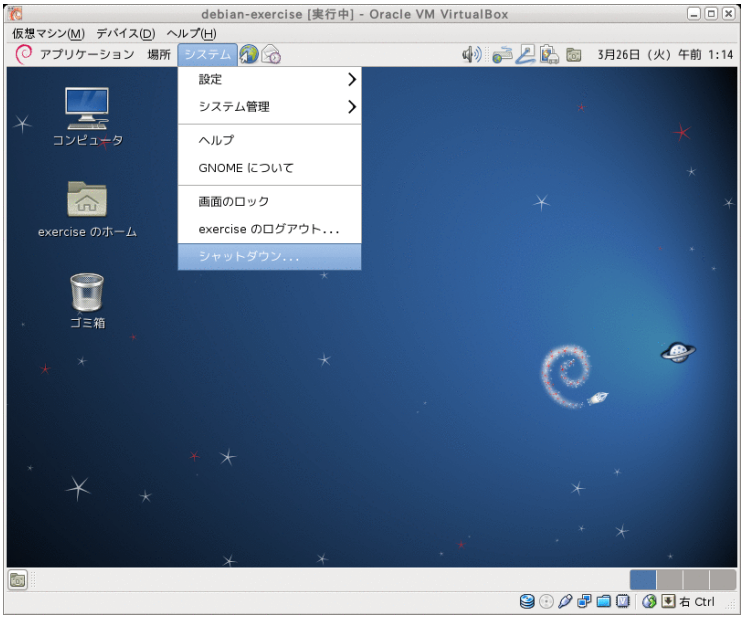


図 1.17　シャットダウンメニュー

第 2 章

音声分析

本章では，教科書 2.1.1 項 ~2.1.2 項に対応する演習として，Praat を用いて音声を録音し，音声波形および音声に含まれる情報の観察を行う。

仮想マシンは音声入力に問題がある場合が多いため，他の章の演習とは異なり，この章の演習に使用する Praat は読者自身の計算機にインストール・実行する必要がある。`http://www.fon.hum.uva.nl/praat/` から読者自身の計算機に対応するバイナリをダウンロードし，インストールして使用せよ。

マイクロホンを用意して使用せよ。できる限り良いマイクロホンを使用すること。なければ，ノート PC 本体に付属しているマイクロホンなどでも良い。

2.1 音声の録音

まず，音声を録音する。録音は，Praat のメニューバーの「New」から，「Record mono Sound」を選択することで行える（図 2.1）[1)]。

新しく表示される録音ウィンドウの左下にある「Record」ボタンを押すと録音が開始され，「Stop」ボタンを押すと録音が停止される。録音したら，「Save to list & Close」ボタンを押して，録音した音声を編集するためのリストに保存せよ（図 2.2）。

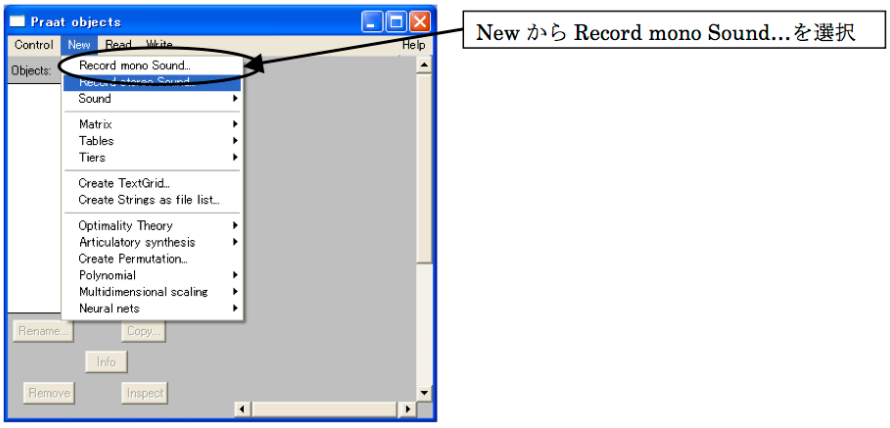


図 2.1　Praat で録音を選択（Praat objects ウィンドウ）

[1)] 以降の図は全て，Windows 版 Praat Version 5.1.11 のスナップショットである。

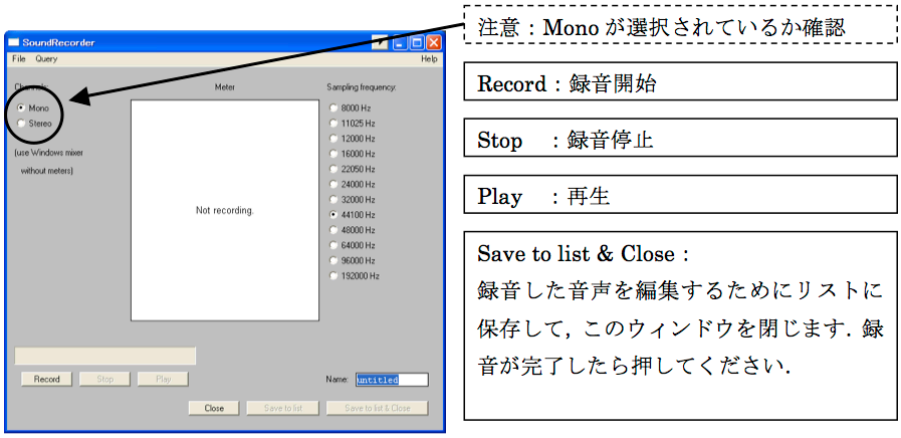


図 2.2　Praat で録音を開始 （SoundRecorder ウィンドウ）

課題 2–1: 音声の録音

自分の「あいうえお」という連続音声を Praat で録音せよ。

2.2　音声波形およびスペクトログラムの観察

録音した音声をリストに保存したら，録音ウィンドウを「Close」ボタンで閉じる。そうすると，リストに先ほど保存した音声が登録されているのが見えるだろう。その音声がリストに登録された状態で，リストの右側にある「View & Edit」ボタンを押すと，音声波形とスペクトログラムが表示されたウィンドウが開く（図 2.3)。スペクトログラムの下にある「Total duration XXXX seconds」と表示されているところをクリックすれば，音声を再生することもできる。また，View & Edit ウィンドウにおいて，カーソル線を観察したい時刻に合わせ，Ctrl+L を押すことで，選択した時刻の短時間スペクトルを観察することもできる（図 2.4)。

課題 2–2: 音声波形およびスペクトログラムの観察

録音した自分の「あいうえお」という連続音声の音声波形とスペクトログラムを Praat で観察せよ。

2.3　ピッチ・フォルマントの観察

デフォルトの設定では，何も操作をしなくてもスペクトログラムに重ねてピッチとフォルマントが表示されるようになっている。

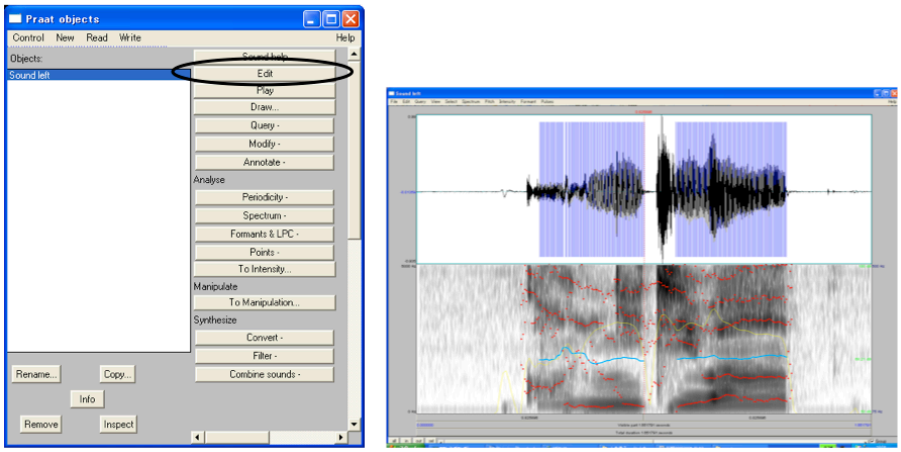


図 2.3　Praat で音声波形を見る （View & Edit ウィンドウ）

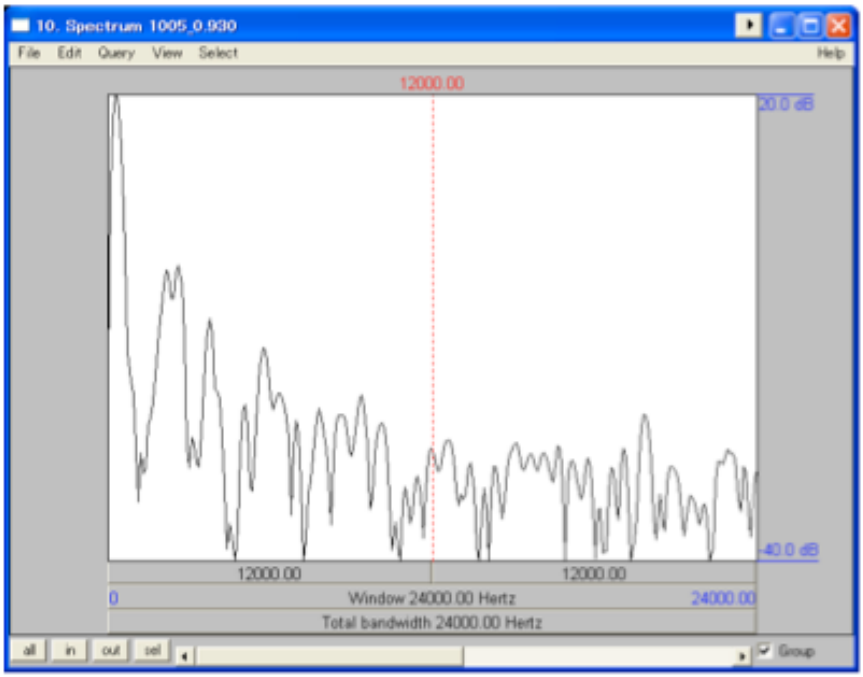


図 2.4　スペクトル表示ウィンドウ （Ctrl+L で出現）

課題 2–3: ピッチ・フォルマントの観察

(1) 録音した自分の「あいうえお」という連続音声のピッチを Praat で観察せよ。自分の声の高さは何 Hz 程度だろうか？

(2) 録音した自分の「あいうえお」という連続音声のフォルマントを Praat で観察し，F_1/F_2 を，教科書 p.21 図 2.5 で示されるものと比べてみよ。図で示される範囲内に収まっているであろうか？

2.4　ピッチ・フォルマントの変更

2.4.1　ピッチの変更

Praat objects ウィンドウから，対象となる音声を選択し，「Manipulate -」ボタンから「To Manipulation...」を選択する。「OK」を押すと，オブジェクトウィンドウに新しいファイルができる。その新しいファイルを選択して，「View & Edit」ボタンを押す。

View & Edit ウィンドウ上でピッチを変更したい部分波形を選択しておいて，メニューバーの「Pitch」から「Multiply pitch frequencies...」を選択，音の高さを何倍にするかを入力して，「OK」を押す。

オブジェクトウィンドウで，「Play （overlap-add）」を押すと，再生することができる。

2.4.2　継続時間長の変更

Praat objects ウィンドウから，対象となる音声を選択し，「Manipulate -」ボタンから「To Manipulation...」を選択する。「OK」を押すと，オブジェクトウィンドウに新しいファイルができる。その新しいファイルを選択して，「View & Edit」ボタンを押す。

メニューバーの「Dur」から「Add duration point at cursor Ctrl+D」を選択すると，View & Edit ウィンドウの一番下に点が表示されるので，その点の高さを変更することで時間長を変更することができる。

オブジェクトウィンドウで，「Play （overlap-add）」を押すと，再生することができる。

課題 2–4: ピッチ・フォルマントの変更

録音した自分の「あいうえお」という連続音声のピッチを 1.5 倍，2/3 倍に変更し，再生してその変化を確認せよ。同様に継続時間を 1.5 倍，2/3 倍に変更し，再生してその変化を確認せよ。

第 3 章

特徴抽出

本章では，教科書 2.1.4 項に対応する演習として，HTK を用いた特徴量の抽出を行う。この特徴量は，次章の演習で用いる。

本章の記述は，演習に必要なことだけを簡潔に抜き出したものである。詳細については，[1] を参照すること。

3.1 音響特徴抽出

音声の特徴抽出は，HCopy コマンドを用いて行う。ここでは，現在の音声認識技術で最も一般的に利用されている MFCC 特徴を抽出する。

ひとつの音声ファイルから MFCC 特徴を抽出する最も簡単なやり方は，以下のようになる。

```
% HCopy -C config_hcopy speech_file_name.wav mfcc_file_name.mfc ↵
```

ここで，speech_file_name.wav は音声波形のファイル名，mfcc_file_name.mfc は抽出された特徴量が出力されるファイル名となる。-C は，分析の条件を指定するためのコンフィギュレーションファイルを与えるオプションで，リスト 3.1 のような内容のファイルを指定する。

リスト 3.1 config_hcopy

```
1  SOURCEKIND     = WAVEFORM
2  SOURCEFORMAT   = WAV
3  SOURCERATE     = 625.0
4
5  TARGETKIND     = MFCC_E_0
6  TARGETRATE     = 100000.0
7  WINDOWSIZE     = 250000.0
8  USEHAMMING     = T
9  PREEMCOEF      = 0.97
10 NUMCHANS       = 24
11 CEPLIFTER      = 22
12 NUMCEPS        = 12
13
14 SAVEWITHCRC    = F
```

リスト 3.2　speech_list_file

```
1 /path/to/wav/speech1.wav   /path/to/mfcc/speech1.mfc
2 /path/to/wav/speech2.wav   /path/to/mfcc/speech2.mfc
3 /path/to/wav/speech3.wav   /path/to/mfcc/speech3.mfc
4 /path/to/wav/speech4.wav   /path/to/mfcc/speech4.mfc
5 /path/to/wav/speech5.wav   /path/to/mfcc/speech5.mfc
```

SOURCEKIND は分析元ファイルの種類で，今回は波形ファイルのため，WAVEFORM とする。 SOURCEFORMAT は，WAV ファイルの場合は WAV とする。ヘッダー無しの音声ファイルの場合は NOHEAD とし，BYTEORDER により音声ファイルのバイトオーダーを指定する必要がある。SOURCERATE は，標本化周期（標本化周波数の逆数）を 100ns を単位として表した数値である。例は 16kHz の標本化周波数の場合となっている。

TARGETKIND は，求めたいパラメータの種類で，例では MFCC とエナジーパラメータおよび 0 次のケプストラム係数を求めるように指定している。TARGETRATE はフレーム周期を 100ns を単位として表した数値，WINDOWSIZE は分析窓長を 100ns を単位として表した数値で，例では 10ms のフレーム周期で，25ms 長の窓を使うように指定している。USEHAMMING は T とすることで分析窓としてハミング窓を使うように指定している。PREEMCOEF にはプリエンファシスの係数を指定する。NUMCHANS にはメルフィルタバンクのフィルタ数を，CEPLIFTER には求めたケプストラムに掛ける等価リフタの次数を指定する。NUMCEPS には最終的に高次係数を切り捨てて求める MFCC の次数を指定する。例では，フレームごとに 12 次元の MFCC と 1 次元のエナジーパラメータ，計 13 次元の特徴量が求まる。

複数の音声ファイルを一度に分析する場合は，リスト 3.2 のようなファイルを作って，HCopy に -S オプションで与えることにより，一度に分析できる。

```
% HCopy -C config_hcopy -S speech_list_file ↵
```

課題 3–1: 特徴抽出

(1) Praat を使って，自分の孤立 5 母音（あ，い，う，え，お）をモノラル，サンプリング周波数 16kHz で 10 回ずつ録音し，WAV フォーマットで適当なファイル名を付けてセーブせよ。その際，実際に発話している音声区間の前後にある無音区間は削除してからセーブすること。録音の段階でサンプリング周波数，量子化ビット数を指定できない場合は，Praat objects ウィンドウの「Convert -」から「Resample...」を選択し，「New sampling frequency [Hz]」を 16000 にしてリサンプリングすること。

(2) 上記を参考に，音声データから MFCC を抽出し，適当なファイル名を付けて保存せよ。ただし，拡張子は “.mfc” とすること。

第 4 章

音声認識

本章では，教科書 2.2 節に対応する演習として，音響モデルの学習と，音声認識を行う。まず，簡単な音響モデル（GMM）による母音認識を行い，その後，もう少し複雑な HMM による連続数字認識を行う。

本章の記述は，前章と同様，演習に必要なことだけを簡潔に抜き出したものである。詳細については，[1] を参照すること。

4.1 音響モデル学習の概要

音響モデルの学習は，通常，初期モデルを学習してから，それを用いて混合ガウス分布の混合数を上げながら連結学習を行うという手法が取られる。初期モデルは，時間詳細ラベルが使える場合と使えない場合で作り方が異なる。ここでは，音素モデルを例にする。

4.1.1 ラベルが必要ない場合と時間詳細ラベルがある場合の初期モデル推定

ラベルが必要ない場合（ひとつの発話ファイル全体をひとつのモデルで表現する場合）と時間詳細ラベル（各音素の始端時刻と終端の時刻が記述されているラベル）が使える場合は，HInit コマンドと HRest コマンドを用いて初期モデルを作成する。HInit は k-means アルゴリズムを用いて，状態ごとに特徴パラメータをクラスタリングし，初期パラメータ（ガウス分布の平均ベクトルと分散共分散行列）を生成する。HRest は，HInit で生成された初期モデルに対して EM 学習によりパラメータの再推定を行う。

まず，プロトタイプモデルの作成を行う。プロトタイプモデルは，モデルの雛形で，推定すべきパラメータの代わりに，適当な値（平均値には 0.0，分散値には 1.0）を入れて作成する。例えば，先ほど求めた 13 次元の特徴パラメータを使って，デルタパラメータをもつ 26 次元の特徴量のモデルを作る場合，出力状態数が 3 状態の音素モデルのプロトタイプはリスト 4.1 のようになる。

特徴量の種別（1 行目）が `MFCC_E_D` となり，`_D` が付くことで，デルタパラメータをもつモデルになる。分散共分散行列をパラメータとして持たせる場合は，大量の学習データが必要になるので，ここでは対角共分散行列の場合の例となっている。HTK で扱う HMM には，出力分布を持たない初期状態と最終状態が付くことに注意すること。リスト 4.1 の

リスト 4.1　音素モデルのプロトタイプ prototype

```
~o <VECSIZE> 26 <MFCC_E_D> <STREAMINFO> 1 26
~h "prototype"
<BEGINHMM>
  <NUMSTATES> 5
  <STATE> 2 <NUMMIXES> 1
  <STREAM> 1
  <MIXTURE> 1 1.0000
    <MEAN> 26
      0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
      0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
    <VARIANCE> 26
      1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
      1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
  <STATE> 3 <NUMMIXES> 1
  <STREAM> 1
  <MIXTURE> 1 1.0000
    <MEAN> 26
      0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
      0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
    <VARIANCE> 26
      1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
      1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
  <STATE> 4 <NUMMIXES> 1
  <STREAM> 1
  <MIXTURE> 1 1.0000
    <MEAN> 26
      0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
      0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
    <VARIANCE> 26
      1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
      1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0 1.0
  <TRANSP> 5
   0.000E+0   1.000E+0   0.000E+0   0.000E+0   0.000E+0
   0.000E+0   6.000E-1   4.000E-1   0.000E+0   0.000E+0
   0.000E+0   0.000E+0   6.000E-1   4.000E-1   0.000E+0
   0.000E+0   0.000E+0   0.000E+0   6.000E-1   4.000E-1
   0.000E+0   0.000E+0   0.000E+0   0.000E+0   0.000E+0
<ENDHMM>
```

例は 3 出力状態 [1] だが，総状態数（<NUMSTATES>）は 5 になっている。また，出力状態を 1 状態とすることで，単純なガウス混合モデル（GMM）を模擬することもできる。2 行目の ~h はモデル名を与えるマクロであり，この例では prototype というモデル名を指定している。学習するモデルに合わせて，これを変更する必要がある。

続いて，ラベルファイルを用意する（ラベルが必要ない場合は必要ない）。時間詳細情報のついたラベルは，例えばリスト 4.2 のようになる。

ラベルは，最初の数字が当該音素の始まるフレームの時刻，2 番目の数字が当該音素の終わるフレーム（の次のフレーム）の時刻を表している。時刻はコンフィギュレーションファイル（リスト 3.1）で用いたものと同じく，100ns を単位として表した数値になっている。3 番目が音素のラベルである。ダブルクォーテーションで囲まれているものは，そ

[1] 分布の定義は，5 行目の第 2 状態（<STATE> 2）から 23 行目の第 4 状態（<STATE> 4）まで

リスト 4.2　時間詳細情報を含むラベルファイル

```
#!MLF!#
"*/speech1.lab"
0 2500000 sil
2500000 3100000 a
3100000 3400000 r
3400000 4400000 a
（中略）
36300000 37000000 n
37000000 37800000 o
37800000 38300000 d
38300000 39700000 a
39700000 42100000 sil
.
"*/speech2.lab"
0 2400000 sil
2400000 3000000 i
3000000 3300000 q
3300000 5600000 sh
5600000 7200000 u:
（以下略）
```

の区画（ダブルクォーテーションで囲まれたものの次の行から，.（ピリオド）まで）が，どの音声特徴量ファイルに対応するかを示している。つまり，*/speech1.lab であれば，speech1.mfc という音声特徴量ファイルに対応するラベルであることを示している。

プロトタイプモデルとラベルファイルが準備できたら，以下のようにすることで，HInit により初期モデルを学習できる。

```
% cp prototype a ↵
（ファイル内のモデル名を変更）
% HInit -C config_train -I label_file -S mfcc_list_file
  -M output_iter01_mix01 -l a -i 100 a ↵（実際は 1 行）
```

-I オプションで時間詳細情報付きのラベルファイルを指定する。-M オプションで，初期パラメータがセットされたモデルを出力するディレクトリを指定する。-l オプションで，どのラベルのモデルを学習するかを指定する。上記の例では，音素/a/のモデルを学習する。このとき，モデル内の ~h マクロで指定されるモデル名をモデルファイル名（この場合は a）に変更しておく必要がある。-i オプションは，k-means アルゴリズムの最大繰り返し回数を指定する。上記の例では最大 100 回としている。-C オプションでは，リスト 4.3 のようなコンフィギュレーションファイルを与える。

リスト 4.3　HInit の設定ファイル config_train

```
SOURCEKIND = MFCC_E_0
TARGETKIND = MFCC_E_D
```

リスト 4.3 の例では，入力に用いる特徴パラメータは，先ほど HCopy で求めた MFCC_E_0

とし，モデルで使う特徴パラメータを MFCC_E_D として，0 次ケプストラム係数を取り除いた上で，デルタパラメータを加えている。この場合，デルタパラメータは HTK が内部で自動的に計算してパラメータに追加するので，ユーザが心配することはない。

-S オプションで与える mfcc_list_file も HCopy の場合とは異なり，リスト 4.4 のように MFCC のファイル名を 1 段で与える。

リスト 4.4　mfcc_list_file

```
1 /path/to/mfcc/speech1.mfc
2 /path/to/mfcc/speech2.mfc
3 /path/to/mfcc/speech3.mfc
4 /path/to/mfcc/speech4.mfc
5 /path/to/mfcc/speech5.mfc
```

これを /a/ だけではなく，学習したい音素すべてに対して行う。シェルスクリプトなどを用いて，自動的に学習できるようにすると良いだろう。

ラベルが必要ない場合もほぼ同様だが，ラベル情報は不要なので，

```
% cp prototype a ↵
（ファイル内のモデル名を変更）
% HInit -C config_train -S mfcc_list_file -M output_iter01_mix01
  -i 100 a ↵（実際は 1 行）
```

のようにする（ラベルファイルの指定とラベル中のモデル名の指定がなくなる）。ただし，この場合でも，モデル内の ~h マクロで指定されるモデル名をモデルファイル名（この場合は a）に変更しておく必要がある。

HInit で初期モデルを作成できたら，HRest を用いて EM 学習によりパラメータを更新していく。具体的には，以下のようにする。

```
% HRest -C config_train -I label_file -S mfcc_list_file
  -M output_iter02_mix01 -l a -i 100
  output_iter01_mix01/a ↵（実際は 1 行）
```

オプションは，HInit と同じである。こちらも，学習したい音素すべてに対して学習できるように，シェルスクリプトなどを用いて，自動的に学習できるようにすると良い。

ラベルが必要ない場合も同様に，

```
% HRest -C config_train -S mfcc_list_file -M output_iter02_mix01
  -i 100 output_iter01_mix01/a ↵（実際は 1 行）
```

となる。

4.1.2　時間詳細ラベルがない場合の初期モデル推定

時間詳細ラベルは，音声波形やスペクトログラムの情報を頼りに，人手作業によって付けられるもので，作成コストが非常に高い。そのため，時間詳細情報の付いていない，発話内容だけ（音素だけ）のラベルが用いられることもよくある。その場合は，HCompV コマンドを用いて初期モデルを作成する。HCompV は，すべての学習データを用いて，学習データ全体の平均ベクトルと分散共分散行列を計算し，それをすべての状態の初期パラメータとする。非常に大雑把な方法だが，その後の連結学習によってモデルを鍛えることで，十分実用に耐えるモデルを学習することができる。

時間詳細情報がないラベルはリスト 4.5 のようになる。

リスト 4.5　時間詳細情報を含まないラベルファイルの例

```
#!MLF!#
"*/speech1.lab"
sil
a
r
a
（中略）
n
o
d
a
sil
.
"*/speech2.lab"
sil
i
q
sh
u:
（以下略）
```

時間情報がないと音素の区切り時刻が分からないので，正確な初期モデルを作ることはできない。そこで，HCompV を使って，大雑把ではあるが初期モデルを作る。プロトタイプモデル，音声特徴量ファイルリストを時間詳細ラベルがある場合の初期モデル推定と同じように用意して，以下のようにする。

```
% HCompV -C config_train -S mfcc_list_file -M output_iter02_mix01
  -f 0.01 -m a ↵（実際は 1 行）
```

HCompV は学習データ全体の平均ベクトルと分散共分散行列を計算するプログラムなので，ラベルファイルを使用しない。-f オプションは，与えた数値と計算された分散の値を掛け合わせたものを分散のフロア（底の値）とするように指示する。これによって，分散が小さくなりすぎるのを防止する。-m オプションは，平均を更新するように指示する。この例では/a/のモデルを作っている（最後の引数がモデル名）が，/i/のモデルでも/u/

のモデルでも，同じパラメータのモデルしかできないので，ひとつ作って，それをコピーすると早くできる。

4.1.3　連結学習と混合数の増加

連結学習には，HERest コマンドを用いる。また，状態ごとの混合ガウス分布の混合数を増やすには，HHEd コマンドを用いる。以下に，1 混合ガウス分布の初期モデルを連結学習と混合数増加を繰り返すことで，4 混合にする例を示す。

まず，1 混合の初期モデルを連結して，リスト 4.12 に示すような，ひとつの大きなモデルファイルを作成する。そのファイル名を，hmmdefs とする。

すべてのモデルがひとつのモデルファイルに収まったら，まず 1 混合での連結学習を行う。

```
% HERest -C config_train -S mfcc_list_file -M output_iter03_mix01
  -H output_iter02_mix01/hmmdefs -I label_file_2
  monophone_list ↵（実際は 1 行）
```

-H オプションで先ほど作成したひとつの大きなモデルファイルを指定する。-I オプションで与えるラベルは，時間詳細情報があるものでもないものでも構わないが，時間詳細情報が付いていても使用されない。ここでは，時間詳細情報がないものを指定している。monophone_list は学習したいモデルセットのモデル一覧で，音素モデルを学習する場合は，リスト 4.6 のようになる。

リスト 4.6　monophone_list

```
N
a
b
by
（中略）
w
y
z
```

HERest は EM 学習を 1 回行うだけで，プログラム内で繰り返しが行われないため，10 回程度繰り返してコマンドを動かさないとパラメータが十分に学習されない。シェルスクリプトなどで上記コマンドを自動的に 10 回程度繰り返すようにすると良い。

次に，混合数が 1 のモデルを 2 混合にする。10 回学習されたモデルファイルが，output_iter12_mix01 に入っているとして，そのモデルに対して HHEd コマンドを使う。

```
% HHEd -H output_iter12_mix01/hmmdefs -w output_iter00_mix02/hmmdefs
  mixup02.hed monophone_list ↵（実際は 1 行）
```

-w オプションで，混合数を 2 にしたモデルの出力先を指定する。mixup02.hed は

HHEd のコマンドファイルで，この中に混合数を増加させる命令を書いておく。具体的には，図 4.7 のようになっている。

リスト 4.7　mixup02.hed

```
MU 02 \{*.state[2-4].mix\}
```

MU が混合数を操作する命令で，02 によりその後に指示される状態に対して，混合数を 2 にする。*によりすべてのモデルが対象となり，状態は 2 状態目から 4 状態目，即ち出力分布をもつ状態すべてが対象となることがわかる。これによって，モデルの全状態の混合数が 2 になる。

HHEd 直後のモデルパラメータは学習されていない状態なので，HERest で再び学習する。

```
% HERest -C config_train -S mfcc_list_file -M output_iter01_mix02
  -H output_iter00_mix02/hmmdefs -I label_file_2
  monophone_list ↵（実際は 1 行）
```

これを再び 10 回程度繰り返す。その後，再び HHEd で，今度は混合数を 4 に上げる。10 回学習された 2 混合のモデルファイルが，output_iter10_mix02 に入っているとして，そのモデルに対して HHEd コマンドを使う。

```
% HHEd -H output_iter10_mix02/hmmdefs -w output_iter00_mixdir_04/hmmdefs
  mixup04.hed monophone_list ↵（実際は 1 行）
```

mixup04.hed の内容はリスト 4.8 のようになる。

リスト 4.8　mixup04.hed

```
MU 04 \{*.state[2-4].mix\}
```

HHEd 直後のモデルパラメータは学習されていない状態なので，HERest で再び学習していく。

```
% HERest -C config_train -S mfcc_list_file -M output_iter01_mix04
  -H output_iter00_mix04/hmmdefs -I label_file_2
  monophone_list ↵（実際は 1 行）
```

これを再び 10 回程度繰り返すと，4 混合のモデルが出来上がる。これ以上の混合数のモデルも同じように HHEd と HERest を繰り返すことで学習できる。混合数は一度に大きく増やしても精度良く学習できないので，増やすときは元モデルの 2 倍を上限とすること。

4.2　記述文法を用いた音声認識

4.2.1　記述文法

HTK では，拡張 BNF（EBNF）記法を用いて，文法を記述することができる（確率的な言語モデルを扱うこともできるがここでは使用しない）。リスト 4.9 に，連続数字認識に用いるための簡単な例を示す。

リスト 4.9　wordnet.syn

```
$digit = ichi | ni | saN | yoN | go | roku
         | nana | hachi | kyu | zero | maru;

 (
  [sil] < $digit [sp] > [sil]
)
```

リスト 4.9 は，数字（1 ~ 9, 0（2 種類））の連続発声を認識するための文法となっている。[と] で囲まれたものは，出現してもしなくてもよい項を，< と > で囲まれたものは，この区間を 1 回以上の任意回数繰り返すことを示している。すなわち，発声の先頭，末尾と数字の間に無音が出現したりしなかったりしながら，変数 $digit に列記された数字モデルのいずれかが 1 回以上の任意回数繰り返されるという文法になっている（sil は長い無音，sp は短い無音を表している）。

文法が書けたら，HParse を使ってネットワークファイルに変換する。

```
% HParse wordnet.syn wordnet.net ↵
```

HVite で認識を行う際は，この wordnet.net を使う。

認識で用いる辞書ファイルも作成しておく。リスト 4.10 は，音素モデルを用いる場合の日本語数字発声用の辞書である。第 1 カラムに単語（wordnet.syn の記述に用いた単位）を書き，第 2 カラム以降にその単語に対応するモデル（音素，音節など学習した音響モデルによるその単語を表現する）列を記述する。

一方，単語単位のモデルを用いる場合は，単語と音響モデルが 1 対 1 対応になるため，リスト 4.11 のように第 1 カラムと第 2 カラム以降が 1 対 1 の対応になるように記述する。

リスト 4.10，リスト 4.11 の第 2 カラムの数値は，ひとつの単語に対して複数の読み方を登録する場合にそれぞれの読み方が出現する確率値となっている。ここでは、読み方はひと通りのため、確率値を 1.0 としている [2]。

[2] 本来は，リスト 4.10 やリスト 4.11 のようにひとつの単語に対して複数の読み方を登録しない場合には第 2 カラムの数値は省略して構わない。しかし，リスト 4.11 の場合は、/nana/ の部分文字列である “nan” が HTK 内部で “Not a number” と解釈されてしまいエラーとなるため，それを避ける目的で確率値として 1.0 を記述する。

4.2.2　HVite の使い方

実際の認識は，HVite コマンドにより行われる。以下に具体例を示す。

```
% HVite -C config_train -H /path/to/trained/hmmdefs
  -S test_mfcc_list_file -i result.mlf -l '*'
```

リスト 4.10　digit-phone.dic

```
ichi  1.0 i ch i
ni    1.0 n i
saN   1.0 s a N
yoN   1.0 y o N
go    1.0 g o
roku  1.0 r o k u
nana  1.0 n a n a
hachi 1.0 h a ch i
kyu   1.0 ky u:
zero  1.0 z e r o
maru  1.0 m a r u
sil   1.0 sil
sp    1.0 sp
```

リスト 4.11　digit-digit.dic

```
ichi  1.0 ichi
ni    1.0 ni
saN   1.0 saN
yoN   1.0 yoN
go    1.0 go
roku  1.0 roku
nana  1.0 nana
hachi 1.0 hachi
kyu   1.0 kyu
zero  1.0 zero
maru  1.0 maru
sil   1.0 sil
sp    1.0 sp
```

```
-w wordnet.net digit.dic monophone_list ↵（実際は 1 行）
```

-C オプションで与える config_train は学習時に用いた config_train と同じものである。-H オプションでは，認識に用いる学習済みの HMM を与える。-S オプションで，認識したい特徴量ファイルのリスト（学習に用いた mfcc_list_file と同じ形式のもの）を与える。-i オプションで認識結果を出力するファイル名を与える。認識結果の出力はラベルファイルと同じ形式となる。-l オプションで，認識結果ファイルに含まれるパスを指定できる。不要なら '*' としておけばよい。-w オプションで，前節で作成したネットワークファイルを与える。その後に，辞書ファイルとモデルリストを与えて実行させる。上記の例では，認識結果は result.mlf に格納される。

4.2.3　認識結果の集計

大規模な認識結果の集計を手作業で行うのは大変なので，集計用のコマンド HResults が用意されている。以下に具体例を示す。

```
% HResults -e "???" sil -e "???" sp -I reference.mlf
```

```
monophone_list result.mlf ↵（実際は 1 行）
```

-e オプションは，集計の際に置き換えを行う（同じラベルと見なす）ためのものである。この例では，sil と sp をそれぞれ "???"（null ラベル）と等価と見なすことで，結果集計の際に無視している（集計結果に sil と sp は現れない）。-I オプションで，正解のラベルファイルを与える。その後にモデルリストを与え，最後に認識結果が入っているファイルを与える。

課題 4–1: 5 母音の認識

前章で抽出した MFCC を用いて，母音の学習・認識実験を行う。10 回分の発声の内，8 回分を学習データ，2 回分をテストデータとして用いよ。

(1) 8 回分の発声を用いて 5 母音のモデルを学習する。学習データに対応する図 4.4 のようなファイルリストを作成せよ。
(2) ラベルとして，詳細時間情報無しラベルを用意せよ。録音時に発話前後の無音区間を削除しているので，1 発話にひとつの母音だけをもつようなラベルとなる。
(3) 出力状態が 1 状態（状態数が 3 状態）であるようなモデルの雛形を作成せよ。
(4) 今回は 1 発話に 1 音素のみしか含まれていないため，HInit & HRest によりモデルを学習せよ。
(5) EBNF 記法により，1 発声で 5 母音のいずれかが単独で出現する文法を記述し，HParse によりネットワークに変換せよ。
(6) 上記文法と HVite を用いて，学習に用いなかった 2 回分の 5 母音の孤立発声を認識せよ。
(7) 認識率を計算せよ。テストデータが少ないので，手計算で十分行える。

課題 4–2: CENSREC–1 を用いた日本語連続数字音声認識

CENSREC–1 データベース [3] の一部を用いて，日本語連続数字音声認識実験を行う。モデルには単語単位のモデルを用いることとし，モデル数は 1（/ichi/）, 2（/ni/）, 3（/saN/）, 4（/yoN/）, 5（/go/）, 6（/roku/）, 7（/nana/）, 8（/hachi/）, 9（/kyu/），0（/zero/, /maru/の 2 種類） と無音（/sil/）の計 12 個である（短い無音/sp/は使用しない）。

(1) 予め抽出された MFCC が `/home/data/CENSREC-1` に用意されているので，これを利用せよ。時間詳細情報なしラベル `train.mlf` と学習データファイルリスト `train.scp`，正解ラベル `test.mlf` とテストデータファイルリスト `test.scp` も，`/home/data/CENSREC-1` に用意されている。
(2) この実験では，数字（単語）単位のモデルを用いる。数字に対しては出力状態が 16 状態（状態数が 18 状態），無音に対しては出力状態が 3 状態（状態数が 5 状態）であるようなモデルの雛形を作成せよ。
(3) HCompV により 1 混合の初期モデルを学習せよ。
(4) 初期モデルをひとつのファイルに結合した後，時間詳細情報なしラベルと HERest を使って，1 混合のモデルを再学習（連結学習）せよ。
(5) HHEd と HERest を使って，モデルの混合数を 1 → 2 → 4 と増やせ。
(6) 4.2.1 節で示した連続数字認識の文法と辞書を今やろうとしていることに合わせて修正し，HVite を用いて，テストデータを認識せよ。
(7) HResults により認識率を求めよ。

[3] `http://www.slp.cs.tut.ac.jp/CENSREC/`

リスト 4.12　モデルファイル

```
~o
<STREAMINFO> 1 26
<VECSIZE> 26 <MFCC_E_D>
~v varFloor1
<VARIANCE> 26
4.759906e-01 3.153631e-01 4.607771e-01 4.720236e-01 5.106528e-01
4.783646e-01 4.211906e-01 4.349874e-01 3.798568e-01 3.106287e-01
2.982970e-01 2.986461e-01 2.999008e-02 2.130013e-02 2.428772e-02
3.038371e-02 3.276679e-02 2.890648e-02 2.923197e-02 3.263238e-02
2.802209e-02 2.417276e-02 2.348400e-02 2.128196e-02 2.961830e-03
4.596694e-03
~h "N"
<BEGINHMM>
<NUMSTATES> 5
<STATE> 2 <NUMMIXES> 1
<MEAN> 26
 4.711703e-10 -6.810081e-10  5.556827e-10 -3.020088e-10 -1.148456e-09
 1.422028e-09  9.085355e-11 -3.555424e-11 -1.787552e-10 -8.051855e-10
 1.303765e-11  3.088365e-10  1.127595e-02 -4.106049e-03 -4.763095e-03
-1.178410e-02 -1.119226e-02 -5.515445e-03 -7.645597e-03 -4.649322e-03
-6.486170e-04  7.072947e-04 -2.902465e-03 -8.740776e-03  5.427544e-03
-1.257517e-03
<VARIANCE> 26
 4.759906e+01 3.153631e+01 4.607771e+01 4.720236e+01 5.106528e+01
 4.783646e+01 4.211906e+01 4.349874e+01 3.798568e+01 3.106287e+01
 2.982970e+01 2.986461e+01 2.999008e+00 2.130013e+00 2.428772e+00
 3.038371e+00 3.276679e+00 2.890648e+00 2.923197e+00 3.263238e+00
 2.802209e+00 2.417276e+00 2.348400e+00 2.128196e+00 2.961830e-01
 4.596694e-01
<GCONST> 3.815139e+02
<STATE> 3
 （中略）
<GCONST> 3.815139e+02
<TRANSP> 5
 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
 0.000000e+00 6.000000e-01 4.000000e-01 0.000000e+00 0.000000e+00
 0.000000e+00 0.000000e+00 6.000000e-01 4.000000e-01 0.000000e+00
 0.000000e+00 0.000000e+00 0.000000e+00 6.000000e-01 4.000000e-01
 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
<ENDHMM>
~h "a"
<BEGINHMM>
<NUMSTATES> 5
<STATE> 2
<MEAN> 26
 （以下略）
```

第 5 章

形態素解析と言語モデル

5.1 形態素解析

5.1.1 MeCab を用いた形態素解析

日本語は，英語などとは異なり，単語間に空白が挿入されておらず，単語区切りが明らかではない。そのため，日本語を対象として各種の処理を行う場合には，まず基本的な処理として形態素解析が必要となることが多い。

MeCab[2] は，CRF (Conditional Random Fields)[3] を用いて形態素の連接コストや生起コストなどのパラメータ推定を行っている形態素解析器である。MeCab を用いて，形態素解析を行うためには，以下のコマンドを実行すれば良い。

```
% mecab < mecab_input_file > mecab_output_file ↵
```

演習用環境で，リスト 5.1 のようなファイルを入力すると，リスト 5.2 のような出力が得られる。リスト 5.2 の各行は，タブ文字で区切られた 2 つの列からなり，第 1 列は形態素の表層形であり，第 2 列は形態素の付加情報である。形態素の付加情報としては，品詞，品詞細分類（3 階層），活用型（どのような活用をする語か），活用形（どのような活用をしているか），原形，読みおよび発音という 9 種類の情報が付与されている [1)]。複数文を解析する必要がある場合は，リスト 5.3 のように 1 行 1 文形式のファイルを用意すれば良い。また，MeCab を用いて単語区切りを行うには，以下のコマンドを実行すれば良い。

```
% mecab -O wakati < mecab_input_file > mecab_wakati_file ↵
```

このコマンドを実行すると，リスト 5.4 のように，単語間に空白が挿入された結果が得られる。

MeCab は，言語や辞書，コーパスにできるだけ依存しないように設計されており，いくつかの異なる品詞体系の形態素解析用辞書が利用できる。そのため，MeCab を利用して形態素解析を行う時は，利用する形態素解析用辞書の品詞体系と文字コードを確認しておく必要がある。演習用環境では，MeCab の設定ファイルは `/etc/mecabrc` にあり [2)]，リスト 5.5 のような設定を含んでいる。この設定パラメータ `dicdir` から，MeCab が使う形態素解析用辞書のディレクトリが分かる。そのディレクトリに `dicrc` という名前のファイルが含まれているはずである。`dicrc` は，演習用環境ではリスト 5.6 のような内容に

[1)] 形態素付加情報は，形態素解析用辞書によって変化する。リスト 5.2 は IPADIC 品詞体系の形態素解析用辞書を用いた場合の例であり，他の形態素解析用辞書を用いた場合については各々の付属文書を参照する必要がある。

[2)] 標準的な手順にしたがって MeCab がインストールされている環境では，`mecab-config --sysconfdir` というコマンドを実行すると，`mecabrc` のディレクトリが分かるはずである。

リスト 5.1　mecab_input_file

```
英語には単語間の空白があり，日本語には単語間の空白がありません。
```

リスト 5.2　mecab_output_file

```
英語	名詞,一般,*,*,*,*,英語,エイゴ,エイゴ
に	助詞,格助詞,一般,*,*,*,に,ニ,ニ
は	助詞,係助詞,*,*,*,*,は,ハ,ワ
単語	名詞,一般,*,*,*,*,単語,タンゴ,タンゴ
間	名詞,接尾,一般,*,*,*,間,カン,カン
の	助詞,連体化,*,*,*,*,の,ノ,ノ
空白	名詞,一般,*,*,*,*,空白,クウハク,クーハク
が	助詞,格助詞,一般,*,*,*,が,ガ,ガ
あり	動詞,自立,*,*,五段・ラ行,連用形,ある,アリ,アリ
,	記号,読点,*,*,*,*,,,,,,
日本語	名詞,一般,*,*,*,*,日本語,ニホンゴ,ニホンゴ
に	助詞,格助詞,一般,*,*,*,に,ニ,ニ
は	助詞,係助詞,*,*,*,*,は,ハ,ワ
単語	名詞,一般,*,*,*,*,単語,タンゴ,タンゴ
間	名詞,接尾,一般,*,*,*,間,カン,カン
の	助詞,連体化,*,*,*,*,の,ノ,ノ
空白	名詞,一般,*,*,*,*,空白,クウハク,クーハク
が	助詞,格助詞,一般,*,*,*,が,ガ,ガ
あり	動詞,自立,*,*,五段・ラ行,連用形,ある,アリ,アリ
ませ	助動詞,*,*,*,特殊・マス,未然形,ます,マセ,マセ
ん	助動詞,*,*,*,不変化型,基本形,ん,ン,ン
。	記号,句点,*,*,*,*,。,。,。
EOS
```

リスト 5.3　複数文からなる入力ファイル

```
1つ目の文です。
2つ目の文です。
```

リスト 5.4　mecab_wakati_file

```
英語 に は 単語 間 の 空白 が あり ， 日本語 に は 単語 間 の 空白 が あり ませ ん 。
```

なっていて，このディレクトリに格納されている形態素解析用辞書の品詞体系は IPADIC 体系であり，文字コードは Unicode であることが分かる [3]。

[3] 品詞体系については，必ずしも dicrc に明記されているとは限らない。

明示的に別の形態素解析用辞書を用いて解析を行うには，-d オプションで形態素解析用辞書のディレクトリを指定する。例えば，演習用環境では，形態素解析器 Juman で使われている品詞体系の形態素解析用辞書が /var/lib/mecab/dic/juman-utf8 にインストールされているので，この辞書を用いて形態素解析を行うには，次のようにオプションを指定して実行すれば良い。

```
% mecab -d /var/lib/mecab/dic/juman-utf8 < mecab_input_file ↵
```

リスト 5.5 MeCab 設定ファイル /etc/mecabrc （抜粋）

```
dicdir = /var/lib/mecab/dic/debian
```

リスト 5.6 形態素解析用辞書設定ファイル /var/lib/mecab/dic/debian/dicrc (抜粋)

```
;
; Configuration file of IPADIC
;
cost-factor = 800
bos-feature = BOS/EOS,*,*,*,*,*,*,*,*
eval-size = 8
unk-eval-size = 4
config-charset = UTF-8
```

課題 5–1: 形態素解析

(1) 学習コーパス /home/data/KyotoWikipedia/large/kyoto-train.ja を，MeCab を使って形態素解析した結果を，kyoto-train.mrph.ja に格納せよ。

(2) テストコーパス /home/data/KyotoWikipedia/large/kyoto-test.ja を，MeCab を使って形態素解析した結果を，kyoto-test.mrph.ja に格納せよ。

5.1.2 UNIX コマンドを用いた形態素解析結果の集計

最初に，形態素数を求める方法について考える。リスト 5.2 に含まれる EOS は文区切りに対応するため，EOS を egrep コマンドを用いて取り除き，wc コマンドを用いて行数を数えると，形態素数を調べることができる。たとえば，以下のコマンドを実行すると，形態素数として 22 という結果が得られる。

```
% egrep -v '^EOS$' mecab_output_file | wc -l ↵
22
```

オプションを指定せず正規表現のみを指定して egrep コマンドを実行すると，egrep コマンドは指定した正規表現に一致する行を出力する。-v オプションを指定して egrep コマンドを実行すると，指定した正規表現に一致しない行を出力する。このコマンド例では ^EOS$ という正規表現を egrep コマンドに対して渡しているが，この正規表現には以下の 2 つの特殊文字が含まれている。

- ^ ··· 行頭に一致する。
- $ ··· 行末に一致する。

よって，^EOS$ という正規表現は，「EOS という文字列だけを含む行」に一致する [4]。なお，$ はシェルの特殊文字のため，単純に ^EOS$ という文字列を egrep コマンドの引数として指定すると，$ がシェルの特殊文字として解釈 · 置換されてしまい，egrep コマンドに $ という文字そのものを渡すことができない。そのため，上記コマンド例のように，正規表現 ^EOS$ をシングルクォート ' で囲み，$ がシェルの特殊文字として解釈されないように保護（escape）しなければならない。

次に，異なり形態素数を求める方法について考える。リスト 5.2 では，形態素の出現順に解析結果が出現している。これを，形態素の種類順に整列し，2 回以上出現している形態素を取り除いてから行数を数えると，異なり形態素数を求めることができる。まず，形態素解析結果を形態素の種類順に整列するには，sort コマンドを用いる。sort コマンドは，指定されたファイル（または標準入力）を読み込んで，整列して出力するコマンドである。整列のための比較基準（テキストとして比較するか，数値として比較するか，大文字 · 小文字を区別して比較するか，など）は，引数と環境変数 [5] を用いて指定する。ただし，日本語を含むファイルを整列する場合には，環境変数による比較基準の指定が予期しない結果をもたらすことが多いため，sort コマンドを実行する前に，以下のように環境変数の設定を変更しておく必要がある。

```
% export LC_COLLATE=C ↵
% export LC_NUMERIC=C ↵
```

なお，演習用環境は設定変更済なので，この操作は省略できる [6]。以下は，形態素解析結果から egrep コマンドを用いて文区切りを取り除き，sort コマンドを用いて結果を整列するコマンドの実行例である。

```
% egrep -v '^EOS$' mecab_output_file | sort ↵
。　　記号,句点,*,*,*,*,。,。,。
あり　　動詞,自立,*,*,五段・ラ行,連用形,ある,アリ,アリ
あり　　動詞,自立,*,*,五段・ラ行,連用形,ある,アリ,アリ
が　　助詞,格助詞,一般,*,*,*,が,ガ,ガ
（以下略）
```

このように，形態素の種類順に整列された形態素解析結果が得られる。次に，連続して出現している同一の形態素を取り除くには，uniq コマンドを使うことができる。以下のコマンドを実行して，uniq コマンドがどのように動作するかを観察してみよう。

```
% egrep -v '^EOS$' mecab_output_file | sort | uniq ↵
。　　記号,句点,*,*,*,*,。,。,。
あり　　動詞,自立,*,*,五段・ラ行,連用形,ある,アリ,アリ
が　　助詞,格助詞,一般,*,*,*,が,ガ,ガ
（以下略）
```

[4] 正規表現は，自然言語処理だけでなく文字列を処理するプログラムを記述するときに大変便利に利用できるので，習得しておくことを勧める。正規表現に関する網羅的な解説は，[4] を参照されたい。

[5] LANG や LC_ALL などのロケールを指定する環境変数が sort コマンドの動作に影響をおよぼす。予期せぬ結果が得られるだけでなく，動作速度も著しく低下する。詳細は http://d.hatena.ne.jp/ny23/20100611/p2 などを参照。

[6] 読者自身の計算機上で sort コマンドを実行する場合，端末を起動するたびにこの設定を行うのは大変面倒だから，読者のホームディレクトリ直下の .bashrc というファイルに以下の設定を書き込んでおくと良い。

```
export LC_COLLATE=C
export LC_NUMERIC=C
```

tcsh などの csh 系のシェルを使っている場合は，読者のホームディレクトリ直下の .cshrc に，以下の設定を書き込んでおくと良い。

```
setenv LC_COLLATE C
setenv LC_NUMERIC C
```

uniq コマンドを使う前の実行例では，動詞「ある」が 2 回出現していたが，uniq コマンドを使うと，動詞「ある」の冗長な出現が取り除かれていることが分かる。このように準備した上で wc コマンドを用いて行数を数えると，異なり形態素数を求めることができる。

```
% egrep -v '^EOS$' mecab_output_file | sort | uniq | wc -l ↵
14
```

次に，形態素ユニグラムとその出現頻度を，出現頻度順に出力する方法を考えよう。uniq コマンドに -c オプションを与えると，同一内容の冗長な行を削除するだけでなく，同一内容の行が出現した回数を出力させることができる。よって，sort コマンドを用いて形態素解析結果を整列してから，-c オプションを与えた uniq コマンドを用いて出現回数を求めて，更にその結果を sort コマンドで整列してやれば良い。

```
% egrep -v '^EOS$' mecab_output_file | sort | uniq -c | sort -r -n ↵
      2 単語      名詞,一般,*,*,*,*,単語,タンゴ,タンゴ
      2 空白      名詞,一般,*,*,*,*,空白,クウハク,クーハク
      2 間        名詞,接尾,一般,*,*,*,間,カン,カン
（以下略）
```

このコマンド中の 2 つ目の sort コマンドには -r オプションと -n オプションが指定されている。sort コマンドは，通常は入力された各行をテキストとして比較して整列する。-n オプションは，この通常の挙動を変更し，入力された各行を数字として比較して整列するように指定するオプションである。また，-r オプションは，降順ではなく昇順に整列するように指定するオプションであり，頻出する形態素から順に出力するように指定している。head コマンドは，入力の先頭から指定された行数だけを出力するという動作をする。したがって，先のコマンド例と head コマンドを以下のように組み合わせると，頻出する 100 形態素だけを出力することができる。

```
% egrep -v '^EOS$' mecab_output_file | sort | uniq -c | sort -r -n | head -100 ↵
```

tail コマンドは，head コマンドとは逆に，入力の末尾から指定された行数だけを出力するという動作をする。したがって，頻度の少ない形態素から順に 100 形態素を取り出すには，以下のコマンドを実行すれば良い。

```
% egrep -v '^EOS$' mecab_output_file | sort | uniq -c | sort -r -n | tail -100 ↵
```

リスト 5.2 の各行は，形態素の表層形と形態素についての付加情報からなり，タブ文字で区切られている。以下のように cut コマンドを用いると，形態素についての付加情報のみを取り出すことができる。

```
% egrep -v '^EOS$' mecab_output_file | cut -f2 ↵
名詞,一般,*,*,*,*,英語,エイゴ,エイゴ
助詞,格助詞,一般,*,*,*,に,ニ,ニ
助詞,係助詞,*,*,*,*,は,ハ,ワ
```

（以下略）

-f オプションは何列目を取り出すかを指定するオプションであり，-f2 オプションは 2 列目を取り出すことを意味する。オプションで指定しなければ空白文字が列の区切りとして用いられるが，-d オプションを用いると区切り文字を指定して列を取り出すことができる。以下のコマンドは，各形態素の品詞のみを取り出す。

```
% egrep -v '^EOS$' mecab_output_file | cut -f2 | cut -f1 -d,
名詞
助詞
助詞
（以下略）
```

課題 5–2: 形態素解析結果の集計

(1) 課題 5–1 (1) の結果 kyoto-train.mrph.ja を集計し，単語数と異なり単語数を求めよ。
(2) 課題 5–1 (1) の結果 kyoto-train.mrph.ja を集計し，コーパスに出現する全ての単語について出現頻度を求めて，出現頻度順に整列せよ。
(3) 問 (2) の結果から上位 2 万語を取り出し，kyoto-train.voc20k.ja に格納せよ。

5.1.3　連想配列を用いた形態素解析結果の集計

形態素解析結果を集計するには，連想配列（associative array）または辞書と呼ばれるデータ構造を用いたプログラミングが必要になることも多い。連想配列とは，形式的には，キーと値の 2 つ組からなる集合である。例えば，あるコーパス T に含まれる単語の出現頻度 $f(w)$ は，単語 w をキー，出現頻度 $f(w)$ を値とする連想配列と見なすことができる。

キー（単語）	値（出現頻度）
おはよう	15
こんにちわ	12
こんばんわ	8
⋮	⋮

連想配列は，各種のプログラミング言語で組み込みオブジェクト（または標準ライブラリ）として利用することができる。リスト 5.7 に，プログラミング言語として Perl を用いた場合の例を示す。2,3 行目は，入出力に Unicode を用いることを宣言している。6 行目で標準入力（またはプログラムの引数として指定されたファイル）のデータを 1 行ずつ $str に読み込み，7 行目で $str の末尾の改行文字を取り除き，8 行目で連想配列 %table にテ

キストを格納して出現頻度をインクリメントしている。11 ~ 13 行目は連想配列に格納されている全てのキーと値を出力している。

リスト 5.7 標準入力から読み込んだテキストを連想配列に格納する Perl スクリプト

```
1  use strict;
2  use utf8;
3  use open qw/ :utf8 :std /;
4
5  my %table;
6  while( my $str = <> ){
7      chomp $str;
8      $table{$str}++;
9  }
10
11 for my $key ( sort keys %table ){
12     print $key, " => ", $table{$key}, "\n";
13 }
```

リスト 5.7 の動作を実際に確かめてみよう。まず，課題 5–1 (1) の結果 kyoto-train.mrph.ja から，各形態素の表層形のみを，egrep コマンドと cut コマンドを用いて取り出す。

```
% egrep -v '^EOS$' kyoto-train.mrph.ja
  | cut -f1 > kyoto-train.word.ja ↵（実際は 1 行）
```

次に，リスト 5.7 を unigram-freq.pl という名前で保存し，以下のコマンドを実行してみよう。

```
% perl unigram-freq.pl kyoto-train.word.ja ↵
```

実際に連想配列を用いる場面では，指定されたキーに対応する値を高速に発見できることが必要である。そのため，連想配列を実現する具体的なデータ構造としては，ハッシュ表（hash table），2 分木，トライ（trie）などが用いられる。ハッシュ表は，ハッシュ値の衝突が発生しない状況では非常に高速に値を発見することができ，実装がきわめて容易であるために広く使われている。しかし，ハッシュ値の衝突が頻発するようになると，急速に性能が低下する問題がある。また，自然言語処理においては，非常に多数のキーを取り扱う必要がしばしば発生する。例えば，Web 日本語 N グラムコーパス [7] は，総単語数が 2550 億，異なり単語数も 256 万に達する非常に大規模なコーパスである。このような大規模なコーパスの統計を作成しようとする場合，空間計算量の制限によりハッシュ表は利用できず，簡潔データ構造のトライが必要となる。簡潔データ構造について，詳しくは [5] を参照されたい。このように，小規模なデータを対象としている場合は，プログラミング言語の組み込みオブジェクト（または標準ライブラリ）を使った簡易なプログラムでも良いが，大規模なデータを対象とする場合には，適切なデータ構造を選択する必要がある。

[7] http://www.gsk.or.jp/catalog/GSK2007-C/catalog.html

課題 5–3: 未知語率

(1) 課題 5–2 (3) で作成した `kyoto-train.voc20k.ja` を読み込んで，連想配列に格納するプログラムを作成せよ。
(2) 課題 5–2 (3) で作成した `kyoto-train.voc20k.ja` を語彙として，テストコーパス `kyoto-test.mrph.ja` の未知語数と異なり未知語数を求めよ。問 (1) で作成したプログラムを改造し，`kyoto-test.mrph.ja` に含まれる語が，語彙に含まれる語か否かをチェックすれば良い。

5.2　言語モデル

5.2.1　N グラムモデル

言語モデルとしては，直前の $N-1$ 個の単語に基づいて次の単語を予測する N グラムモデルが広く用いられている。N グラムモデルは，テストコーパス $T = w_1^{|T|}$ の生起確率 $P(T)$ を次式のように表す。

$$P(T) = \prod_{i=1}^{|T|} P(w_i | w_{i-N+1}^{i-1}) \tag{5.1}$$

N グラムモデルを学習する方法としては，最尤推定法（Maximum Likelihood Estimation）が最も一般的である。最尤推定法では，$N-1$ 個の単語 $w_{i-N+1}, \ldots, w_{i-1}$ の直後に単語 w_i が出現する条件付き確率 $P_{ML}(w_i | w_{i-N+1}^{i-1})$ を，次のように推定する。

$$P_{ML}(w_i | w_{i-N+1}^{i-1}) = \frac{f(w_{i-N+1}^{i})}{f(w_{i-N+1}^{i-1})} \tag{5.2}$$

ここで，$f(\boldsymbol{w})$ は単語列 $\boldsymbol{w}$ が学習コーパス L に出現した頻度である。ユニグラムモデルは，条件として用いる単語列が空という N グラムモデルであり，次のように推定する。

$$P_{ML}(w_i) = \frac{f(w_i)}{\sum_{w \in V} f(w)} = \frac{f(w_i)}{|L|} \tag{5.3}$$

ここで，V は学習コーパス L に出現する全ての単語からなる集合（語彙）である。

実際には，データスパースネス（data sparsness）とゼロ頻度問題（zero frequency problem）という 2 つの大きな問題があるため，式 (5.2) によって得られた確率をそのまま用いることは少ない。データスパースネスとは，ある単語列の学習コーパスにおける出現頻度が非常に小さい場合，その単語列について統計的に信頼できる確率を求めることが難しいという問題である。例として，単語列の出現頻度が次のような状態である時，ある単語列 $\boldsymbol{w}$ の直後に単語 w が出現する確率 $P(w|\boldsymbol{w})$ と，ある単語列 $\boldsymbol{w'}$ の直後に単語 w' が出現する確率 $P(w'|\boldsymbol{w'})$ について考える。

$$\begin{aligned} f(\boldsymbol{w}) &= 10 \\ f(\boldsymbol{w}, w) &= 3 \\ f(\boldsymbol{w'}) &= 10000 \\ f(\boldsymbol{w'}, w') &= 3000 \end{aligned}$$

最尤推定法は，単語列の出現頻度の違いを考慮せず，$P(w|\boldsymbol{w}) = P(w'|\boldsymbol{w}') = 0.3$ という確率値を与える。実際には，学習コーパスの分布と真の分布には違いがあり，真の分布において $P(w|\boldsymbol{w})$ が 0.3 という値をとる確率と，$P(w'|\boldsymbol{w}')$ が 0.3 という値をとる確率には大きな違いがある。そのため，より頑健な言語モデルを学習するには，単語列 $\boldsymbol{w}$ の出現頻度 $f(\boldsymbol{w})$ をそのまま用いる代わりに，Good-Turing 法 [6] や Witten-Bell 法 [7] を用いて出現頻度を補正（discounting）する対策が必要になる。

ゼロ頻度問題とは，ある単語列 w_{i-N+1}^{i} が学習コーパスに出現していない場合，すなわち $f(w_{i-N+1}^{i}) = 0$ である場合に，$P(w_i|w_{i-N+1}^{i-1})$ をどのように推定するかという問題である。この場合，最尤推定法では，式 (5.2) より $P_{ML}(w_i|w_{i-N+1}^{i-1}) = 0$ となる。したがって，学習コーパスに出現していない単語列がテストコーパスに出現している場合，テストコーパス全体の生起確率 $P_{ML}(T)$ は 0 になる。これは，最尤推定法を用いて学習した N グラムモデルが，テストコーパスをまったく説明できていないということを意味する。すべての可能な単語と N グラムを網羅した学習コーパスを用意することは原理的に不可能であるから，ゼロ頻度問題は本質的に避けられない問題であり，なんらかの対策が必要である。

ゼロ頻度問題を解決する方法としては，直前の $N-1$ 個の単語の代わりに $N-2$ 個の単語を条件とする確率 $P(w_i|w_{i-N+2}^{i-1})$ を用いて，確率 $P(w_i|w_{i-N+1}^{i-1})$ を推定する方法が一般的である。推定方法としては，バックオフスムージング（back-off smoothing）と補間（interpolation）という 2 つの方法がある。バックオフスムージングは，ゼロ頻度問題が発生した時に限って，より低次の N グラムを用いて確率を推定する方法であり，次のように定式化することができる。

$$P(w_i|w_{i-N+1}^{i-1}) = \begin{cases} P'(w_i|w_{i-N+1}^{i-1}) & \text{if } f(w_{i-N+1}^{i}) > 0 \\ \alpha(w_{i-N+2}^{i-1})P(w_i|w_{i-N+2}^{i-1}) & \text{otherwise} \end{cases} \tag{5.4}$$

ここで，$\alpha(w_{i-N+2}^{i-1})$ はバックオフ係数，$P'(w|\boldsymbol{w})$ は適当なディスカウンティング法を用いて推定された条件付き確率である。バックオフ係数を求める方法には，modified Kneser-Ney 法 [8] や Katz 法 [9] などの各種の方法がある。

補間は，バックオフスムージングとは異なり，ゼロ頻度問題が発生していない時も常に，より低次の N グラムを用いて確率を推定する方法であり，次のように定式化される。

$$P(w_i|w_{i-N+1}^{i-1}) = \lambda(w_{i-N+2}^{i-1})P'(w_i|w_{i-N+1}^{i-1}) + (1 - \lambda(w_{i-N+2}^{i-1}))P(w_i|w_{i-N+2}^{i-1}) \tag{5.5}$$

ここで，$\lambda(\boldsymbol{w})$ は，直前の単語列 $\boldsymbol{w}$ によって決まる補間係数である。補間係数を求める方法には，modified Kneser-Ney 法 [8] などの各種の方法がある。直前の単語列によらずに補間係数が一定である場合は，線形補間（linear interpolation）と呼ばれ，次式のように定義される。

$$P(w_i|w_{i-N+1}^{i-1}) = \sum_{j=1}^{N-1} \lambda_j P'(w_i|w_{i-j}^{i-1}) + \lambda_0 P'(w_i) \tag{5.6}$$

ここで，λ_j $(j = 0, \ldots, N-1)$ は補間係数であり，$\sum_{j=0}^{N-1} \lambda_j = 1$ を満たす。線形補間の補間係数を求める方法としては，学習コーパスともテストコーパスとも異なる開発コーパス（held-out corpus）を用いて最適化する方法が一般的である。ディスカウンティング，バックオフスムージングおよび補間についての網羅的な解説は，[10] を参照されたい。また，

参考書やマニュアルによっては，ディスカウンティングとバックオフスムージングまたは補間を区分せずに，スムージング（smoothing）としてまとめて説明している場合も多いので注意されたい。

バックオフスムージングまたは補間を用いると，出現頻度が 0 であるような N グラムについて，より低次の N グラムに基づいて生起確率を割り当てる。最終的には，ユニグラムモデルに基づいて割り当てることになるが，ユニグラムモデルにおいても，学習コーパスに出現していない語彙外語（out-of-vocabulary）のゼロ頻度問題が避けられない。この問題に対しては，語彙外語は，出現頻度が少ない語と良く似た振る舞いをすると仮定し，出現頻度が少ない語の分布を用いて語彙外語の生起確率をモデル化するという方法が良く用いられる。具体的には，学習コーパス L に出現する全ての単語 V を出現頻度順に整列し，r 番目までの単語からなる集合 V_r を考える。そして，V_r に含まれない単語の出現率を，語彙外語の出現率 β として用いる。

$$\beta = \frac{\sum_{w \in \{V \cap \overline{V_r}\}} f(w)}{\sum_{w \in \{V\}} f(w)} \tag{5.7}$$

全ての語彙外語が等確率に出現すると仮定すると，語彙外語を含む単語ユニグラムモデルは次式のように定式化できる。

$$P(w) = \begin{cases} P_{ML}(w) & \text{if } w \in V_r \\ \beta \cdot \frac{1}{u} & \text{otherwise} \end{cases} \tag{5.8}$$

ここで，u は全ての語彙外語（V_r に含まれない語）の種類数である。ただし，一般に u は不可知であるため，確率モデルとしては厳密さを欠くが，次式のように近似することも多い。

$$P(w) = \begin{cases} P_{ML}(w) & \text{if } w \in V_r \\ \beta & \text{otherwise} \end{cases} \tag{5.9}$$

課題 5–4: 単語ユニグラムモデル

(1) 学習コーパス `/home/data/KyotoWikipedia/large/kyoto-train.ja` から，式 (5.9) に基づいて 2 万語の単語ユニグラムモデルを推定せよ。課題 5–2 (2) の結果を連想配列に読み込むプログラムを作成し，その連想配列を $f(w)$ として用いて，式 (5.9) に基づいて確率を求めれば良い。

(2) 問 (1) で推定した単語ユニグラムモデルを用いて，課題 5–1 (2) で形態素解析したテストコーパス `kyoto-test.mrph.ja` の尤度を求めよ。

5.2.2　SRILM を用いた言語モデルの学習と評価

SRILM[11] は，統計的言語モデルを作成，評価するためのツールキットである。通常の N グラムモデルだけでなく，クラス言語モデル，キャッシュモデル，スキップモデルなどの言語モデルを作成，評価することができる。

リスト 5.8　ARPA 形式

```
\data\
ngram 1=161364
ngram 2=1287904
ngram 3=539057

\1-grams:
-4.433913       !               -0.6886011
 （中略）
-5.213964       alternately     -0.2661553
-6.793748       alternates      -0.1655277
 （中略）

\2-grams:
-1.456746       ! !             -0.04122325
 （中略）
-1.076524       alternate name  -0.2525124
-1.359145       alternate names -0.1006942
 （中略）

\3-grams:
-0.8986078      ! ! (
 （中略）

\end\
```

1 行 1 文形式で，かつ単語間に空白が挿入されている学習コーパスを，入力ファイルとして用意する。その入力ファイルを対象として，以下のコマンドを実行すると，単語トライグラムモデルを学習することができる。

```
% ngram-count -order 3 -text TRAIN_FILE -lm LM_FILE ↵
```

ここで，-text オプションは入力ファイルを指定するオプション，-lm オプションは出力ファイルを指定するオプションである。-order オプションは，N グラムモデルの N を指定するオプションであり，以下のように指定すると単語 5-gram モデルを学習することになる。

```
% ngram-count -order 5 -text TRAIN_FILE -lm LM_FILE ↵
```

出力ファイルには，言語モデルが ARPA 形式 (リスト 5.8) で保存されている [8]。ARPA 形式は通常のテキストファイルであり，less コマンドや lv コマンドなどで内容を確認することができる。リスト 5.8 のように，各行は，N グラムの条件付確率，N グラム，バックオフ係数の順番に並んでいる。条件付確率とバックオフ係数は対数（底は 10）をとった値である。

SRILM では，オプションによって明示的に指定しなければ，Good-Turing 法 [6] によるスムージングに基づいて言語モデルを推定する。modified Kneser-Ney 法 [8] によるディスカウンティングと補間を組み合わせて言語モデルを推定するには，次のようにオプションを指定する必要がある。

[8] ARPA 形式について詳細は，http://www.speech.sri.com/projects/srilm/manpages/ngram-format.html などを参照

```
% ngram-count -interpolate -kndiscount -text TRAIN_FILE -lm LM_FILE ↵
```

modified Kneser-Ney 法によるディスカウンティングと補間の組合わせは，機械翻訳用言語モデルとして広く使われている。

学習コーパスに出現した全ての単語を用いて言語モデルを推定するのではなく，語彙を制限する場合は，以下のように -vocab オプションを用いて語彙を格納したファイルを指定する。

```
% ngram-count -interpolate -kndiscount -text TRAIN_FILE
    -vocab VOCAB_FILE -lm LM_FILE ↵（実際は 1 行）
```

作成した言語モデルをテストするには，以下のコマンドを実行すれば良い。

```
% ngram -lm LM -ppl  TESTDATA ↵
file data/tok/kyoto-dev.en: 1166 sentences, 24309 words, 548 OOVs
0 zeroprobs, logprob= -56971.8 ppl= 192.995 ppl1= 249.863
```

課題 5–5: 単語トライグラムモデル

(1) 学習コーパス /home/data/KyotoWikipedia/large/kyoto-train.ja を，MeCab を使って単語分割した結果を，kyoto-train.tok.ja に格納せよ。
(2) テストコーパス /home/data/KyotoWikipedia/large/kyoto-test.ja を，MeCab を使って単語分割した結果を，kyoto-test.tok.ja に格納せよ。
(3) SRILM を用いて，2 万語の単語トライグラムグラムモデルを推定せよ。ディスカウンティングと補間には modified Kneser-Ney 法を，学習コーパスとしては問 (1) の結果 kyoto-train.tok.ja を，語彙としては 課題 5–2 (3) の結果 kyoto-train.voc20k.ja を用いよ。
(4) 問 (3) で作成した単語トライグラムモデルを用いて，問 (2) で単語分割したテストコーパス kyoto-test.tok.ja の尤度を求めよ。その結果を，課題 5–4 (2) の結果と比較し，単語ユニグラムモデルと単語トライグラムモデルの優劣について論じよ。

第 6 章

係り受け解析と固有表現抽出

6.1 CaboCha を用いた係り受け解析

CaboCha[12] は，SVM （Support Vector Machines）[13] に基づく日本語係り受け解析器である。言語や辞書，コーパスにできるだけ依存しないように設計されており，いくつかの異なる品詞体系の係り受け解析用辞書が利用できる。また，係り受け解析用辞書のパラメータ推定プログラムも同時に配布されており，コーパスを用意すれば，利用者自身の用途に適した係り受け解析器を構築することができる。

以下のコマンドを実行すると，入力ファイル（リスト 6.1）を係り受け解析した結果が得られる。

```
% cabocha < cabocha_input_file ↵
                    クロールで-D
                      泳いでいる-D
    <PERSON>高橋</PERSON>さんを-D
                                見た
EOS
```

オプションを指定しない CaboCha は，上のように人間にとって判別しやすい形式で出力する。この形式はプログラムから利用するには適していないため，結果をプログラムから利用する場合は，以下のように -f1 オプションを指定して出力形式を指定する必要が生じる。

```
% cabocha -f1 < cabocha_input_file > cabocha_output_file ↵
```

-f1 オプションを指定した CaboCha の出力（リスト 6.2）について，細かく見てみよう。出力には，行頭が *（アスタリスク）になっている行（1 行目，4 行目，8 行目および 12 行目）と，リスト 5.2 と良く似た形式になっている行が含まれている。行頭が *（アスタリスク）になっている行は，文節区切りと係り受け情報を表しており，リスト 5.2 と良く似た形式になっている行は，形態素と固有表現についての情報を表している。

まず，文節区切りと係り受け情報を表している行について説明する。この行は，以下のような構成になっている。

リスト 6.1　cabocha_input_file

```
クロールで泳いでいる高橋さんを見た
```

リスト 6.2　cabocha_output_file

```
1  * 0 1D 0/1 1.676636
2  クロール　　名詞,一般,*,*,*,*,クロール,クロール,クロール　0
3  で　助詞,格助詞,一般,*,*,*,で,デ,デ 0
4  * 1 2D 0/2 1.372891
5  泳い　動詞,自立,*,*,五段・ガ行,連用タ接続,泳ぐ,オヨイ,オヨイ　0
6  で　助詞,接続助詞,*,*,*,*,で,デ,デ　0
7  いる　動詞,非自立,*,*,一段,基本形,いる,イル,イル　0
8  * 2 3D 1/2 1.372891
9  高橋　名詞,固有名詞,人名,姓,*,*,高橋,タカハシ,タカハシ　B-PERSON
10 さん　名詞,接尾,人名,*,*,*,さん,サン,サン　0
11 を　助詞,格助詞,一般,*,*,*,を,ヲ,ヲ 0
12 * 3 -1D 0/1 0.000000
13 見　動詞,自立,*,*,一段,連用形,見る,ミ,ミ　0
14 た　助動詞,*,*,*,特殊・タ,基本形,た,タ,タ　0
15 EOS
```

* 文節番号 空白 係り先文節番号 係り種類 空白 主辞/機能語情報 空白 スコア

出力例の先頭の文節「クロールで」に対応する行は，リスト 6.2 の 1 行目である。この行では，先頭の文節の文節番号は 0 であり，係り先文節番号は 1 となっている。したがって，先頭の文節の係り先は「泳いでいる」であることが分かる。出力例の末尾の文節「見た」には係り先が存在しないため，係り先文節番号は −1 になっている（12 行目）。

係り種類は D または P の 1 文字である。D は通常の係り受け，P は並列の係り受けであることを表している [14]。リスト 6.2 の 1 行目では，係り種類は D となっているから，「クロールで」から「泳いでいる」に対する係り受けは通常の係り受けであることが分かる。

主辞/機能語情報は，その文節中において主辞と機能語がどの範囲かという情報を表している。リスト 6.2 の 1 行目では，0/1 という情報が付与されており，主辞が文節中で 0 番目の形態素（クロール），機能語が文節中で 1 番目の形態素（で）ということを表している。以上のように，行頭が *（アスタリスク）になっている行が文節区切りと係り受け情報を表しているから，行頭が *（アスタリスク）になっている行を 5.1.2 項の方法で数えると，文節数が求められる。

次に，形態素と固有表現についての情報を表している行について説明する。この行は，以下のような構成になっている。

形態素表層形 タブ文字 形態素付加情報 タブ文字 固有表現ラベル

第 1 列の形態素表層形と，第 2 列の形態素付加情報は，5.1.1 項 で説明した通りである。第 3 列の固有表現ラベルは，CaboCha による固有表現抽出（6.2 節）の結果を表している。

既述の通り，CaboCha は，いくつかの異なる品詞体系の係り受け解析用辞書が利用できる。そのため，CaboCha を利用して係り受け解析を行う時は，利用する係り受け解析用辞書の品詞体系を確認しておく必要がある。演習用環境では，CaboCha の設定ファイル

リスト 6.3 CaboCha 設定ファイル /etc/cabocharc (抜粋)

```
posset = IPA
parser-model  = /var/lib/cabocha/model/dep.ipa.model
chunker-model = /var/lib/cabocha/model/chunk.ipa.model

ne = 1
ne-model = /var/lib/cabocha/model/ne.ipa.model
```

は /etc/cabocharc にあり [1]，リスト 6.3 のような設定を含んでいる。設定パラメータ posset から，CaboCha が使う品詞体系が IPADIC 体系であることが分かる。また，設定パラメータ parser-model と chunker-model から，CaboCha が係り受け解析に使うモデルファイルの場所が分かる。

[1] 標準的な手順にしたがって CaboCha がインストールされている環境では，cabocha-config --sysconfdir というコマンドを実行すると，cabocharc のディレクトリが分かるはずである。

課題 6–1: 係り受け解析

(1) 学習コーパス /home/data/KyotoWikipedia/large/kyoto-train.ja を CaboCha を使って構文解析した結果を，kyoto-train.dep.ja に格納せよ。
(2) kyoto-train.dep.ja を集計し，文節数を求めよ。
(3) CaboCha による係り受け解析結果には誤りが含まれる場合がある。「クロールで泳いでいる少女を見た」「望遠鏡で泳いでいる少女を見た」という 2 つの文を構文解析し，その構文解析結果を比較せよ。

6.2 CaboCha を用いた固有表現抽出

固有表現抽出は，人名や地名などの固有名詞，日付表現，時間表現，数量表現などの固有表現を抽出する処理である。この処理は，形態素列に含まれる各形態素に対して，その形態素が固有表現に含まれるかどうか，含まれる場合はどのような種類の固有表現に含まれるかを表すチャンクラベル（chunk label）を付与するチャンキング（chunking）問題として定式化することが一般的である。チャンクラベルを付与する方法としては，SVM や CRF などの統計的機械学習による方法 [15] と，人手で作成した規則に基づく方法 [16] があるが，固有表現ラベル付与済のコーパスが利用できる現在では，統計的機械学習に基づく方法が主流である。

CaboCha は，[15] と同様に，前後 2 形態素の形態素付加情報や文字種などを素性とする SVM を用いて固有表現抽出を行う。固有表現の分類には，IREX[17] で採用された 8 種類の分類（表 6.1）を用いている。形態素が固有表現に含まれるかどうかを表す方法としては，以下の 3 つのラベルを使う IOB2 形式が用いられている [2]。

B 固有表現の先頭の形態素である。
I 固有表現に含まれる先頭以外（中間または末尾）の形態素である。
O 固有表現に含まれない形態素である。

[2] IOB2 形式は，固有表現に限らず，あるシンボル列中に含まれるチャンク（chunk）を表現するために一般的に用いることができる。例えば，形態素列を文節単位に区切る場合は，文節の先頭形態素に B，文節の中間形態素に I を付与することによって表現できる。実際に，CaboCha では，文節区切りをチャンキング問題として定式化し，各形態素に B または I のラベルを付与する SVM を学習することによって解いている。

実際には，B ラベルおよび I ラベルと，固有表現のタイプ（8 種類）を組み合わせた 17 種類のラベル（例: B-LOCATION）を用いて，固有表現の種類毎に区別してチャンキングを行う。例えば，リスト 6.2 の 2 行目の形態素「クロール」は固有表現に含まれない形態素であるから，固有表現ラベルとして O が付与されている。リスト 6.2 の実行例に含まれる固有表現は，「高橋」という人名のみである。その解析結果（9 行目）の固有表現ラベルは B-PERSON となっており，この形態素が人名の先頭形態素であることを表している。また，複数の形態素からなる固有表現の例を，リスト 6.4 に示す。リスト 6.4 では，「札幌」「駅」という 2 形態素からなる地名を表現するために，B-LOCATION と I-LOCATION という固有表現ラベルが用いられている。

表 6.1　IREX による固有表現の分類

固有表現の種類		例
ARTIFACT	固有物名	ノーベル文学賞
DATE	日付表現	五月五日
LOCATION	地名	日本，韓国
MONEY	金額表現	２０００万ドル
ORGANIZATION	組織名	社会党
PERCENT	割合表現	三割，二〇%
PERSON	人名	村山富市
TIME	時間表現	午前五時

リスト 6.4　複数形態素からなる固有表現

```
* 0 1D 1/2 0.000000
札幌    名詞,固有名詞,地域,一般,*,*,札幌,サッポロ,サッポロ    B-LOCATION
駅      名詞,接尾,地域,*,*,*,駅,エキ,エキ      I-LOCATION
に      助詞,格助詞,一般,*,*,*,に,ニ,ニ O
* 1 -1D 1/2 0.000000
到着    名詞,サ変接続,*,*,*,*,到着,トウチャク,トーチャク      O
し      動詞,自立,*,*,サ変・スル,連用形,する,シ,シ    O
た      助動詞,*,*,*,特殊・タ,基本形,た,タ,タ    O
EOS
```

課題 6–2: 固有表現抽出

(1) 課題 6–1 (1) の結果を集計し，固有表現の出現頻度を数えよ。
(2) 課題 6–1 (1) の結果を集計し，固有表現のタイプ毎の出現頻度を数えよ。
(3) リスト 6.2 に含まれる固有表現「高橋」の形態素数は 1，リスト 6.4 に含まれる固有表現「札幌駅」の形態素数は 2 である。課題 6–1 (1) の結果を集計し，固有表現のタイプ毎に，固有表現あたり形態素数の平均を求めよ。

第 7 章

文書検索

7.1 転置ファイルを用いた文書検索

文書検索のために，文書集合に対して前処理を行うことを索引付け（indexing）という。ここでは，もっとも簡単な索引の 1 つとして，転置ファイル（inverted file）を構築する手順を実際に行い，転置ファイルについての理解を深める。

転置ファイルは，文書集合中に現れる部分文字列をある基準で収集し，その部分文字列の出現位置のリストを高速に参照できるように作成したデータ構造である。部分文字列を収集する基準は，転置ファイルの用途に応じて様々であるが，ここでは，文書集合に含まれる全ての名詞を収集することにする。このような場合は特に，収集した部分文字列を索引語と呼ぶ。以下では，転置ファイルを次のような手順で構築する。

1. 文書集合に含まれる全ての文書を形態素解析する。
2. 文書集合に含まれる全ての名詞と，その名詞が出現する文書のファイル名の対を列挙する。
3. 文書集合に含まれる全ての名詞と，その名詞が出現する文書のファイル名のリストを作る。

7.1.1 文書集合の形態素解析

最初に，文書集合に含まれる全ての文書ファイルを形態素解析する方法について考える。演習用環境では，文書ファイルは /home/data/KyotoWikipedia/ja/ に配置されている。以下のコマンドを実行すると，文書ファイルの一覧を取得することができる。

```
% ls /home/data/KyotoWikipedia/ja ↵
```

ファイル数を確認するために，以下のコマンドを実行してみよう。

```
% ls /home/data/KyotoWikipedia/ja | wc -l ↵
```

ファイルは 14000 以上あるはずである。このような大量のファイルを対象として何らかの処理を行う場合，そのためのコマンドを手動で入力することは非現実的であるため，シェルスクリプトを用意する必要がある。

リスト 7.1　list.sh

```
#!/bin/sh
for f in /home/data/KyotoWikipedia/ja/*
do
  echo $f
done
```

リスト 7.1 に，シェルスクリプトの例を示す。シェルスクリプトを記述する方法としては，sh 形式と csh 形式の 2 通りがあるが，リスト 7.1 は sh 形式である [1]。リスト 7.1 の 2 行目は，/home/data/KyotoWikipedia/ja/* というパターンに一致するファイル名を 1 つずつ変数 f に代入して，3 行目から 5 行目までのループを実行する，という意味である。ファイル名のパターン /home/data/KyotoWikipedia/ja/* は，アスタリスク * という特殊文字を末尾に含んでいる。アスタリスク * は，ピリオド以外の文字で始まるファイル名とマッチする特殊文字である [2]。よって，ファイル名のパターン /home/data/KyotoWikipedia/ja/* は，ディレクトリ /home/data/KyotoWikipedia/ja/ にある全てのファイルとマッチする。したがって，リスト 7.1 の 2 行目は，ディレクトリ /home/data/KyotoWikipedia/ja/ にあるファイルの名前を 1 つずつ変数 f に代入して，3 行目から 5 行目までのループを実行するという意味である。4 行目の $f は，変数 f の値に置換される。echo コマンドは引数として指定された文字列を標準出力に出力する（表示する）コマンドであるから，4 行目は変数 f の値を標準出力に出力する（表示する）という動作をする。

したがって，リスト 7.1 は，ディレクトリ /home/data/KyotoWikipedia/ja/ にある全てのファイルのファイル名を標準出力に出力するという動作をする。実際に，リスト 7.1 を list.sh というファイル名で保存し，以下のコマンドを実行して，期待した動作をしていることを確認してみよう。

```
% sh list.sh ↵
```

[1] 実際には，ksh や zsh などもっと多くの拡張形式が存在するが，基本となるのは sh 形式（B シェルとも言う）と csh 形式（C シェルとも言う）の 2 通りである。sh 形式のシェルスクリプトの拡張子には .sh が，csh 形式のシェルスクリプトの拡張子には .csh が用いられることが一般的である。

[2] ファイル名のパターンとして用いる場合は，アスタリスク * をワイルドカードとも呼ぶ。なお，アスタリスク * は，正規表現でも特殊文字として用いられるため，正規表現とワイルドカードを混同しやすいが，両者は異なる概念であり，注意が必要である。詳しくは，http://blog.unfindable.net/archives/600 などの議論を参照されたい。

リスト 7.2　count.sh

```
#!/bin/sh
for f in /home/data/KyotoWikipedia/ja/*
do
  b=‘basename $f‘
  wc -l $f > $b.count
done
```

シェルスクリプトでは，コマンドを実行し，そのコマンドが標準出力に出力した結果を変数に取り込むことができる。その例を，リスト 7.2 の 4 行目に示す。リスト 7.2 の 2 行目は，リスト 7.1 の 2 行目とまったく同じであり，ディレクトリ /home/data/KyotoWikipedia/ja/ にあるファイルの名前を 1 つずつ変数 f に代入して，3 行目から 6 行目までのループを実行するという意味である。バッククォート ‘ は，バッククォートで囲まれた部分をコマンドとして実行し，そのコマンドが標準出力に出力した結果を文字列として取り込むという処理を行う。すなわち，4 行目は，basename $f

というコマンドの実行結果を変数 b に代入するという意味である。$f は，変数 f の値に置換される。また，basename コマンドは，引数として与えられたファイル名からディレクトリ部分を削除して，残った部分のみを出力するコマンドである。basename コマンドの動作を理解するため，以下のコマンドを実行してみよう。

```
% basename /etc/hosts ↵
hosts
```

このように，basename コマンドは，引数として与えられた完全パス名 /etc/hosts から，ディレクトリ部分 /etc/ を削除し，残った部分 hosts のみを出力するコマンドである。よって，リスト 7.2 の 4 行目は，変数 f に代入されているファイル名からディレクトリ部分を削除し，残った部分のみを変数 b に代入するという意味である。

したがって，リスト 7.2 は，ディレクトリ /home/data/KyotoWikipedia/ja/ にある全てのファイルについて行数を数え，その結果をカレントディレクトリに .count という拡張子のファイルで保存するという動作をする。実際に，リスト 7.2 を count.sh というファイル名で保存し，以下のコマンドを実行して，期待した動作をしていることを確認してみよう。

```
% sh count.sh ↵
```

シェルスクリプトについてより詳しくは，[18] を参照されたい。

課題 7–1: 文書集合の形態素解析

ディレクトリ /home/data/KyotoWikipedia/ja/ にある全ての文書ファイルを，MeCab を用いて形態素解析せよ。リスト 7.2 を改造して，wc コマンドを呼び出す代わりに，MeCab を呼び出して，形態素解析結果を適当なファイルに保存するようにすれば良い。

7.1.2　索引語と文書ファイル名の対の列挙

次に，形態素解析結果から名詞のみを取り出し，以下のような形式で出力するプログラムを用意する。

名詞 1　タブ文字　ファイル名
名詞 2　タブ文字　ファイル名
名詞 3　タブ文字　ファイル名
（略）

リスト 7.3 に，プログラミング言語として Perl を用いた場合の例を示す。2,3 行目は，入出力に Unicode を用いることを宣言している。5 行目は，プログラムの引数を，変数 $file に代入している。6 行目は，プログラムの引数として指定されたファイルを開いて，1 行ずつ変数 $str に読み込む（7 行目）。8 行目では $str の末尾の改行文字を取り

リスト 7.3　extract-noun.pl

```
1  use strict;
2  use utf8;
3  use open qw/ :utf8 :std /;
4
5  my $file = $ARGV[0];
6  open( my $fh, '<:utf8', $file ) or die $!;
7  while( my $str = <$fh> ){
8      chomp $str;
9      my( $x, $y ) = split( /\t/, $str, 2 );
10     if( $y =~ /^名詞,/ ){
11         print $x, "\t", $file, "\n";
12     }
13 }
```

除き，9 行目では $str の値をタブを区切りとして 2 つに分割して，前半（表層形）を $x に，後半（形態素の付加情報）を $y に代入する。10 行目は，変数 $y の値が「^名詞,」という正規表現に一致していたら，11 行目を実行して，「名詞 タブ文字 ファイル名」という形式で出力する。これを，ファイルを全て読み込み終わるまで繰り返す（7 ~ 13 行目の while ループ）。

実際に，リスト 7.3 を extract-noun.pl というファイル名で保存し，以下のコマンドを実行して，期待した通りの動作をしているかを確認してみよう。

```
% mecab /home/data/KyotoWikipedia/ja/BLD00001.ja > BLD00001.mrph.ja ↵
% perl extract-noun.pl BLD00001.mrph.ja ↵
日本    BLD00001.mrph.ja
城郭    BLD00001.mrph.ja
（略）
```

課題 7–2: 索引語と文書ファイル名の対の列挙

文書集合に含まれる全ての名詞と，その名詞が出現する文書のファイル名の対を列挙し，kyoto.noun-did というファイルに格納せよ。課題 7–1 の結果として得られた全ての形態素解析結果ファイルを対象として，extract-noun.pl を呼び出すシェルスクリプトを，リスト 7.2 を改造して作成すれば良い。

7.1.3　索引語と文書ファイル名のリストの作成

課題 7–2 の結果として得られた索引語と文書ファイル名の対を列挙したファイル kyoto.noun-did には，以下のように文書ファイル毎に対が固まっているはずである。

名詞 1　タブ文字　ファイル名 1
名詞 2　タブ文字　ファイル名 1
名詞 1　タブ文字　ファイル名 2
名詞 2　タブ文字　ファイル名 2
（略）

これを sort コマンドを用いて並べ替え，索引語毎に対が固まるようにする。

```
% sort kyoto.noun-did > kyoto.sorted-noun-did
```

このコマンドの結果 kyoto.sorted-noun-did には，以下のように索引語毎に対が固まっているはずである。

名詞 1　タブ文字　ファイル名 1
名詞 1　タブ文字　ファイル名 2
名詞 2　タブ文字　ファイル名 1
名詞 2　タブ文字　ファイル名 2
（略）

上記のように整列された索引語とファイル名の対を読み込んで，以下のような転置ファイルを出力するプログラムを作成する。

名詞 1　タブ文字　ファイル名 1　空白　ファイル名 2 ···
名詞 2　タブ文字　ファイル名 1　空白　ファイル名 2 ···
（略）

リスト 7.4 に，プログラミング言語として Perl を用いた場合の例を示す。2,3 行目は，入出力に Unicode を用いることを宣言している。6 行目で標準入力（またはプログラムの引数として指定されたファイル）のデータを 1 行ずつ $str に読み込み，7 行目で $str の末尾の改行文字を取り除く。9 行目では $str の値をタブを区切りとして 2 つに分割して，前半（索引語）を $x に，後半（ファイル名）を $y に代入する。10 ~ 12 行目では，直前の行の索引語 $pre と現在の行の索引語 $x が一致している間は，出力せずに配列 @docid にファイル名を蓄積し続ける。直前の行の索引語 $pre と現在の行の索引語 $x が変化していたら，索引語とファイル名のリストを出力（13 ~ 15 行目）し，直前の行の索引語 $pre とファイル名のリスト @docid を初期化する（16 ~ 17 行目）。

実際に，リスト 7.4 を unify-noun.pl というファイル名で保存し，以下のコマンドを実行して，期待した通りの動作をしているかを確認してみよう。

```
% perl unify-noun.pl kyoto.sorted-noun-did > kyoto.inv-file ↵
```

リスト 7.4　unify-noun.pl

```
use strict;
use utf8;
use open qw/ :utf8 :std /;

my $pre = '';
my @docid;
while( my $str = <> ){
    chomp $str;
    my( $x, $y ) = split( /\t/, $str, 2 );
    if( $pre eq $x ){
        push( @docid, $y );
    } else {
        if( @docid ){
            print $pre, "\t", join( ' ', @docid ), "\n";
        }
        $pre = $x;
        @docid = ( $y );
    }
}

if( @docid ){
    print $pre, "\t", join( ' ', @docid ), "\n";
}
```

課題 7–3: 転置ファイルを用いた文書検索

(1) 構築した転置ファイルを用いて，「新田」「田辺」という 2 つの名詞がともに出現する記事を列挙せよ。

(2) 構築した転置ファイルを用いて，「新田辺」という駅名が出現する記事を列挙せよ。

(3) 構築した転置ファイルを用いて，文書集合に含まれる全ての名詞について $IDF(w)$ を求めよ。$IDF(w)$ の降順に整列し，上位 10 語を列挙せよ。$IDF(w)$ については，教科書 4.2.2 項（1）を参照せよ。

7.2　Minise を用いた文書検索

Minise は，転置索引，N グラム索引および接尾辞配列が利用できる全文検索エンジンである。索引付けを行う minise_build コマンドと，作成した索引を用いて検索を行う minise_search コマンドの 2 つからなる。

まず，索引付けを行う minise_build コマンドの使い方について述べる。minise_build コマンドを使うには，最初に，索引付けの対象となるファイルの一覧を用意する必要がある。以下は，find コマンドを用いて，/home/data/KyotoWikipedia/ja/ 以下の全ての通常のファイルを列挙し，列挙した結果を document.list というファイルに格納するコマンドである。

```
% find /home/data/KyotoWikipedia/ja/ -type f > document.list ↵
```

find コマンドは，指定されたディレクトリ以下を調べて，指定された条件に合致するファイルやディレクトリの名前を標準出力に出力する（表示する）コマンドである。この例では，-type f というオプションによって，「通常のファイルである」という条件を指定している。そのため，このコマンドは /home/data/KyotoWikipedia/ja/ 以下の全てのファイルを列挙するという意味となっている。索引付けの対象とするファイルを制限したい場合は，find コマンドに条件を追加[3]したり，find コマンドの出力を grep コマンドで加工したりすれば良い。

[3] 例えば，-type f というオプションに加えて，-name '*.txt' というオプションを加えれば，'*.txt' というファイル名パターンに一致する通常のファイルのみが列挙される。

次に，列挙したファイルを対象として，索引付けを行う。以下は，minise_build コマンドを用いて，document.list にファイル名が格納されている全てのファイル（文書集合）を対象として索引付けを行うコマンドである。

```
% minise_build -m 2gram -l document.list -i 2gram.index ↵
```

オプションの意味は以下の通りである。

-m 2gram
: 索引付けの方式として，2 グラム索引を指定するオプション。

-l document.list
: 索引付けの対象ファイル一覧が格納されているファイルを指定するオプション。

-i 2gram.index
: 索引付けの結果である索引ファイルを指定するオプション。

-m オプションは索引付け方式を指定するオプションであり，以下のような方式を指定することができる。

seq	逐次検索
inv	空白または改行で区切られた単語を検索語として収集する転置ファイル
1gram	全ての文字ユニグラムを検索語として収集する転置ファイル
2gram	全ての文字バイグラムを検索語として収集する転置ファイル
sa	接尾辞配列
sa8	（Unicode を仮定した）接尾辞配列

作成した索引ファイルを用いて，検索を行うためのコマンドは以下の通りである。

```
% minise_search -i 2gram.index ↵
```

minise_search コマンドの起動後に，検索語を入力するためのプロンプトが出現するので，そこで適当な語を入力すれば良い。コマンドラインから検索語を指定する必要がある場合は，以下のようにコマンドを実行すれば良い。

```
% echo サンプル | minise_search -i 2gram.index > result.txt ↵
```

これは，索引ファイル 2gram.index から「サンプル」という語を検索した結果を，

`result.txt` というファイルに格納するコマンドである。

課題 7–4: 文字ユニグラム索引を用いた文書検索

(1) ディレクトリ `/home/data/KyotoWikipedia/ja/` にある全ての文書ファイルを対象として，Minise を用いて文字ユニグラム索引を作成せよ。
(2) 問 (1) で構築した文字ユニグラム索引を用いて，「新田」「田辺」という 2 つの文字列がともに出現する記事を列挙し，課題 7–3 (1) の結果と比較せよ。
(3) 問 (1) で構築した文字ユニグラム索引を用いて，「新田辺」という文字列が出現する記事を列挙し，課題 7–3 (2) の結果と比較せよ。
(4) 問 (2) および問 (3) の結果を踏まえて，ユニグラム索引を用いた文書検索と，名詞を索引語とする転置ファイルを用いた文書検索という，2 つの方式の得失について述べよ。

第 8 章

統計的機械翻訳

この章では，教科書 6.3 節に対応する演習として，統計的機械翻訳を実際に試してみる手順について述べる。翻訳モデルおよび単語アラインメントモデルを作成するツールキットとしては，GIZA++ を用いる。統計的機械翻訳のツールキットとしては，Moses を用いる。Moses は，フレーズおよび構文木に基づく統計的機械翻訳モデルの学習と実験を行うために必要な各種のツールを備えている。

8.1 対訳コーパスの前処理

最初に，Moses および GIZA++ から利用する対訳コーパスを用意する。対訳コーパスは，原言語（英語）の文と，その文に対応する対訳文（日本語）を，それぞれ別のファイルに，対訳文対が同じ行になるように格納されている必要がある。また，単語に関する前処理も必要になる。

Wikipedia 日英京都関連文書対訳コーパス [1] は，日本語と英語の対訳文対を XML 形式で格納しているため，これをリスト 8.1 とリスト 8.2 のように分離する処理が必要になる。演習用環境では，京都フリー翻訳タスク [2] の付属スクリプトを用いて分離したファイルを /home/data/KyotoWikipedia/ に用意しているので，これを用いる。

単語に関する前処理は英語と日本語で異なる。英語の場合は，トークン化（tokenization）と小文字化という 2 つの前処理が必要である。この 2 つの前処理は，Moses に付属しているスクリプト [3] を用いて行うことができる。トークン化とは，つながっている単語を分離する処理であり，文中のカンマや括弧，文末のピリオドなどを独立したトークンとして分離する。小規模学習コーパス /home/data/KyotoWikipedia/small/kyoto-train.en をトークン化して，kyoto-train.tok.en に格納するコマンドは以下の通りである [4]。

```
% tokenizer.perl -l en
  < /home/data/KyotoWikipedia/small/kyoto-train.en
  > kyoto-train.tok.en ↵（実際は 1 行）
```

次に，大文字で始まっている語と，小文字で始まっている語を同一視するため，コーパスに出現する全ての文字を小文字化する。以下は，トークン化された小規模学習コーパス kyoto-train.tok.en を小文字化して，kyoto-train.low.en に格納するコマンドで

[1] http://alaginrc.nict.go.jp/WikiCorpus/

[2] http://www.phontron.com/kftt/

[3] 演習用環境では，Moses 付属のスクリプト類は /usr/lib/moses/scripts/ 以下に配置されている。また，Moses 付属のスクリプトを，絶対パスを指定することなく呼び出すことができるよう環境設定済である。読者自身の計算機に Moses をインストールして使う場合は，適切に読み替える必要がある。

[4] 本テキストでは，仮想マシンから利用できるメモリと演習時間の制限を回避するため，非常に小規模な対訳コーパスを用いる演習手順を示す。メモリと演習時間に余裕がある読者は，/home/data/KyotoWikipedia/large/ に格納されているコーパスを用いて演習を試みてほしい。

リスト 8.1　kyoto-test.ja（抜粋）

```
1 奈良線(ならせん)は，京都府木津川市の木津駅(京都府)から京都府京都市下京区の京
      都駅に至る西日本旅客鉄道(JR西日本)の鉄道路線(幹線)である。
2 全線が大都市近郊区間に含まれる。
3 関西本線の支線としての沿革を持つため，正式な起点は木津駅だが，列車運行上は京
      都から木津へ向かう列車が下り(列車番号は奇数)，逆が上り(同偶数)となってい
      る。
```

リスト 8.2　kyoto-test.en（抜粋）

```
1 The JR Nara Line is a railway line (arterial line) of the West Japan Railway
      Company (JR West) that runs between Kizu Station (Kyoto Prefecture) in
      Kizugawa City, Kyoto Prefecture, and Kyoto Station in the Shimogyo Ward
      of Kyoto City, Kyoto Prefecture.
2 The entire rail line is included in the section covering the metropolitan
      area and its suburbs.
3 Although the Nara Line officially starts at Kizu Station because it is
      historically a branch line of the Kansai Main Line, outbound trains (odd
      -numbered trains) run from Kyoto to Kizu and inbound trains (even-
      numbered trains) run in the opposite direction.
```

ある。

```
% lowercase.perl < kyoto-train.tok.en > kyoto-train.low.en ↵
```

課題 8–1: 英語コーパスの前処理

(1) 小規模テストコーパス /home/data/KyotoWikipedia/small/kyoto-test.en の前処理を行い，結果を kyoto-test.low.en に格納せよ。

(2) 以下のように 2 つの小規模開発コーパスを結合し，結果を kyoto-tunedev.en に格納せよ。

```
% cat
  /home/data/KyotoWikipedia/small/kyoto-tune.en
  /home/data/KyotoWikipedia/small/kyoto-dev.en
  > kyoto-tunedev.en ↵（実際は 1 行）
```

(3) 結合した小規模開発コーパス kyoto-tunedev.en の前処理を行い，結果を kyoto-tunedev.low.en に格納せよ。

日本語の場合は，単語分割を行い，単語間にスペースを挿入する必要がある。以下に，小規模学習コーパス /home/data/KyotoWikipedia/small/kyoto-train.ja の単語間に，MeCab を用いてスペースを挿入するコマンドを示す。

```
% mecab -O wakati < /home/data/KyotoWikipedia/small/kyoto-train.ja
  > kyoto-train.tok.ja ↵（実際は 1 行）
```

日本語文では，大文字 · 小文字の使い分けはないはずだが，念のために小文字化も行って

おく。

```
% lowercase.perl < kyoto-train.tok.ja > kyoto-train.low.ja ↵
```

なお，日本語コーパスにおいては，英数字や記号に，半角文字と全角文字が混在していることがある。演習用環境のコーパスについては，全角文字を全て半角文字に統一済であるが，他のコーパスを用いる場合は，付録 A を参考に統一する必要がある。

課題 8–2: 日本語コーパスの前処理

(1) 小規模テストコーパス /home/data/KyotoWikipedia/small/kyoto-test.ja の前処理を行い，結果を kyoto-test.low.ja に格納せよ。
(2) 以下のように 2 つの小規模開発コーパスを結合し，結果を kyoto-tunedev.ja に格納せよ。

```
% cat
  /home/data/KyotoWikipedia/small/kyoto-tune.ja
  /home/data/KyotoWikipedia/small/kyoto-dev.ja
  > kyoto-tunedev.ja ↵（実際は 1 行）
```

(3) 結合した小規模開発コーパス kyoto-tunedev.ja の前処理を行い，結果を kyoto-tunedev.low.ja に格納せよ。
(4) 比較的大規模な学習コーパス /home/data/KyotoWikipedia/large/kyoto-train.ja の前処理を行い，結果を kyoto-largetrain.low.ja に格納せよ。前処理中に発生する中間ファイル名が衝突しないように注意すること。

学習用対訳コーパスについては，GIZA++ の制限を回避するため，長すぎる文や短すぎる文，空行，冗長なスペースを削除するなどの整形処理も必要である。そのため，Moses 付属のスクリプト clean-corpus-n.perl を以下のように実行する。

```
% clean-corpus-n.perl kyoto-train.low ja en kyoto-train.cln 1 40 ↵
```

このコマンドは，前処理済のコーパス kyoto-train.low.ja と kyoto-train.low.en を整形処理した結果を，kyoto-train.cln.ja と kyoto-train.cln.en に出力する。

8.2 目的言語の言語モデルの作成

統計的機械翻訳を，教科書 6.3.2 項は，次のように定式化している。

$$\hat{e} = \arg\max_{e} P(f|e)P(e)$$

ここで，$P(f|e)$ は翻訳モデル（translation model），$P(e)$ は言語モデル（language model）である。このように，統計的機械翻訳では，翻訳モデルだけでなく，目的言語の言語モデルが必要になる。そのため，前処理済の比較的大規模な学習コーパス

kyoto-largetrain.low.ja から，modified Kneser-Ney 法によるディスカウンティングと補間を用いた単語 5 グラムモデルを作成するコマンドを，以下のように実行する。

```
% ngram-count -order 5 -interpolate -kndiscount
  -text kyoto-largetrain.low.ja
  -lm kyoto-largetrain.lm.ja ↵（実際は 1 行）
```

コマンドのオプションについての詳細は，本テキストの 5.2.2 項を参照。

8.3　フレーズ翻訳モデルの作成

Moses は，翻訳モデル $P(f|e)$ としてフレーズ翻訳モデル [5] を用いる統計的機械翻訳ツールキットである。Moses によるフレーズ翻訳モデルの作成手順は，以下の通りである。

[5] 教科書 6.3.3 項を参照。

1. 単語の数値化，クラスタリング，共起単語表の作成などの前処理。
2. GIZA++ を用いて IBM モデル [6] を推定し，最尤な単語アラインメントを得る。英日・日英の両方向で行う。
3. 両方向の単語アラインメントから，ヒューリスティックスを用いて対称な単語アラインメントを得る。
4. 対称な単語アラインメントに矛盾しないフレーズ対応を得る。
5. フレーズ対応を集計し，フレーズ翻訳確率を求める。
6. 並べ替えモデル（reordering model）を作成する。

[6] 教科書 6.3.2 項を参照。

これらの手順は，Moses に付属する train-model.perl というスクリプトで一度に実行可能である。small というディレクトリを作成し，そのディレクトリ以下にフレーズ翻訳モデルを作成するためのコマンドを，以下に示す。

```
% mkdir small ↵
% train-model.perl
  --root-dir small
  --corpus kyoto-train.cln
  --f en
  --e ja
  --alignment grow-diag-final-and
  --reordering msd-bidirectional-fe
  --lm 0:5:‘pwd‘/kyoto-largetrain.lm.ja:0 ↵（実際は 1 行）
```

スクリプト train-model.perl のオプションの意味は以下の通りである。

--root-dir small
: フレーズ翻訳モデルを保存するディレクトリを指定するオプション。

--corpus kyoto-train.cln
: 学習用対訳コーパスのプレフィックスを指定するオプション。

`--f en`
: 原言語の学習用コーパスのサフィックスを指定するオプション。この例では，`--corpus` オプションと合わせて，原言語の学習用コーパスとして，`kyoto-train.cln.en` というファイルを用いるという意味である。

`--e ja`
: 目的言語の学習用対訳コーパスのサフィックスを指定するオプション。この例では，`--corpus` オプションと合わせて，目的言語の学習用コーパスとして，`kyoto-train.cln.ja` というファイルを用いるという意味である。

`--alignment grow-diag-final-and`
: 対称な単語アラインメントを得るヒューリスティックスを指定するオプション。

`--reordering msd-bidirectional-fe`
: 並べ替えモデルの種類を指定するオプション。

`--lm 0:5:‘pwd‘/kyoto-largetrain.lm.ja:0`
: 目的言語の言語モデルの仕様とファイル名を指定するオプション。ファイル名は絶対パスで指定しなければならない。この例は，pwd コマンドの結果に置き換えられる ‘`pwd`‘ という記法 [7] を用いて，カレントディレクトリ直下の `kyoto-largetrain.lm.ja` というファイルを使うように指定している。

[7] pwd コマンドは，カレントディレクトリ名を標準出力に出力する（表示する）コマンドである。この記法について詳しくは，リスト 7.2 のバッククォートについての説明を参照。

以下では，フレーズ翻訳モデルを構成するファイルを確認する。

8.3.1 方向がある単語アラインメント

上述の通り，Moses は，対訳文対からフレーズ対を獲得するための第 1 段階として，単語単位のアラインメントを必要とする。GIZA++ は，IBM モデル [8] に基づいて，最尤な単語単位のアラインメントを求めるツールキットである。GIZA++ による英日方向の単語アラインメント結果は，`small/giza.en-ja/en-ja.A3.final.gz` に格納されている。以下のコマンドを実行して，単語アラインメント結果を確認してみよう。

[8] 教科書 6.3.2 項を参照。

```
% lv small/giza.en-ja/en-ja.A3.final.gz ↵
```

ある対訳文対の単語アラインメントの例を，リスト 8.3 に示す。2 行目は，英文である。3 行目は，日本語文と単語アラインメント結果を表している。3 行目の各日本語単語の右にある丸括弧と中括弧に挟まれた数字が，その日本語単語に対応する英単語の位置を表す。例えば，「**待合室** ({ 6 7 8 })」は，「待合室」という単語が，英文の 6 番目，7 番目および 8 番目の単語（a waiting room）に対応していることを意味している。英日方向では，すべての英単語がちょうどひとつの日本語単語に対応するので，英単語に対応しない日本語単語もあれば，複数の英単語に対応する日本語単語もありえる。日英方向では，これが逆となる。

リスト 8.3　英日方向の単語アラインメント例

```
# Sentence pair (12143) source length 7 target length 9 alignment score : 2.05918e-15
the platform is equipped with a waiting room .
NULL ({ }) ホーム ({ 2 }) 上 ({ }) に ({ 1 }) 待合室 ({ 6 7 8 }) を ({ }) 持つ ({ 3 4 5 }) 。 ({ 9 })
```

課題 8–3: 日英方向の単語アラインメント

(1) 以下のコマンドを実行して，日英方向の単語アラインメント結果を確認せよ。

```
% lv small/giza.ja-en/ja-en.A3.final.gz
```

(2) リスト 8.3 と同じ対訳文対について，日英方向の単語アラインメント結果を調べ，英日方向の単語アラインメントとの違いについて述べよ。

8.3.2　対称な単語アラインメント

GIZA++ による単語アラインメントは，単語アラインメントが付与されていない対訳文対を入力として，辞書などの対訳リソースをまったく参照せずに求めているが，かなり精度が高い。しかし，明らかに誤っているアラインメントも含まれている上に，方向がある単語アラインメントという考え方自体が自然ではない。そのため，英日方向および日英方向の単語アラインメントを参照し，ヒューリスティックスを用いて，双方向に 1 対多対応を考慮する対称な単語アラインメントを求める。

基本的な手順は，英日方向および日英方向の単語アラインメントの積集合から出発して，和集合に含まれる単語アラインメントをヒューリスティックスによって選択 · 追加し，積集合と和集合の中間的な集合を求めるという手順である。ここで，英日方向および日英方向の単語アラインメントの積集合とは，両方向ともに存在する単語アラインメントからなる集合であり，和集合とは，少なくとも片方向に存在する単語アラインメントからなる集合である。

どのようなヒューリスティックスを用いるかは，8.3 節でも説明したように，Moses の付属スクリプト `train-model.perl` の `--alignment` オプションによって指定する。以下のヒューリスティックスが利用可能である [9]。

- `intersect`
- `union`
- `grow`
- `grow-final`
- `grow-diag`
- `grow-diag-final`
- `grow-diag-final-and`
- `srctotgt`
- `tgttosrc`

[9] 詳しくは，`http://www.statmt.org/moses/?n=FactoredTraining.AlignWords` を参照。

求められた対称な単語アラインメントは, small/model/aligned.grow-diag-final-and というファイルに格納されている。リスト 8.3 において例示した対訳文対

- the platform is equipped with a waiting room .
- ホーム 上 に 待合室 を 持つ 。

に対応する対称な単語アラインメントを，リスト 8.4 に示す。マイナス記号 – でつながれた数字のペアをアラインメント・ポイントと呼び，対称な単語アラインメントを表している。マイナス記号 – の左側の数字が原言語の単語位置，右側の数字が目的言語の単語位置である。ただし，方向がある単語アラインメントの場合とは異なり，左端の単語の位置は 0 から始まる。よって，1-0 は，英語の ‘platform’ と日本語の「ホーム」の単語アラインメントを意味している。

リスト 8.4　対称な単語アラインメント例

```
1-0 4-2 5-3 6-3 7-3 3-4 2-5 3-5 4-5 8-6
```

課題 8–4: 対称な単語アラインメント

リスト 8.3 およびリスト 8.4 とは異なる適当な 1 つの対訳文対について，英日方向の単語アラインメント，日英方向の単語アラインメント，対称な単語アラインメントの 3 つを比較せよ。

8.3.3　フレーズテーブル

対称な単語アラインメントから，フレーズペアとしての条件 [10] を満たす全てのフレーズペアを抽出した結果が，small/model/phrase-table.gz に格納されている。以下のコマンドを実行して，フレーズテーブルの抽出結果を確認してみよう。

```
% lv small/model/phrase-table.gz ↵
```

フレーズテーブルの一部をリスト 8.5 に示す。リスト 8.5 の各行は，||| で区切られて，左から順に以下の 5 つの情報が記述されている。

- 英語フレーズ
- 日本語フレーズ
- フレーズ翻訳スコア
- フレーズ内単語対応
- デバッグ用情報（フレーズ出現頻度）

さらに，フレーズ翻訳スコアは，以下の 5 つのスコアからなっている [11]。

（a）逆向きフレーズ翻訳確率 $\varphi(f|e)$
（b）逆向き単語翻訳確率 $lex(f|e)$

[10] 教科書の図 6.6「フレーズペア抽出結果の例」に関する説明を参照。

[11] 詳しくは, http://www.statmt.org/moses/?n=FactoredTraining.ScorePhrases を参照。

（c）フレーズ翻訳確率 $\varphi(e|f)$
（d）単語翻訳確率 $lex(e|f)$
（e）フレーズペナルティ

どのようなフレーズ翻訳スコアを用いるかについては，Moses の付属スクリプト train-model.perl の --score-options オプションによって変更することができる。例えば，逆向き単語翻訳確率および単語翻訳確率を使わない場合は，以下のようにオプションを指定する。

```
% train-model.perl （他のオプション） --score-options '--NoLex' ↵
```

リスト 8.5　フレーズテーブル（抜粋）

```
1 ticket gates are ||| 改札 口 ||| 0.0357143 0.0133661 0.2 0.0809927 2.718 ||| 0-0 1-1 2-1 ||| 140 25 5
2 ticket gates are ||| 改札 口 が ||| 0.0526316 0.00041515 0.04 0.00473862 2.718 ||| 0-0 0-1 1-1 ||| 19 25 1
3 ticket gates are ||| 改札 口 が ある ||| 0.25 0.00540811 0.04 3.74377e-05 2.718 ||| 0-0 1-1 2-1 2-2 2-3 ||| 4 25 1
4 ticket gates are ||| 改札 口 は ||| 0.129032 0.00699206 0.32 0.0204659 2.718 ||| 0-0 1-1 2-1 1-2 ||| 62 25 8
5 ticket gates are ||| 改札 機 ||| 0.0181818 0.00106971 0.08 0.188595 2.718 ||| 0-0 1-1 ||| 110 25 2
6 ticket gates are ||| 改札 機 による ||| 0.0571429 0.00106971 0.08 0.000200665 2.718 ||| 0-0 1-1 ||| 35 25 2
7 ticket gates are ||| 改札 機 による 対応 ||| 0.0571429 0.00106971 0.08 2.37226e-07 2.718 ||| 0-0 1-1 ||| 35 25 2
8 ticket gates are ||| 改札 機 も ||| 0.333333 0.00255195 0.04 0.00245096 2.718 ||| 0-0 1-1 2-2 ||| 3 25 1
```

課題 8–5: フレーズテーブルの確認

“coupon ticket” または “coupon tickets” を含むフレーズについて，フレーズテーブルの内容を確認せよ。

8.3.4　並べ替えモデル

ある原言語から目的言語にフレーズ翻訳モデルを用いて翻訳するとき，原言語におけるフレーズの出現順序と目的言語におけるフレーズの出現順序は必ずしも一致せず，並べ替えが必要になる。並べ替えモデルとは，そのようなフレーズの並べ替えについてのモデルである。並べ替えモデルについて，詳しくは Moses のウェブサイト [12] を参照されたい。

[12] http://www.statmt.org/moses/?n=FactoredTraining.BuildReorderingModel

8.4　設定ファイルの編集

フレーズ翻訳モデルの作成が完了した時点で， small/model/moses.ini に，このフレーズ翻訳モデルを用いて翻訳を行うための設定ファイルが保存されている。リスト 8.6 に，設定ファイルの例を示す。設定ファイルは，Moses に対して以下の内容を指定している。

- 利用するモデルの仕様 [13]
- 利用するモデル・ファイルの場所
- 各モデルのモデルパラメータに対する重み
- デコーダの各種パラメータ（スタックサイズ, 考慮するフレーズ数, 並び替えの範囲）

[13] 例えば，目的言語の N グラムモデルの N の値など。

実際に統計的機械翻訳を行うには，翻訳モデルのパラメータ調整（8.5 節）が重要である。パラメータ調整の段階では，デコーダ（Moses）を繰り返し実行する必要が生じるため，デコーダの実行時間が重要である。リスト 8.6 の 29 行目（[ttable-limit] の次行）は，1 つの原言語のフレーズに対して考慮する目的言語のフレーズ数を制限する設定である。リスト 8.6 は，20 個までのフレーズを考慮するように指定しているが，この値を 10 程度まで減らすとデコーダはかなり高速に実行できるようになる。次に，リスト 8.6 の 64 行目（[distortion-limit] の次行）は，フレーズ並べ替えの範囲の制限を表している。欧米系言語間の翻訳の場合は 6 で十分なことが多いようだが，英日翻訳の場合は無制限にしたほうがよい結果が得られる。無制限に設定するには，64 行目を –1 と書き換える。

課題 8–6: moses.ini の修正

上記の説明にしたがって，[ttable-limit] と [distortion-limit] の設定を修正せよ。

8.5　フレーズ翻訳モデルのパラメータ調整

8.3 節で作成したフレーズ翻訳モデルは，以下のような複数のパラメータを持つ。

言語モデル　目的言語の生成確率。パラメータは 1 つ。
フレーズテーブル　フレーズの翻訳確率。パラメータは 5 つ。
並べ替えモデル　フレーズの並び替え確率。パラメータは 7 つ。
ワードペナルティ　目的言語の長さに関するペナルティ。パラメータは 1 つ。

合計 14 個と多数のパラメータがあるため，Minimum Error Rate Training （MERT）という手法を用いて自動的に調整することが一般的である。

MERT は，評価関数（通常は BLEU）を最大にするような翻訳結果が選ばれるように，パラメータを調整する。翻訳を試行するための開発コーパスを Moses に入力し，各文について上位 100 個程度の翻訳候補を出力し，評価関数の観点からみてより良い翻訳候補がより上位に出現するようにパラメータを調整する。この手順を，パラメータが収束するまで 10 ~ 20 回程度繰り返す。

Moses には，MERT を実行するためのスクリプトが付属しており，以下のようにコマンドを実行すると MERT によるパラメータの最適化ができる。

```
% mkdir tuning ↵
% mert-moses.pl
  kyoto-tunedev.low.en
  kyoto-tunedev.low.ja
  /usr/lib/moses/bin/moses
  ./small/model/moses.ini
  --working-dir `pwd`/tuning ↵（実際は 1 行）
```

リスト 8.6　moses.ini

```
#########################
### MOSES CONFIG FILE ###
#########################

# input factors
[input-factors]
0

# mapping steps
[mapping]
0 T 0

# translation tables: table type (hierarchical(0), textual (0), binary (1)), source-factors, target-factors, number of scores, file
# OLD FORMAT is still handled for back-compatibility
# OLD FORMAT translation tables: source-factors, target-factors, number of scores, file
# OLD FORMAT a binary table type (1) is assumed
[ttable-file]
0 0 0 5 /home/exercise/small/model/phrase-table.gz

# no generation models, no generation-file section

# language models: type(srilm/irstlm), factors, order, file
[lmodel-file]
0 0 5 /home/exercise/kyoto-train.lm.ja

# limit on how many phrase translations e for each phrase f are loaded
# 0 = all elements loaded
[ttable-limit]
20

# distortion (reordering) files
[distortion-file]
0-0 wbe-msd-bidirectional-fe-allff 6 /home/exercise/small/model/reordering-table.wbe-msd-bidirectional-fe.gz

# distortion (reordering) weight
[weight-d]
0.3
0.3
0.3
0.3
0.3
0.3
0.3

# language model weights
[weight-l]
0.5000

# translation model weights
[weight-t]
0.20
0.20
0.20
0.20
0.20

# no generation models, no weight-generation section

# word penalty
[weight-w]
-1

[distortion-limit]
6
```

オプションの意味は以下の通りである。

`kyoto-tunedev.low.en`
: 第 1 引数として，原言語の開発コーパスを指定する。

`kyoto-tunedev.low.ja`
: 第 2 引数として，目的言語の開発コーパスを指定する。

`/usr/lib/moses/bin/moses`
: 第 3 引数として，デコーダ（Moses）の実行ファイルを指定する。

`./small/model/moses.ini`
: 第 4 引数として，デコーダ（Moses）の設定ファイルを指定する。これは，MERT によるパラメータの最適化対象となるフレーズ翻訳モデルの場所や仕様を指定することを意味する。

`--working-dir ‘pwd‘/tuning`
: 途中経過ファイルを保存するディレクトリを指定するオプション。ディレクトリは絶対パスで指定しなければならない。この例は，pwd コマンドの結果に置き換えられる ‘pwd‘ という記法 [14] を用いて，カレントディレクトリ直下の tuning というディレクトリに保存するよう指定している。

[14] pwd コマンドは, カレントディレクトリ名を標準出力に出力する（表示する）コマンドである。この記法について詳しくは, リスト 7.2 のバッククォートについての説明を参照。

先にのべた通り，MERT は，各繰り返し毎に開発コーパスに対する BLEU 値が大きくなるようにパラメータを調整する。mert-moses.pl は，各繰り返しにおいて得られたパラメータと BLEU 値を，--working-dir オプションで指定されたディレクトリに，run#.mert.log[15] というファイル名で保存する。リスト 8.7 に，そのファイルの例を示す。11 行目に，この繰り返しにおいて推定された 14 個のパラメータと，目的関数（BLEU）の値が記入されている。

[15] # の部分は, 繰り返し番号で置き換えられる

リスト 8.7　run1.mert.log

```
1  shard_size = 0 shard_count = 0
2  Seeding random numbers with system clock
3  name: case value: true
4  Data::m_score_type BLEU
5  Data::Scorer type from Scorer: BLEU
6  Loading Data from: run1.scores.dat and run1.features.dat
7  loading feature data from run1.features.dat
8  loading score data from run1.scores.dat
9  Data loaded : [Wall 1.42955 CPU 1.42809] seconds.
10 Creating a pool of 1 threads
11 Best point: 0.056171 0.056171 0.056171 0.056171 0.0843707 0.056171 0.0932191
       0.0936183 -0.269745 0.0374473 0.0374473 0.0284031 0.0374473 0.0374473
       => 0.178183
12 Stopping... : [Wall 113.215 CPU 113.215] seconds.
```

課題 8–7: MERT による BLEU 値の変化

以下のコマンドを実行して，MERT による BLEU 値の増加を確認せよ。

```
% egrep '^Best point:' tuning/run*.mert.log ↵
```

MERT の結果として，--working-dir オプションで指定したディレクトリに moses.ini というファイルが作成される。このファイルが，MERT 実行後の最終的な設定ファイルである。一部を抜き出すと以下のようになる。

リスト 8.8　MERT の結果（tuning/moses.ini の抜粋）

```
# language model weights
[weight-l]
0.0989463

# translation model weights
[weight-t]
0.0175748
0.0738663
0.0707554
0.0192307
0.00543955
```

各パラメータは，正の大きな数値であればあるほど重要視されていることを意味する。リスト 8.8 では，言語モデルのパラメータは約 0.1（3 行目），フレーズテーブルの 4 つのパラメータは約 0.02 ~ 0.07（7 ~ 10 行目），フレーズペナルティは約 0.005（11 行目）である [16]。言語モデルのパラメータとフレーズテーブルのパラメータは大体同程度であるが，フレーズテーブルの 4 つのパラメータを合計すると，言語モデルよりもかなりフレーズテーブルが重要視されていることが分かる。

[16] MERT は確率的な処理を含むため，必ずしもリスト 8.8 と演習結果は一致しない。

8.6　翻訳実験と自動評価

MERT によるパラメータ調整を行った設定ファイル tuning/moses.ini を用いて，テストコーパス kyoto-test.low.en を翻訳し， kyoto-test.trans.ja に格納するコマンドを以下に示す。

```
% moses -config ./tuning/moses.ini -input-file kyoto-test.low.en
  > kyoto-test.trans.ja ↵（実際は 1 行）
```

上記の例では，目的言語が日本語であるため，最終結果の後処理は特に行っていない。しかし，目的言語が英語である場合は，大文字 · 小文字を元に戻す処理（recasing）が必要になる。Moses には，recasing の確率モデルを学習し，そのモデルを用いて recasing するプログラムが付属している。

BLEU 値を計算するには，Moses 付属のスクリプト multi-bleu.perl を用いて，以下のようにコマンドを実行する。

```
% multi-bleu.perl kyoto-test.low.ja < kyoto-test.trans.ja ↵
```

このコマンドは，目的言語のテストコーパス kyoto-test.low.ja と翻訳結果 kyoto-test.trans.ja を比較して，BLEU 値の計算を行う。

課題 8–8: 翻訳実験と自動評価

(1) MERT によるパラメータ調整を行う前の設定ファイル small/model/moses.ini を用いて，テストコーパス kyoto-test.low.en を翻訳し，BLEU 値を求めよ。
(2) MERT によるパラメータ調整を行った設定ファイル tuning/moses.ini を用いて，テストコーパス kyoto-test.low.en を翻訳し，BLEU 値を求めよ。
(3) 問 (1) と問 (2) の翻訳結果を観察し，BLEU 値の違いが，人間の目から見た翻訳品質の違いを反映しているか検討せよ。

参考文献

[1] Steve Young, Dan Kershaw, Julian Odell, Dave Ollason, Valtcho Valtchev, and Phil Woodland. *The HTK Book*, 2000. http://htk.eng.cam.ac.uk/docs/docs.shtml.

[2] 工藤拓, 山本薫, 松本裕治. Conditional Random Fields を用いた日本語形態素解析. 情報処理学会研究報告, 第 2004-NL-161 巻, pp. 89–96, 2004.

[3] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of ICML*, pp. 282–289, 2001.

[4] Jeffrey E.F. Friedl. 詳説 正規表現 第 3 版. オライリージャパン, 2008.

[5] 岡野原大輔. 高速文字列解析の世界. 岩波書店, 2012.

[6] I. J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, Vol. 40, No. 3/4, pp. 237–264, 1953.

[7] I. H. Witten and T. C. Bell. The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression. In *IEEE Transactions on Information Theory*, Vol. 37, pp. 1085–1094, Jul 1991.

[8] Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, ACL '96, pp. 310–318, 1996.

[9] S. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. 35, No. 3, pp. 400–401, 1987.

[10] 北研二. 確率的言語モデル, 言語と計算, 第 4 巻. 東京大学出版会, 1999.

[11] A. Stolcke. SRILM - an extensible language modeling toolkit. In *Proceedings of International Conference of Spoken Language Processing*, Denver, Colorado, September 2002.

[12] 工藤拓, 松本裕治. チャンキングの段階適用による日本語係り受け解析. Vol. 43, No. 6, pp. 1834–1842, 2002.

[13] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[14] 黒橋禎夫, 居蔵由衣子, 坂口昌子. 形態素・構文タグ付きコーパス作成の作業基準, 2000. http://nlp.ist.i.kyoto-u.ac.jp/nl-resource/corpus/KyotoCorpus4.0/doc/syn_guideline.pdf.

[15] 山田寛康, 工藤拓, 松本裕治. Support Vector Machine を用いた日本語固有表現抽出.

情報処理学会論文誌, Vol. 43, No. 1, pp. 44–53, Jan 2002.

[16] 桝井文人, 鈴木伸哉, 福本淳一. テキスト処理のための固有表現抽出ツール next の開発. 言語処理学会第 8 回年次大会発表論文集, pp. 176–179, 2002.

[17] Satoshi Sekine and Yoshio Eriguchi. Japanese named entity extraction evaluation: analysis of results. In *Proceedings of the 18th conference on Computational linguistics*, pp. 1106–1110, Morristown, NJ, USA, 2000. Association for Computational Linguistics.

[18] 山森丈範. シェルスクリプト基本リファレンス. 技術評論社, 2011.

付録 A

文字コードの扱い

日本語を対象として言語処理を行う場合，データの文字コードには常に注意が必要である。日本語の文字コードとしては，Unicode や Shift-JIS, EUC-JP, ISO-2022-JP などがある。以前は EUC-JP または Shift-JIS を用いることも多かったが，現在では Unicode を用いることが一般的である。

現在では，ほとんどのツールは Unicode のファイルを処理できる。Unicode を処理できない古いツールや Unicode 以外の文字コードで保存されたデータを扱わなければならない場合は，iconv コマンドを使って文字コードを変換する必要がある。例えば，Unicode のファイル input_file を EUC-JP に変換して output_file に保存するには，以下のコマンドを実行すれば良い。

```
$ iconv -f UTF-8 -t EUC-JP input_file > output_file
```

逆に，EUC-JP のファイル input_file を Unicode に変換して output_file に保存するには，以下のコマンドを実行すれば良い。

```
$ iconv -f EUC-JP -t UTF-8 input_file > output_file
```

入力ファイルの文字コードが不明の場合には，入力ファイルの文字コードを自動的に推定する機能がある nkf コマンドが便利である。例えば，以下のコマンドを実行すると，入力ファイル input_file の文字コードを自動的に推定し，推定された文字コードから Unicode への変換を行う。

```
$ nkf -w input_file > output_file
```

ただし，文字コードの自動推定は誤る場合があるため，自動推定に頼りすぎるべきではない。できるだけ，入力ファイルの文字コードを明示的に指定するべきである。

また，アルファベットや数字については，「ＡＢＣ０１２」のように全角文字で表現されている場合と，「ABC012」のように半角文字で表現されている場合とがある。2 つの表現形式が混在しているとツールやプログラムが混乱する危険性があるため，適切な正規化が必要である。標準的な方法としては，Unicode 正規化法のひとつである NFKC 正規化がある [1]。標準入力から読み込んだテキストを，NFKC 正規化を行って標準出力に出力する Perl スクリプトをリスト A.1 に示す。

[1] Unicode 正規化については，http://unicode.org/reports/tr15/ や http://homepage1.nifty.com/nomenclator/unicode/normalization.htm を参照。

リスト A.1　NFKC 正規化を行う Perl スクリプト

```
use Unicode::Normalize;
use strict;
use utf8;
use open qw/ :utf8 :std /;

while(<>){
    print Unicode::Normalize::NFKC($_);
}
```

付録 B

研究用環境の構築

筆者らが用意した仮想マシンイメージ [1] にインストールされている音声処理および言語処理用の各種フリーソフトウェアは，全て Debian パッケージの形式に整理されている。仮想マシンイメージと同等の計算機環境を構築するには，以下の手順を行えば良い。

[1] http://www.coronasha.co.jp/support/slp-and-nlp

（1）読者の計算機に Debian GNU/Linux 6.0[2] をインストールする。

[2] http://www.debian.org/

（2）リスト B.1 の設定を /etc/apt/sources.list に追加する。

（3）以下のコマンドを実行する。

```
% sudo apt-get update ↵
% sudo apt-get install corona-exercise ↵
```

リスト B.1　sources.list の追加設定

```
deb http://nlp.imc.tut.ac.jp/~tsuchiya/debian squeeze-tutimcnlp main
```

「音声言語処理と自然言語処理」演習

発行　2013 年 4 月 16 日
著者　土屋 雅稔，山本 一公
URL　http://www.coronasha.co.jp/support/slp-and-nlp