第 62 回 東京エリア Debian 勉強会資料

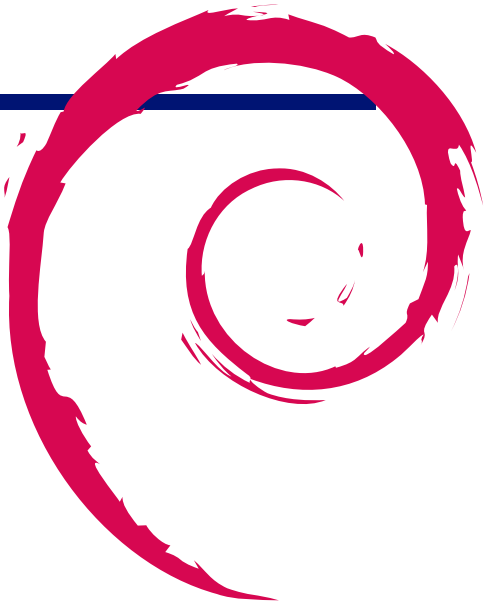
月刊 Deb専

日本唯一のDebian専門月刊誌

2010 年 3 月 20 日

特集1: ニューラルネットワーク特集

特集2: man-db にはまってみた



1 Introduction

上川 純一

今月の Debian 勉強会へようこそ。これから Debian の世界にあしを踏み入れるという方も、すでにどっぷりとつかっているという方も、月に一回 Debian について語りませんか?

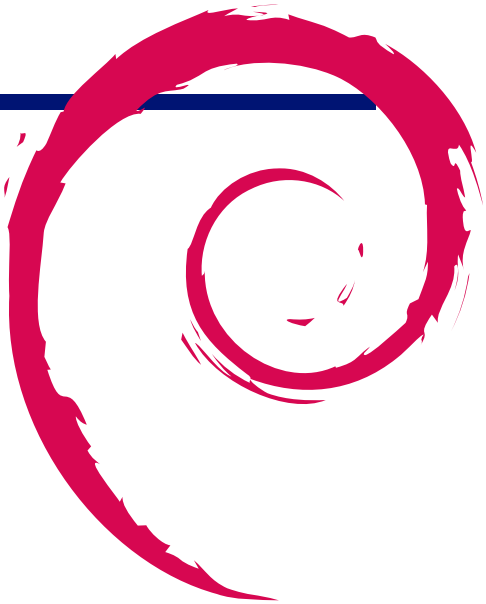
Debian 勉強会の目的は下記です。

- Debian Developer (開発者) の育成。
- 日本語での「開発に関する情報」を整理してまとめ、アップデートする。
- 場の提供。
  - 普段ばらばらな場所にいる人々が face-to-face で出会える場を提供する。
  - Debian のためになることを語る場を提供する。
  - Debian について語る場を提供する。

Debian の勉強会ということで究極的には参加者全員が Debian Package をがりがりと作るスーパーハッカーになった姿を妄想しています。情報の共有・活用を通して Debian の今後の能動的な展開への土台として、「場」としての空間を提供するのが目的です。

デビアン勉強会

目次



2 事前課題

上川 純一

今回の事前課題は以下です:

1. 好きな日本語 Man ページ
2. ニューラルネットワークで解決できる問題

この課題に対して提出いただいた内容は以下です。

2.1 日比野 啓

好きな日本語 man ページは accept(2) です。 Linus の socklen_t と BSD ソケット層についてのコメントが引用されているのが興味深いです。

2.2 岩松 信洋

1。 好きな日本語 Man ページ日本語だと情報古いからみんな嫌いよ! 2。 ニューラルネットワークで解決できる問題

2.3 吉野

2.3.1 好きな日本語 Man ページ

sed(1) の一番最後でしょうか．関連項目の regex(5) [うーん、書かないとダメですねえ] がいつも気になります．ただ，原文では既に削除されています．

2.4 本庄

2.4.1 好きな日本語 Man ページ

ぼくは man head が大好きです。特にありません。

2.4.2 ニューラルネットワークで解決できる問題

イアン・エアーズ著『その数学が戦略を決める』という本にニューラルネットを活用した事例がいくつか載っていました。ひとつ引用します。

> たとえばアリゾナ大学の研究者たちは、ツーソン・グレイハウンド公園でのグレイハウンド・ドッグレースで勝ち犬を予測するニューラル・ネットワークを構築した。 (中略) 最高の予想屋でも 60 ドル d すっていたが、ニューラル・ネットワークは 125 ドル儲けた。

お金が儲かります。

2.5 henrich

1. 好きなページは特になく、原文との差に心を痛めています。結構手作業が発生しそうで、とても何かの合間にやるというのも現実的ではない様子。スポンサーがいて作業するならいいんだけど…

2.6 akedon

2.6.1 好きな日本語 Man ページ

ED, マニュアルページの DESCRIPTION がチュートリアルっぽい点が気に入っています。昔、 emacs どころか vi も無かった頃のエディタとして基本でした。 TCPDUMP, IPTABLES のマニュアルページもネットワークの勉強ではお世話になりました。(なっています?)

2.7 mkouhei

- 考えたことはとくにありませんでしたが、あえて挙げるなら bash ですね。 3172 行って、どんだけ膨大なんだか。
- 家計 (特に光熱費) の分析に使ってみて、取り合えずどんなものかをまず見てみようと思います。

2.8 yasuhiro

- なんでしょうね．．まずは ls ですかね．あんな豊富なのをよく書いて訳すよなあ，というかんじ．つぎは man 7 ipv6 っですかね．
- ニューラルネットワークっていちおうなんでもできるんじゃないのすかね．

2.9 koedoyoshida

2.9.1 好きな日本語 Man ページ

openssh-jman `http://www.unixuser.org/~euske/doc/openssh/jman/`

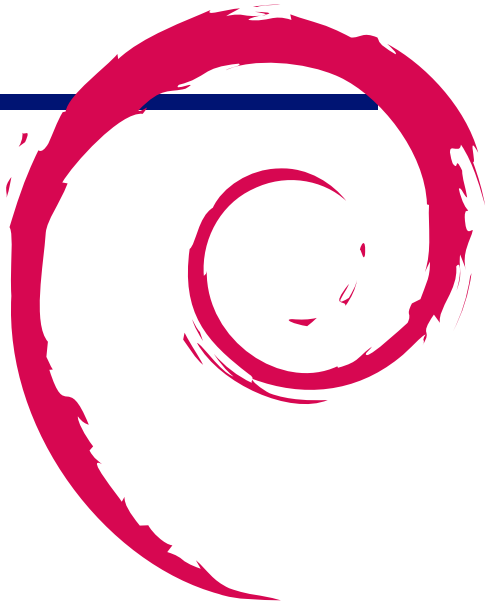
基本的に Debian はサーバ使用なので ssh のマニュアルはよく見ます。 Debian には日本語マニュアルはパッケージになっていないようなので、ローカルにインストールしたり、 Web ページを見ることが多いです。過去に deb パッケージも作っていたのですが、結構頻繁に更新されるので結局上記になりました。

2.10 なかお

好きな日本語 man ページ： section 7 url

2.11 上川純一

好きな日本語の Man ページはありません。英語版を更新しても日本語版が更新されないという仕組みがよくない気がしています。



3 最近の Debian 関連のミーティング報告

前田耕平

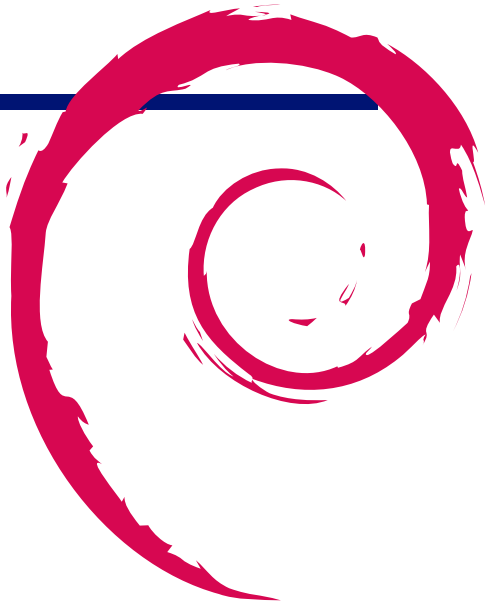
3.1 東京エリア Debian 勉強会 61 回目報告

今月の Debian 勉強会は木更津高専の教室とその近所の公民館をお借りして 1 泊 2 日で開催しました。木更津側で会場の準備、参加者の募集をしてくださった坂口さん、大枝先生、宿泊手続きを中心に幹事サポートをしてくださったやまねさん、プレゼンターの皆さん (やまねさん、日比野さん、坂口さん、上川さん)、そして参加していただいた皆さんのおかげで、無事成功裏に終えることができたと思います。皆様ありがとうございました。

セッションの時間配分や宿と会場との移動が忙しなくなってしまった点など、タイムスケジュールに問題があったのは私の至らぬ点が原因ですので、反省点として次回以降に活かしたいと思います。

勉強会で行った内容については、事前配布資料や発表資料、参加者のブログにも掲載されると思いますのでここでは特に触れませんが、今回の勉強会の表向きのテーマは「関数型言語」「温泉」でしたが、開催してみて気づいた真のテーマは「縁」でした。やまねさんが事前課題回答、セッションで Debian を使う理由に「縁」を挙げていましたが、まさに「縁」がきっかけで開催でき、今回の勉強会をきっかけに新たな「縁」が出来たなと思います。 Debian 勉強会を通じた縁以外に、関数型言語やニューラルネットワークといった興味のある話題を通じた縁、 debian-users の ML を通じた縁、Debian と Ubuntu というディストリビューションを通じた縁、一昨年の Debian 温泉を通じた縁、などなど色々ありますが、特に昨年秋の OSC で「東京都心だけでなく木更津などの郊外でも開催してほしい」と坂口さんが要望を出し、現地の幹事として実際に動いてくださった、というのが一番大きな縁じゃないかと思います。

今週末には OSC 2010 Tokyo/Spring があり、そこで東京エリア Debian 勉強会もブースとセッションを出すので、今回参加された皆さんはもちろん、参加出来なかった方も、参加したことが無かったけど興味が出てきた方も、ぜひ足を運んでいただければと思います。「縁」を深め、また新たな「縁」ができるとよいですね。



4 ニューラルネットワークで画像認識してみた

本庄弘典

4.1 はじめに

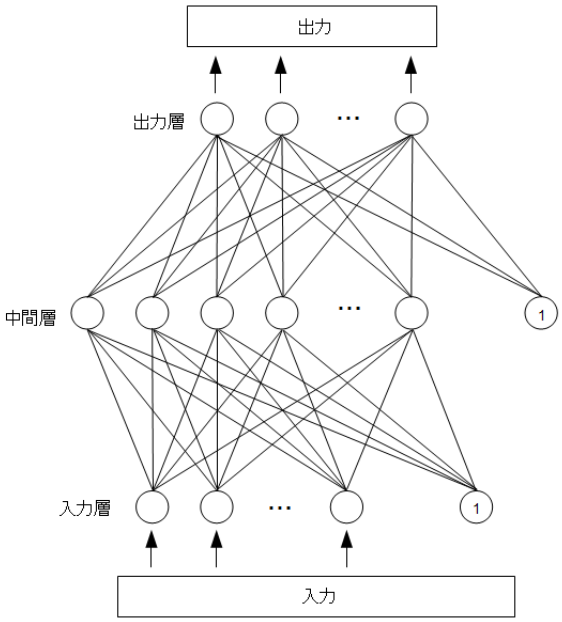
ドキュメントスキャナで本をスキャンした際、画像のサイズが大きすぎるため保存に適しません。この画像を 2 値画像とグレースケール、カラー画像それぞれの処理を加えることでファイルサイズを縮小し、ニューラルネットを用いることによりある程度自動化できないかと考えました。今回はニューラルネットとして一般的な三層パーセプトロンを用いた画像判別の一例を解説します。

4.2 三層パーセプトロンとバックプロパゲーション

4.2.1 三層パーセプトロン

三層パーセプトロンは入力層、中間層、出力層と別れた三層の各ニューロンが重みと呼ばれる係数で結ばれたモデルとなります。

図 1 三層パーセプトロン



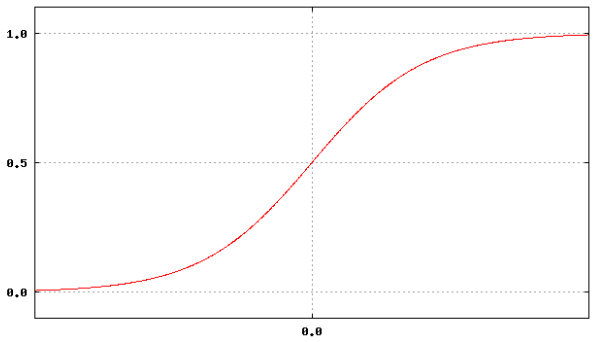
それぞれの重みは実数で表され、パーセプトロンが機能するためにはこの重みが適切に設定されている必要があります。

ある入力が与えられた際、入力値に重みを掛け合わせ、それぞれの合計に次のようなシグモイド関数を適用した数値を中間層の持つ値とします。

図 2　シグモイド関数の式

$$\varsigma_1(x) = \frac{1}{1+e^{-x}}$$

図 3　シグモイド関数のグラフ



各出力層も同様の計算がなされ、パーセプトロンの出力が行われます。

4.2.2　バックプロパゲーション

多層パーセプトロンで適切な出力を行うための学習方法として一般的なものにバックプロパゲーションがあります。バックプロパゲーションではまず入力に対する正しい出力 (教師信号) を多数用意し、各重みをランダムに設定します。用意された入力に対してランダムな重みからパーセプトロンの出力はでたらめな値となりますが、この出力と教師信号との比較から出力層と中間層の間の重みを修正し、次いで中間層と入力層の重みを修正することで適切な重みを探し出します。

4.3　足し算と引き算を学習してみる

作成したパーセプトロンとバックプロパゲーションが正常に動作するかを確かめます。次のような入力を用意しました。

```
# 学習用教師信号ペア
0.40,0.20       0.60,0.20
0.30,0.20       0.50,0.10
0.80,0.10       0.90,0.70
0.20,0.10       0.30,0.10
0.50,0.50       1.00,0.00
0.60,0.20       0.80,0.40
# 評価用入力値
*0.50,0.10
*0.50,0.40
*0.10,0.40
```

入力値と教師信号のペアはタブ区切りの左が入力、右が入力に対する教師信号です。ここでは足し算と引き算の教師信号を与えました。

実行します。

```
$ ./backprop.exe sample.txt 10000
       0 0.87640153
     100 0.26410368
     200 0.10289131
     300 0.03820243
     400 0.02475167

...(中略)...

    9600 0.00077714
    9700 0.00077174
    9800 0.00076646
    9900 0.00076128
0.4000, 0.2000  0.60, 0.18      0.60, 0.20
0.3000, 0.2000  0.50, 0.11      0.50, 0.10
0.8000, 0.1000  0.90, 0.70      0.90, 0.70
0.2000, 0.1000  0.30, 0.11      0.30, 0.10
0.5000, 0.5000  0.98, 0.02      1.00, 0.00
0.6000, 0.2000  0.80, 0.41      0.80, 0.40
0.5000, 0.1000  0.63, 0.35
0.5000, 0.4000  0.93, 0.06
0.1000, 0.4000  0.87, 0.00
Ratio=0.00075626
Count=10000
Sample=6
Input=2
Middle=4
Output=2
InputHidden0=-2.57936471,-2.20525001,-1.50656422,4.05055823,-0.66468037
InputHidden1=-1.29032439,8.71632107,-1.24344376,-0.85214732,-0.66468037
InputHidden2=2.04901840,-2.94096519,1.04866634,-1.98825291,0.29698485
HiddenOutput0=-2.91458436,-1.16992032
HiddenOutput1=5.84673832,-6.31188860
HiddenOutput2=-1.80018561,-0.42470539
HiddenOutput3=3.60356071,3.84028669
HiddenOutput4=1.40998866,-1.22885398
```

頼りないながらもそれなりの演算結果が出力されています。評価として最後の数値は減算結果が負になるはずなのですが、シグモイド関数を通すことで出力が 0.0〜1.0 となるため正常な結果が得られません。

4.4 画像を分類するための入力値を考える

画像判別の入力値として次の値を使用しました。

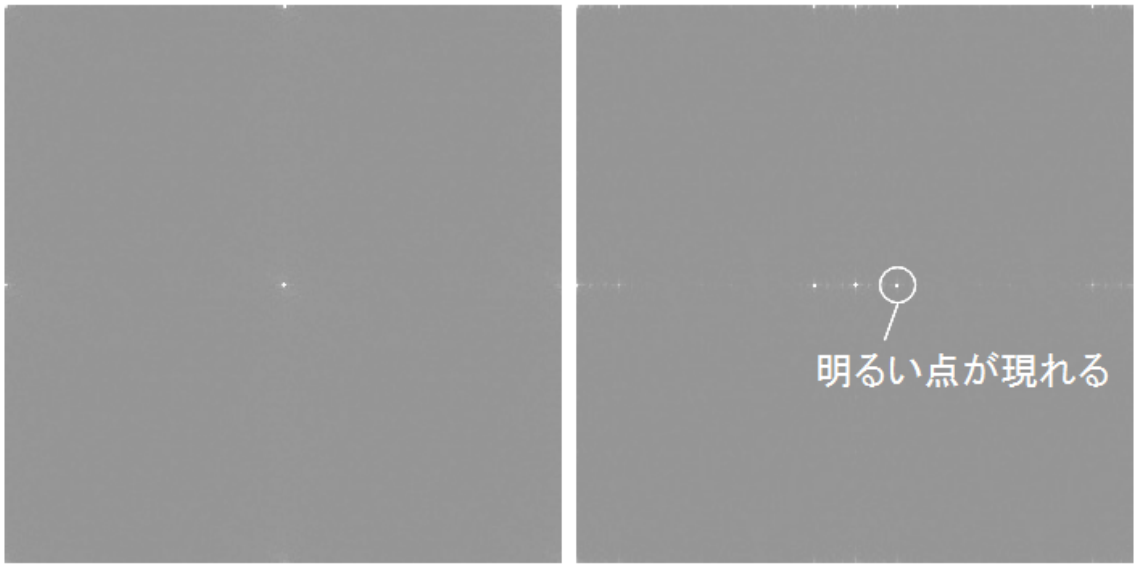
- 補正した画像の RGB の差
- 微分した画像の RGB の差
- 画像の複雑さ。
- 使われている色の数
- 平均彩度
- FFT 処理した画像の明るいピクセルを利用する
- HSV に変換し、色素の平均を利用する
- 色素の分散を利用する

この中から文章と絵の判別として画像の FFT を、カラー画像の判別として HSV への変換を解説します。

4.4.1 モノクロ画像の処理・文字と絵を分類してみる

縦書きの文章は横方向に一定の周波数を持っていると見なすことが出来ます。これにより、文章の画像を微分し FFT 処理を行った結果から振幅を描画するとで、明るく光る点が現れることがわかりました。

図 4　イラストと文章を微分した画像の FFT 結果



この点の明るさを入力値とすることで、文章とイラストの判別が行えると期待できます。

4.4.2　カラー画像とそうでない画像を分類してみる

カラー画像とモノクロ画像は画像の RGB を HSV に変換し、色相から判別を行っています。
RGB のうちから最大のものを MAX、最小のものを MIN とすると色相は次の式となります。

図 5　色相の計算式

$$
\begin{aligned}
H = & 60\frac{G-B}{MAX-MIN}+0, && if MAX=R \\
& 60\frac{B-R}{MAX-MIN}+120, && if MAX=G \\
& 60\frac{R-G}{MAX-MIN}+240, && if MAX=B
\end{aligned}
$$

モノクロ画像は色相を持たないため、 RGB のうち青の成分を減らすことで黄色いフィルタをかけました。こうすることでモノクロ画像の色相の平均は黄色となり、カラーとモノクロを判別するための入力値として期待できます。

4.5　学習の条件

ニューラルネットの学習は次の条件で行いました。

- 入力層 8 個
- 中間層 24 個
- 出力層 2 個
- サンプルとして使用した本 23 冊 (漫画 2 冊/文庫 20 冊/技術書 1 冊)
- ページ数 6742 ページ
- 学習回数 50 万回

実際にこの条件で学習を行った際、 Core i7 950 で 7 時間弱の学習時間となりました。

4.6 判別の精度

作成されたツールで実際に判別を行い、その精度を調べました。評価に使用した本は学習に使われていないものを選びました。

サンプル 1. **ライトノベル最新巻**

240 ページ中、人間の判別と食い違うページが 2 ページ。内訳はカラー 12 ページ、グレースケール 15 ページ、文章 213 ページ。そのうちイラストが文章と判別されたのが 1 ページ、文章がイラストと判別されたのが 1 ページ。

サンプル 2. SF **長編シリーズの上巻**

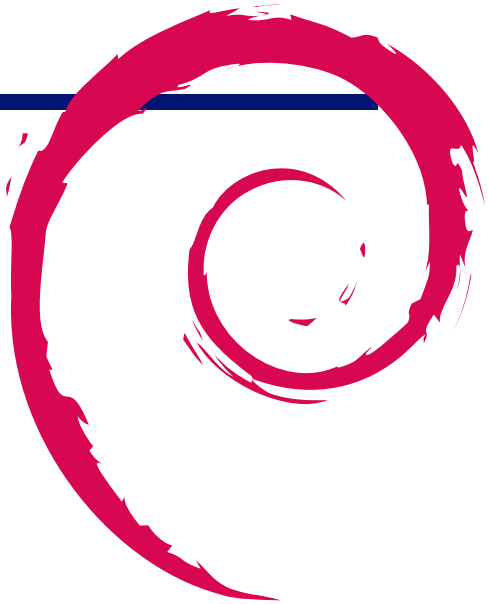
568 ページ中、人間の判別と食い違うページはなし。内訳はカラー 5 ページ、グレースケール 3 ページ、文章 560 ページ。

サンプル 3. **ローマ人のシリーズ** 1 **巻**

216 ページ中、人間の判別と食い違うページは 5 ページ。内訳はカラー 6 ページ、グレースケール 9 ページ、文章 201 ページ。そのうちカラー 4 ページの判別に失敗していますが、原因はカバーページがベージュで文章として判別されました。地図やイラストと文章が混じったページも学習通り文章として判別されています。

サンプル 4. 50 **年前に発行された芥川賞受賞作**

40 年近く前に出版された本で、 280 ページ中、人間の判別と食い違うページが 16 ページ。内訳はカラー 6 ページ、文章 274 ページ。判別の失敗が多い理由は変色と推測され、カラーだけではなくイラストとも多く間違って判別されました。



5 Weka を使ってみる

前田耕平

5.1 概要

ニューラルネットワークをはじめとするデータマイニングを一般人が使うには、本庄さんのネタのように理論を理解した上で自分でプログラムを作らないといけないとすると、非常にハードルが高いと思います。学生のころに学んだり、研究していたか、仕事として普段から扱っているような人でなければ、単語としては耳にしたことがあっても、なんだかよう分からん、という人がほとんどではないでしょうか。[*1]

そこで、バックグラウンドとしてニューラルネットワークだけでなく、データマイニングの基礎知識を持っていない私と同じような立場の人でも、 Debian なら気軽に試してみる環境を整えて、取り合えず使ってみることができるよ、という趣旨で、 Weka というツールを紹介します。

5.2 Weka とは

Weka とは、 “Waikato Environment for Knowledge Analysis” の略で、ニュージーランドの国立ワイカト大学[*2]で GPL のもとオープンソースで開発されているデータマイニングツールです。[*3] Java で書かれています。

5.3 インストール

Debian ではパッケージが用意されています。

```
$ sudo apt-get install weka
```

5.4 Weka の使い方

コマンドラインで `weka` スクリプトを実行します。

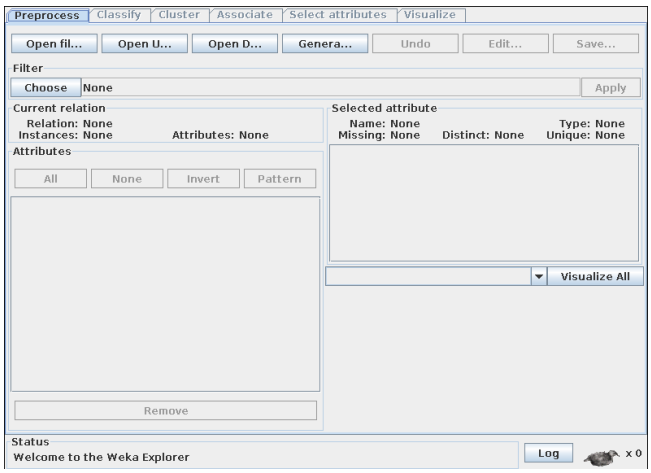
```
$ weka &
```

すると Weka のウィンドウが起動します。そこから、 “Applications” の “Explorer” を実行すると、 Weka Explorer が起動します。

[*1] Debian 勉強会の常連はむしろ知っている人の方が多いのかもしれませんが、少なくとも私は単語を耳にしたことがあるレベルです。
[*2] `http://www.waikato.ac.nz/`
[*3] `http://www.cs.waikato.ac.nz/~ml/weka/index.html`

図 6　Weka 起動メニュー

図 7　Weka Explorer



5.4.1　Weka で扱うデータフォーマット

Weka では、 ARFF(Attribute-Relation File Format) というフォーマットのテキストファイルを入力データとして扱います。

データフォーマットは次のようになります。

```
@relation 名前
@attribute 属性名 属性の型
@attribute 属性名 属性の型
:
:
@data
データ, データ,…, データ
```

- @relation はデータ全体の名前を指定します。
- @attribute はデータの属性を表し、 1 つ目の引数は属性名、 2 つめの引数は属性データの型を表します。
- @attribute のデータ型には、 numeric, real, integer, string, date 型を使えます。
  - numeric は real か integer を指定できます。
  - real は実数を指定できます。
  - integer は整数を指定できます。
  - string は文字列を指定できます。
  - date 型は日時で、デフォルトは”yyyy-MM-dd’T’HH:mm:ss” という書式です。
- @data データ領域の宣言です。
  - その次の行から CSV 形式でデータを記述します。
  - @attribute 行で上から設定した順に CSV の一行での左から右への各データとなります。

昨年の 11 月の勉強会で GNU R で扱った光熱費のデータと、気象庁が公開している気温、降水量などの気象データ[*4]を使ってみます。

まず、 CSV で以下のように記述したとします。

```
"年","月","降水量合計 (mm)","平均日平均気温 (℃)","平均日最高気温 (℃)","平均日最低気
温 (℃)","平均風速 (m/s)","最大風速 (m/s)","日照時間 (h)","電気使用量 (kWh)","電気使用
量 (kWh)/日","料金 (円)/日","合計料金","ガス使用量 (m3)","ガス使用量 (m3)/日","料金 (
円)/日","合計料金"
2007,1,50,6,10.8,1.1,1,5,188.9,234,6.88235294117647,161.235294117647,5482,9,0.333333333333333,80.962962962963,2186
2007,2,44,7.3,12.6,1.9,1.3,6,198.1,198,7.07142857142857,168.071428571429,4706,9,0.321428571428571,87.7142857142857,2456
(snip)
```

これは、 ARFF フォーマットでは以下のようになります。

[*4] http://www.data.jma.go.jp/obd/stats/etrn/index.php

```
@relation 降水量・気温（府中市）と電気代、ガス代の関係について

@attribute 年 real
@attribute 月 real
@attribute 降水量合計（mm） real
@attribute 平均日平均気温（℃） real
@attribute 平均日最高気温（℃） real
@attribute 平均日最低気温（℃） real
@attribute 平均風速（m/s） real
@attribute 最大風速（m/s） real
@attribute 日照時間（h） real
@attribute 電気使用量（kWh） real
@attribute 電気使用量（kWh）/日 real
@attribute 料金（円）/日 real
@attribute 合計料金 real
@attribute ガス使用量（m3） real
@attribute ガス使用量（m3）/日 real
@attribute 料金（円）/日 real
@attribute 合計料金 real

@data
2007,1,50,6,10.8,1.1,1,5,188.9,234,6.88235294117647,161.235294117647,5482,9,0.333333333333333,80.962962962963,2186
2007,2,44,7.3,12.6,1.9,1.3,6,198.1,198,7.07142857142857,168.071428571429,4706,9,0.321428571428571,87.7142857142857,2456
(snip)
```

5.4.2 ARFF をロードする

先ほど用意した kohnetsu.arff をロードしてみましょう。 Preprocess タブの Open File ボタンを押します。ダイアログが表示されるので、 kohnetsu.arff を指定します。 (図 8)

ARFF ファイルを読み込むと図 9 のようになります。 UTF-8 エンコードであれば、ご覧のとおり日本語も正常に読み込めます

図 8　ARFF ファイルをロードする

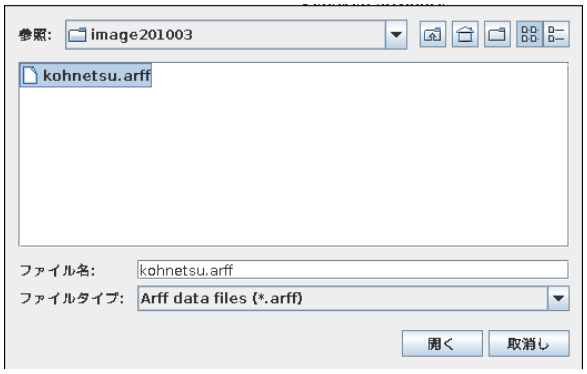
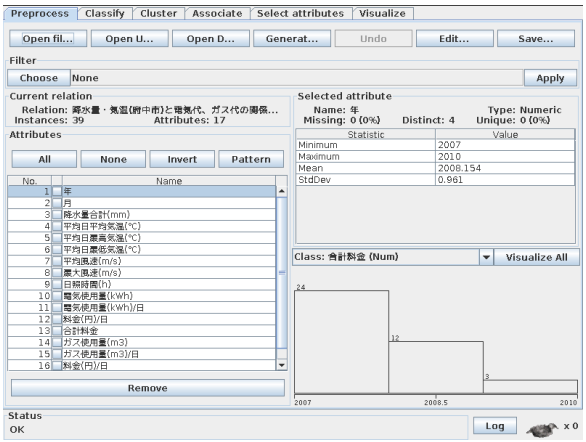


図 9　ARFF からデータを読み込んだ結果



5.4.3 可視化してみる

それでは読み込んだデータを可視化してみましょう。 Visualize タブをクリックすると図 10 のようなマトリックスが表示されます。

適当に開いてみます。 Y 軸に一日の平均気温の月平均 (℃) と、 X 軸に一日あたりの電気使用量 (kWh) を取って見てみると図 11 のようになります。

図 10　Visualize 画面

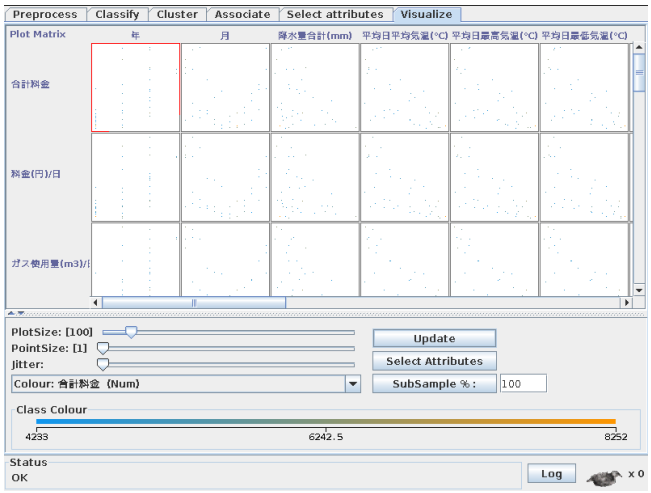
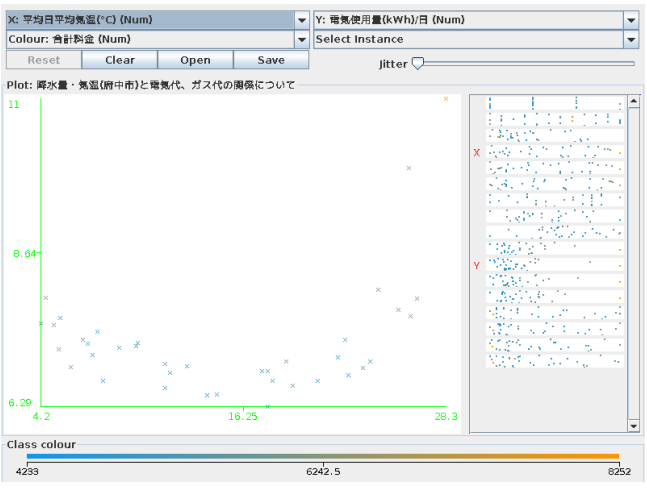


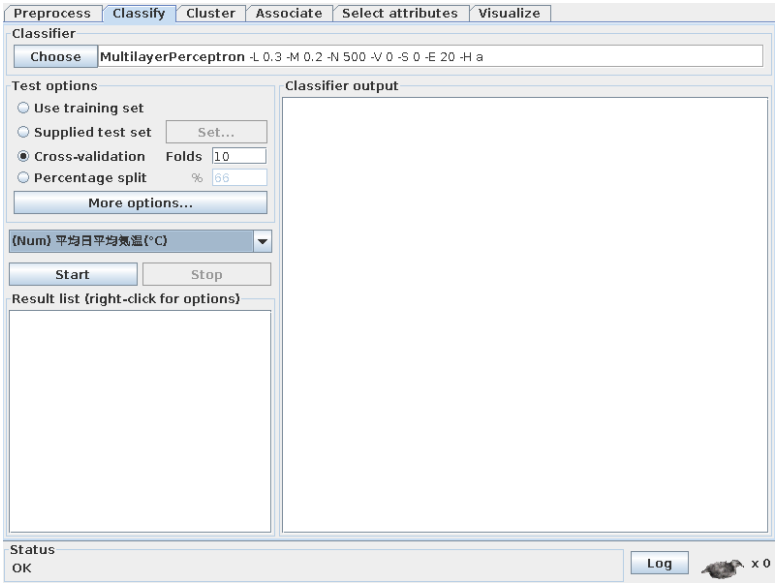
図 11　Visualize 詳細画面



5.4.4　分類してみる

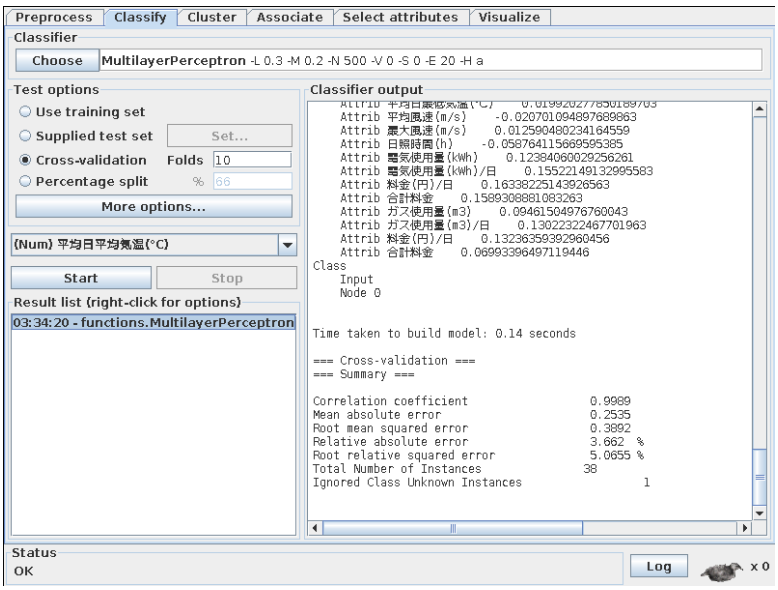
次に分類してみます。 Classify タブ → Choose ボタンを押し、表示されたツリーから Multilayer Perceptron (ニューラルネットワークによる分類) を選択します。次に、 (Num) 平均日平均気温 (℃) を目的関数として選択します。

図 12　Classify 画面



Start ボタンをクリックすると分類が実行され、結果が表示されます。

図 13　Classify 結果



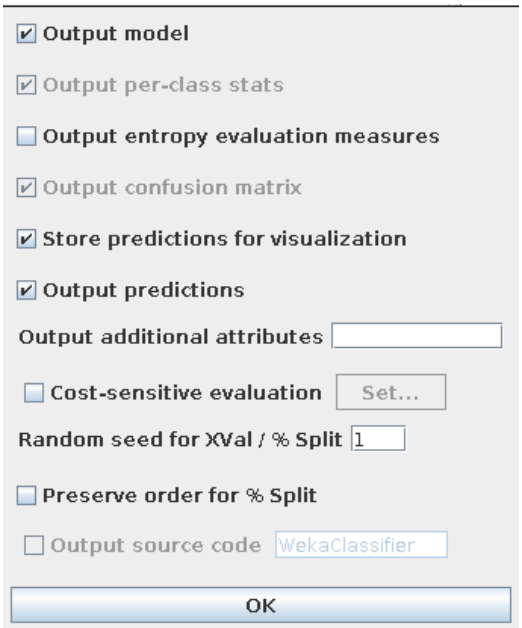
5.4.5　予測してみる

実は、 kohnetsu.arff の最後の行 (2010 年 3 月のデータ) は、ほとんどの項目を’?’ を入力しています。

```
2010,3,?,?,?,?,?,?,?,?,?,?,?,?,?,?
```

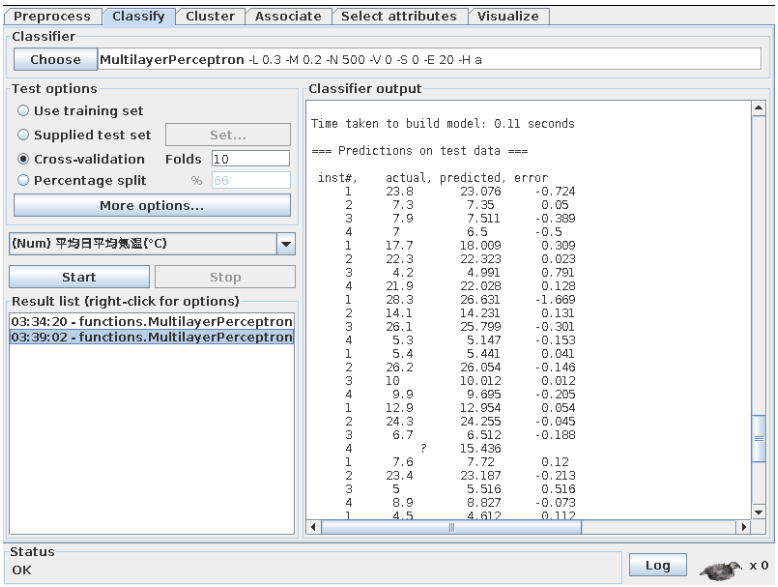
これはまだ今月のデータが出ていないからです。それでは、これらを予測してみます。先ほどの Classify タブの画面で More options をクリックし、 Output predictions のチェックを入れ、 OK を押します。

図 14　Classify の More Options ダイアログ



Start を実行すると、予測結果が表示されます。 actual が実データで predicted が予測結果です。今回見たいのは、 3 月の平均日平均気温です。 actual が ? になっている行の、 predicted の値を見ると、 15.436 となっています。過去読み込ませたデータからニューラルネットワークでの分析して予測した結果、おそらく一日あたりの平均気温は 15.4 ℃ になるという予測です。来月、気象庁の統計データが更新されたら確認してみましょう。

図 15　予測結果



5.5　まとめ

なんとなく使えそうな気がしたでしょうか。私はこれで毎月のガス代、電気代、さらには水道代の予測をして、給与日前日の予算計画に利用していこうと思います。仕事でも、企画上の裏付けデータの分析、予測などにも使えそうですね。

5.6　参考文献

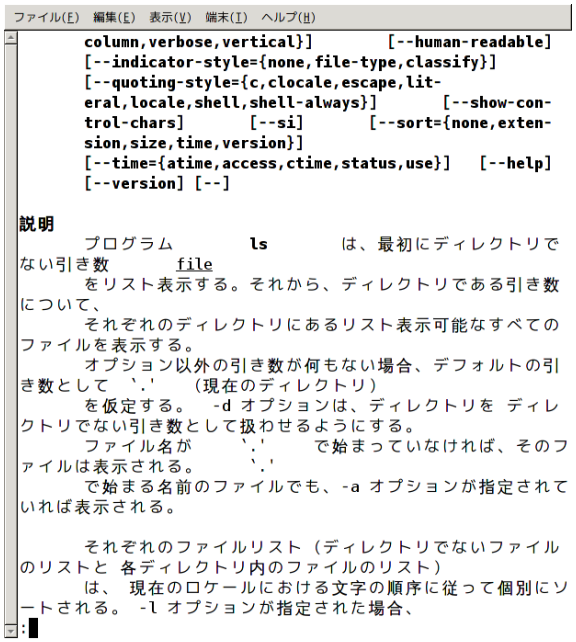
- http://www.ilibrary.jp/MOTtextBooks/text/weka.pdf
- http://web.sfc.keio.ac.jp/~soh/dm03/
- http://www1.doshisha.ac.jp/~mjin/R/23.pdf
- http://weka.wikispaces.com/ARFF

6 man-db を深追いした

日比野 啓

6.1 日本語の man が変

こんな感じ



日本語の文字の表示幅がアルファベットと同じだと判定されてしまっているので、結果的に日本語で書かれた行の長さが倍ぐらいになってしまっています。

6.2 解は groff にあり

6.2.1 そもそも roff ってどういうもの?

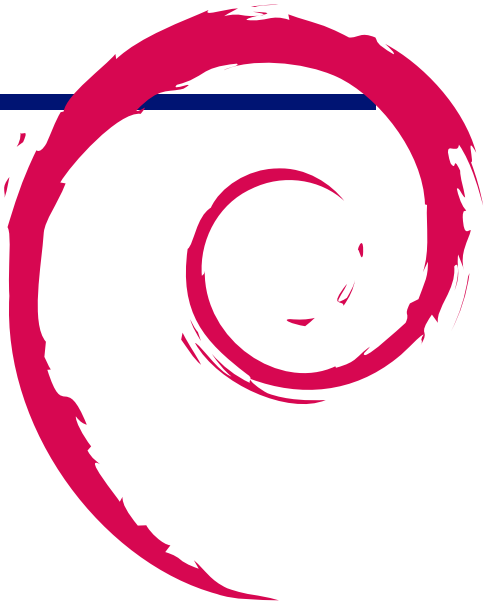
簡単に言えば、タイプライターのようなものです。いろいろなデバイスに対して印字を行なうことができるシステムになっていますが、 man では文字端末に文字の幅と高さを考慮しながら印字をしていきます。

6.2.2 roff 文字幅の指定

以前のバージョンの groff では日本語の文字の表示幅がアルファベットの倍であることを unicode の code point の範囲で設定してありました。 groff にはもともと code point の範囲で文字の幅を指定する機能が無いため、その機能の追加と日本語文字の表示幅を倍にするパッチがあててありました。

更新されたバージョンにおいては日本語対応のパッチが当たらなくなってしまっていますが、やはり同様の対処を行なう

必要がありそうです。



7 Debian で libfftw を使ってみる

上川純一

7.1 はじめに

Debian で FFT を取り扱う C のアプリケーションを書いてみたいと思うことはありませんか?今日は音声データを分析してみましょう。

wav ファイルを入力として受け取り、 FFT を実行してその結果を表示するアプリケーションを作成してみます。

7.2 インストール

libfftw3 をインストールします。あと、音声ファイルをロードするために sndfile1 を利用します。

```
$ apt-get install libfftw3-dev libsndfile1-dev
```

7.3 実験対象の準備:簡単な sine 波を作成する

まず、テスト用に FFT の結果が予想できるデータを作成してみます。ここでは 440Hz のきれいなサイン波を作成しています。

$$data(x) = sin(\frac{2\pi 440x}{44100})^{*5}$$

[*5] sin は radian

```
/*BINFMTC: -lsndfile -lm

  Create a sine wave at 44.1kHz for 1 second called sine.wav
 */
#include <stdlib.h>
#include <stdio.h>
#include <sndfile.h>
#include <math.h>

int create_sine(const char* filename, int size, double frequency)
{
  SF_INFO sfinfo = {
    .frames = size,
    .samplerate = 44100,
    .channels = 1,
    .format = SF_FORMAT_WAV | SF_FORMAT_PCM_16,
    .sections = 0,
    .seekable = 0
  };
  SNDFILE* s = sf_open(filename, SFM_WRITE, &sfinfo);
  double* data = malloc(sizeof(double) * size);
  int i;

  for (i=0; i < size; ++i)
    {
      data[i] = sin(frequency * 2.0 * M_PI * i / 44100.0);
    }

  sf_writef_double(s, data, size);
  sf_close(s);
  return 0;
}

int main()
{
  return create_sine("sine.wav", 44100, 440.0);
}
```

7.4 実験対象の準備:複雑な入力値例の準備

テスト用の入力値として、適当な wav ファイルを用意しましょう。

今回は手元で、 aeolus というオルガンシミュレータを起動し、 jack で接続させ、 ecasound を jack 入力に対して待機させ、 qjackctl で接続させて収録しました。

それなりに長い時間録音したデータから 16-bit mono の PCM データ 1 秒分を切り出して実験用データを作成しました。

```
$ qjackctl &
$ aeolus &
$ vkeybd &
$ ecasound -i jack -o test.wav
ctrl-C で中断
$ sweep test.wav # 適当に編集
$ file ra-mono.wav  # 切り出した結果を確認
ra-mono.wav: RIFF (little-endian) data, WAVE audio, Microsoft PCM, 16 bit, mono 44100 Hz
```

7.5 FFTW を使って wav ファイルを処理してみる

sndfile と fftw3 を使ってフーリエ変換して出力をダンプしてみましょう。サンプルコードは sndfile を使い double の配列に wav ファイルの中身を展開して、その内容を fftw に渡して処理しています。 double の値は各 1/44100 秒の瞬間における空気の圧力を表しているようです。

```
/*BINFMTC: -lsndfile -lfftw3 -lm
 */

#include <stdlib.h>
#include <stdio.h>
#include <sndfile.h>
#include <math.h>
#include <complex.h>
#include <fftw3.h>

/*
  process with FFTW
 */
void study_sound(double* data, int size)
{
  fftw_complex* spectrum;
  fftw_plan p;
  int i;

  spectrum = (fftw_complex*) fftw_malloc(sizeof(fftw_complex) * (size / 2 + 1));
  p = fftw_plan_dft_r2c_1d(size, data, spectrum, FFTW_ESTIMATE);

  /* process with FFTW */
  fftw_execute(p);

  /* dump output in CSV format */
  printf("i,abs,arg\n");
  for (i=0; i<(size/2+1); ++i) {
    printf("%i,%f,%f\n", i,
           cabs(spectrum[i]),
           carg(spectrum[i]) / 2.0 / M_PI * 360.0);
  }
  fftw_destroy_plan(p);
  fftw_free(spectrum);
}

/*
  Process wav file.

  @return 1 on failure, 0 on success.
*/
int process_wav_file(const char* filename, int size)
{
  SF_INFO sfinfo = {0, 0, 0, 0, 0, 0};
  SNDFILE* s = sf_open(filename, SFM_READ, &sfinfo);
  double* data = malloc(sizeof(double) * size);

  if (!s || !data)
    {
      fprintf(stderr,
              "Something went wrong opening the file or allocating memory\n");
      return 1;
    }
  if (sfinfo.channels != 1)
    {
      fprintf(stderr,
              "Please give me monaural audio data\n");
      return 1;
    }

  /* Read wav file into an array of double */
  sf_readf_double(s, data, size / sfinfo.channels);
  study_sound(data, size / sfinfo.channels);
  sf_close(s);
  return 0;
}

int main(int argc, char** argv)
{
  process_wav_file(argv[1], atoi(argv[2]));
  return 0;
}
```

実行してみます。

```
$ ./sndfile-fftw.c sine.wav 44100 > sine.csv
$ ./sndfile-fftw.c ra-1sec.wav 44100 > ra.csv
```

7.6 出力を確認してみる

CSV ファイル形式でデータが出力されました。簡単にグラフを作成するためのツールとしてここでは R を使ってみます。

```
$ R
> sine <- read.csv("sine.csv")
> ra <- read.csv("ra.csv")
> postscript("sine.eps", horizontal=FALSE, height=3, width=3)
> plot(sine$i, sine$abs, xlim=c(400,500), ylim=c(0,22000), type="l")
> dev.off()
> postscript("ra.eps", horizontal=FALSE, height=3, width=3)
> plot(ra$i, ra$abs, xlim=c(0,2000), ylim=c(0,100), type="l")
> dev.off()
```

図 7.6 の 440Hz のサイン波を処理した結果を見てみると、 440 Hz あたりにグラフの突起があるのが見て取れます。

しかし、実際にオルガン音を処理した結果の図 7.6 を見てみると、グラフに突起が多数あって、結構複雑です。そのまま簡単に処理させてくれはしなさそうです。

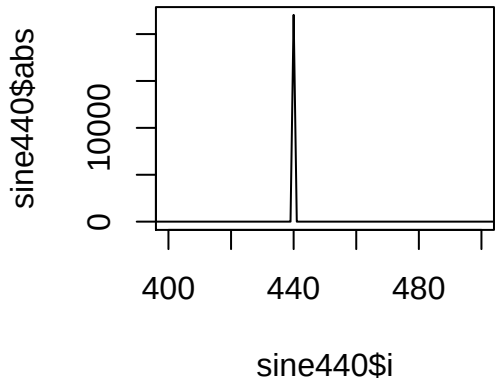


図 16　440Hz のサイン波

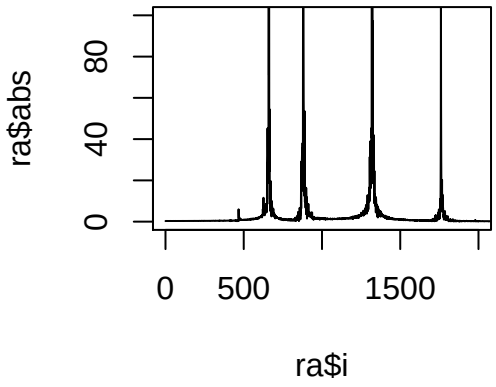
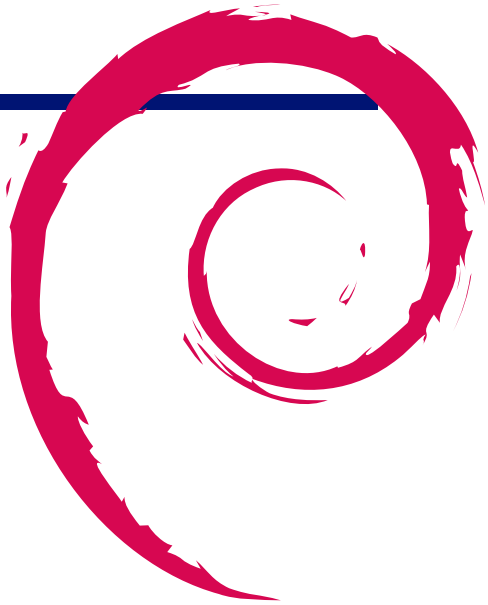


図 17　ラを aeolus で適当に演奏した音



8 dpkg ソース形式 “3.0 (quilt)”

吉野与志仁

8.1 はじめに

普段 Debian を使っている皆さんはご存知とは思いますが、近々デフォルトになる予定で squeeze の release goal である Debian ソースパッケージのフォーマット “3.0 (quilt)” の復習をしたいと思います。

8.2 “1.0”

まずは “3.0 (quilt)” の前に、いままで一般に使われてきたフォーマット (“1.0”) を簡単にまとめます。
1.0 では、ソースパッケージは以下の 3 ファイルで構成されます。

- *packagename*-*upstreamversion*.orig.tar.gz
- *packagename*-*debianversion*.diff.gz
- *packagename*-*debianversion*.dsc

なお、正確には 1.0 は 2 種類あって、上の通常のパッケージのほかに “Debian native な” パッケージがあります。Debian native パッケージは次の 2 ファイルで構成されます。

- *packagename*-*version*.tar.gz
- *packagename*-*version*.dsc

ここで、*.orig.tar.gz には、通常上流の元のソースツリーが含まれます。*.diff.gz には、ソースパッケージからパッケージなどをビルドするのに必要なスクリプトなどが入った debian/ ディレクトリや、上流のソースに対するパッケージメンテナの変更が含まれます。

8.3 しかし

と説明してきましたが、このファイル構成には

1. アーカイブの圧縮形式に gzip しか使えない
2. 複数のアーカイブで構成される上流のソースがそのまま扱えない
3. メンテナが当てたソースへのパッチが全部つながってしまっている
4. debian/ 以下にバイナリファイルが直接置けない

などの問題点があります。
そこで、さまざまな方法が検討されました。

1 は、これにより上流が bz2 で配布していても gz に圧縮し直さなければならなかったりしていました。`*.orig.tar.gz` の中身が上流のアーカイブの実体である、といった方法なども（ ちょっと無駄ですが...）使われてきました。この方法はビルド時にその tarball を展開して作業します。 cdbs にはこの方法へのサポートもあります。

2 は 1 と同様の方法で複数の tarball が入った`*.orig.tar.gz` を用意したりしていました。

3 は、当たっているパッチのそれぞれがどんな意図で行われたのかがわからない、ということ、また、 `debian/`以下のファイルも上流ソースへのパッチも一緒くたになってしまっていること、が問題でした。そこで、まとまった意味のある単位に分割されたパッチをまず用意しておき、それらを `debian/patches/` 下に配置し、その細かいパッチをビルド時に当てる/外すフレームワーク (patch system) が利用されています。これには dpatch や quilt などがあります。なお、この細かいパッチのそれぞれには、先頭にパッチの意図を説明する文章を記述することが推奨されています (`http://dep.debian.net/deps/dep3/`)。

4 は、バイナリファイルの diff を取ろうとしても普通の patch ではできないことが原因なので、 uuencode などでテキストに落として patch を取る、といった手法が用いられてきました。

8.4 そこで

このような問題を解決するために新たなソースパッケージのフォーマットも検討されました。それが “`3.0 (quilt)`” フォーマット（ と “`3.0 (native)`” フォーマット）です。

`3.0 (quilt)` は次の 3 つ以上のファイルで構成されます。

- *packagename*`-`*upstreamversion*`.orig.tar.`*ext*
- *packagename*`-`*upstreamversion*`.orig-`*component*`.tar.`*ext*（ 任意）
- *packagename*`-`*debianversion*`.debian.tar.`*ext*
- *packagename*`-`*debianversion*`.dsc`

なお、 `1.0` にあった Debian native パッケージに相当する `3.0 (native)` は次の 2 ファイルで構成されます。

- *packagename*`-`*version*`.tar.`*ext*
- *packagename*`-`*version*`.dsc`

ここで、まず、 tar の拡張子部分 *ext* に gz のほか、 bz2, lzma, xz が利用できるようになりました。これにより問題 1 が解決されました（ `3.0 (native)` における主な変更点はこれです）。

また、 *component* の部分を適当に変えることにより複数の tarball をきちんと扱えるようになりました。これが問題 2 を解決します。

次に、 `debian/`下のファイルはすべて `*.debian.tar.gz` に入れることになりました。これですべてが混ざった状態はなくなりました。さらに、 `debian/patches/` 下のパッチが、パッチシステム quilt と基本的に同じ方法で `dpkg-source(1)` によって “ソースパッケージの展開時に” 自動的に当たるようになりました。これにより、ビルド時にパッチを当てるように `debian/rules` ファイルを記述する必要はなくなりましたし、 `debian/control` ファイルに `Build-Depends: quilt` などと書く必要もなくなりました。これらによって問題 3 は解決されました。

問題 4 については、 `debian/`下のファイルを diff として保持することはもはやなくなったので解決し、 `*.debian.tar.`*ext* に直にバイナリファイルを配置できます。

8.5 最近の動向

この `3.0 (quilt)` フォーマットは、 Debian のアーカイブが巨大化していっているため、 gzip より lzma（ lenny 当時で、現在なら xz）を使って圧縮すればサイズを抑えられるといった理由で、 lenny 以降でより推進されるようになりました。 Debian にあるソースパッケージすべてが `3.0 (quilt)` 化可能になったら（ 潰すべき minor/wishlist バグが `http://bugs.debian.org/cgi-bin/pkgreport.cgi?users=hertzog@debian.org;tag=3.0-quilt-by-default` にあります）、 dpkg はデフォルトで `3.0 (quilt)` フォーマットでソースパッケージをビルド

するように変更される、とのことです。というわけでこれから作るパッケージは 3.0 化しましょう。

8.6 例 1： gzip パッケージ

具体例として、まず gzip ソースパッケージ (1.3.12-9) を `3.0 (quilt)` 化してみました。このソースパッケージは直に patch を当てた形式を取っていました。 `dpkg-source(1)` による展開時のメッセージで分かります:

```
$ apt-get source gzip
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています
状態情報を読み取っています... 完了
479kB のソースアーカイブを取得する必要があります。
取得:1 http://ftp.jp.debian.org testing/main gzip 1.3.12-9 (dsc) [1,647B]
取得:2 http://ftp.jp.debian.org testing/main gzip 1.3.12-9 (tar) [462kB]
取得:3 http://ftp.jp.debian.org testing/main gzip 1.3.12-9 (diff) [15.7kB]
479kB を 1s で取得しました (250kB/s)
dpkg-source: info: extracting gzip in gzip-1.3.12
dpkg-source: info: unpacking gzip_1.3.12.orig.tar.gz
dpkg-source: info: applying gzip_1.3.12-9.diff.gz
dpkg-source: info: upstream files that have been modified:
 gzip-1.3.12/.gbp.conf
 gzip-1.3.12/deflate.c
 gzip-1.3.12/gzip.1

(snip)
```

8.6.1 バージョンの切り換え、ビルド

現在のデフォルトは `1.0` なので、 `3.0 (quilt)` と明示してからソースパッケージを作ります。

```
$ cd gzip-1.3.12/
$ mkdir -p debian/source
$ echo '3.0 (quilt)' > debian/source/format
$ debuild -S -us -uc
 dpkg-buildpackage -rfakeroot -d -us -uc -S
dpkg-buildpackage: set CFLAGS to default value: -g -O2

(snip)

 dpkg-source -b gzip-1.3.12
dpkg-source: info: using source format `3.0 (quilt)'
dpkg-source: info: building gzip using existing ./gzip_1.3.12.orig.tar.gz
dpkg-source: info: local changes stored in gzip-1.3.12/debian/patches/debian-changes-1.3.12-9, the modified files are:
 gzip-1.3.12/.gbp.conf
 gzip-1.3.12/deflate.c

(snip)

dpkg-source: info: building gzip in gzip_1.3.12-9.debian.tar.gz
dpkg-source: info: building gzip in gzip_1.3.12-9.dsc
 dpkg-genchanges -S >../gzip_1.3.12-9_source.changes

(snip)
```

と、とりあえず昔の diff.gz（ の上流ソースへのパッチ部分）に相当する 1 つのパッチを自動で出力してくれます。この大きなパッチそのままではこのフォーマットにした意味がほとんどないので分けましょう。また、パッチの説明文テンプレートも `debian/changelog` を基にして付けてくれるので、修正して正しい説明にしましょう。（ 今回は共に省略）

バイナリパッケージをビルドしてみます。

```
$ debuild -b -us -uc
 dpkg-buildpackage -rfakeroot -D -us -uc -b
dpkg-buildpackage: set CFLAGS to default value: -g -O2
dpkg-buildpackage: set CPPFLAGS to default value:
dpkg-buildpackage: set LDFLAGS to default value:
dpkg-buildpackage: set FFLAGS to default value: -g -O2
dpkg-buildpackage: set CXXFLAGS to default value: -g -O2
dpkg-buildpackage: source package gzip
dpkg-buildpackage: source version 1.3.12-9
dpkg-buildpackage: source changed by Bdale Garbee <bdale@gag.com>
dpkg-buildpackage: host architecture amd64
dpkg-checkbuilddeps: Unmet build dependencies: mingw32
dpkg-buildpackage: warning: Build dependencies/conflicts unsatisfied; aborting.
dpkg-buildpackage: warning: (Use -d flag to override.)
debuild: fatal error at line 1330:
dpkg-buildpackage -rfakeroot -D -us -uc -b failed
```

`Build-Depends` を満たして、ビルドします。

```
$ mk-build-deps
dh_testdir

(snip)

dpkg-deb: ‘../gzip-build-deps_1.0_all.deb’ にパッケージ ‘gzip-build-deps’ を構築しています。

The package has been created.
Attention, the package has been created in the current directory,
not in ".." as indicated by the message above!
$ sudo dpkg -i gzip-build-deps_1.0_all.deb
未選択パッケージ gzip-build-deps を選択しています。
(データベースを読み込んでいます ... 現在 203017 個のファイルとディレクトリがインストールされています。)
(gzip-build-deps_1.0_all.deb から) gzip-build-deps を展開しています...
dpkg: 依存関係の問題により gzip-build-deps の設定ができません:
 gzip-build-deps は以下に依存 (depends) します: mingw32 ... しかし:
  パッケージ mingw32 はまだインストールされていません。
dpkg: gzip-build-deps の処理中にエラーが発生しました (--install):
 依存関係の問題 - 設定を見送ります
以下のパッケージの処理中にエラーが発生しました:
 gzip-build-deps
$ sudo aptitude install gzip-build-deps
パッケージリストを読み込んでいます... 完了

(snip)

タスクの記述を読み込んでいます... 完了

現在の状態: 依存関係破損が 0 個 [-1]。
$ debuild -b -us -uc
 dpkg-buildpackage -rfakeroot -D -us -uc -b
dpkg-buildpackage: set CFLAGS to default value: -g -O2

(snip)

dpkg-deb: ‘../gzip_1.3.12-9_amd64.deb’ にパッケージ ‘gzip’ を構築しています。
 dpkg-genchanges -b >../gzip_1.3.12-9_amd64.changes
dpkg-genchanges: binary-only upload - not including any source code
dpkg-buildpackage: binary only upload (no source included)
Now running lintian...
W: gzip: missing-dependency-on-install-info
Finished running lintian.
```

8.7 例 2： bash-completion パッケージ

次に、パッチシステムとして quilt を使っていた bash-completion パッケージを `3.0 (quilt)` 化してみました。

8.7.1 まずビルド

パッチシステムでは、ビルド前はパッチは当たっていないので、自動でツリーにパッチが当てられてソースパッケージがビルドされます。

```
$ apt-get source bash-completion
パッケージリストを読み込んでいます... 完了

(snip)

$ cd bash-completion-1.1/
$ mkdir -p debian/source
$ echo ’3.0 (quilt)’ > debian/source/format
$ debuild -S -us -uc
 dpkg-buildpackage -rfakeroot -d -us -uc -S
dpkg-buildpackage: set CFLAGS to default value: -g -O2

(snip)

 fakeroot debian/rules clean
dh --with quilt clean
   dh_testdir
   dh_auto_clean
   dh_quilt_unpatch
適用されているパッチはありません
   dh_clean
 dpkg-source -b bash-completion-1.1
dpkg-source: info: using source format ‘3.0 (quilt)’
dpkg-source: warning: patches have not been applied, applying them now (use --no-preparation to override)
dpkg-source: info: applying 01-fix_550943.patch
dpkg-source: info: applying 02-fix_552109.patch
dpkg-source: info: applying 03-fix_552631.patch
dpkg-source: info: building bash-completion using existing ./bash-completion_1.1.orig.tar.gz
dpkg-source: info: building bash-completion in bash-completion_1.1-3.debian.tar.gz
dpkg-source: info: building bash-completion in bash-completion_1.1-3.dsc
 dpkg-genchanges -S >../bash-completion_1.1-3_source.changes

(snip)
```

今回は問題ありませんでしたが、 3.0 (quilt) は quilt とは微妙に異なり、すべて patch -p1 として扱われるので path を適当に調整する必要があるかもしれません。

8.7.2 quilt 周りの変更

パッチは展開時に当たるので、 debian/rules 内でパッチを当てている部分はもう要りません。このソースパッケージは dh(1) を使用していたので簡単でした。

```
--- debian/rules        2010-03-18 10:50:30.000000000 +0900
+++ debian/rules.new        2010-03-18 10:54:53.000000000 +0900
@@ -21,11 +21,11 @@

 build: build-stamp
 build-stamp:
-        dh --with quilt build
+        dh build
         touch $@

 clean:
-        dh --with quilt $@
+        dh $@

 install: install-stamp
 install-stamp: build
```

quilt には通常は Build-Depends しません。

```
--- debian/control        2010-03-18 10:50:30.000000000 +0900
+++ debian/control.new        2010-03-18 10:57:00.000000000 +0900
@@ -3,7 +3,7 @@
 Priority: standard
 Maintainer: Bash Completion Maintainers <bash-completion-devel@lists.alioth.debian.org>
 Uploaders: David Paleino <dapal@debian.org>
-Build-Depends: debhelper (>= 7.0.50), quilt (>= 0.46-7~)
+Build-Depends: debhelper (>= 7.0.50)
 Build-Depends-Indep: perl
 Standards-Version: 3.8.3
 Vcs-Git: git://git.debian.org/git/bash-completion/debian.git
```

8.7.3 もう一回

今度は初めから 3.0 (quilt) です。

```
$ debuild -S -us -uc
 dpkg-buildpackage -rfakeroot -d -us -uc -S

(snip)

 fakeroot debian/rules clean
dh clean
   dh_testdir
   dh_auto_clean
   dh_clean
 dpkg-source -b bash-completion-1.1
dpkg-source: info: using source format `3.0 (quilt)'
dpkg-source: info: building bash-completion using existing ./bash-completion_1.1.orig.tar.gz
dpkg-source: info: building bash-completion in bash-completion_1.1-3.debian.tar.gz
dpkg-source: info: building bash-completion in bash-completion_1.1-3.dsc
 dpkg-genchanges -S >../bash-completion_1.1-3_source.changes
dpkg-genchanges: not including original source code in upload
dpkg-buildpackage: binary and diff upload (original source NOT included)
Now running lintian...
Finished running lintian.
```

バイナリパッケージをビルドします。

```
$ debuild -b -us -uc
 dpkg-buildpackage -rfakeroot -D -us -uc -b
dpkg-buildpackage: set CFLAGS to default value: -g -O2

(snip)

dpkg-deb: `../bash-completion_1.1-3_all.deb' にパッケージ `bash-completion' を構築しています。
 dpkg-genchanges -b >../bash-completion_1.1-3_amd64.changes
dpkg-genchanges: binary-only upload - not including any source code
dpkg-buildpackage: binary only upload (no source included)
Now running lintian...
Finished running lintian.
```

8.8 例 3： ptex-bin パッケージ

次に、実際には複数のソースアーカイブを使用している ptex-bin ソースパッケージ (3.1.11+0.04b-0.1) を 3.0 (quilt) 化してみました。

ptex-bin ソースパッケージは、実際には ptex-src-*.tar.gz と jmpost-*.tar.gz の 2 つの上流 tarball から構成されます。さらに Build-Depends: ptex-buildsupport となっていますが、この ptex-buildsupport パッケージは tetex-src-*-stripped.tar.gz のみが含まれる、ほぼビルド専用のパッケージです。すなわち、 3 つの上流 tarball が使われています。その上、このソースパッケージには debian/patches/ディレクトリがありますが、 quilt や dpatch といった近代的なものではなくビルド時に patch を debian/rules 内で直接呼ぶ構成になっていました。

8.8.1 ソースの用意、名前の変更

```
$ apt-get source ptex-bin
$ apt-get source ptex-buildsupport
```

適当に名前を変えます。ここでは

```
$ mv ptex-bin-3.1.11+0.04b/ptex-src-3.1.11.tar.gz ptex-bin_3.1.11+0.04b+3.0.orig.tar.gz
$ mv ptex-bin-3.1.11+0.04b/jmpost-0.04b.tar.gz ptex-bin_3.1.11+0.04b+3.0.orig-jmpost.tar.gz
$ mv ptex-buildsupport-3.0/tetex-src-3.0-stripped.tar.gz ptex-bin_3.1.11+0.04b+3.0.orig-tetex-stripped.tar.gz
```

としました。 3.0 (quilt) では、*.orig.tar.*ext* がまずメインで展開され、そのソースツリーに *component* ディレクトリが掘られてその下に各*.orig-*component*.tar.*ext* が展開されます。また、 *component* には英数字とハイフンのみが使えます。

8.8.2 とりあえずビルド

```
$ mkdir ptex-bin-3.1.11+0.04b+3.0
$ cd ptex-bin-3.1.11+0.04b+3.0/
$ cp -a ../ptex-bin-3.1.11+0.04b/debian .
$ mkdir -p debian/source
$ echo '3.0 (quilt)' > debian/source/format
$ dch -v 3.1.11+0.04b+3.0-0.1
( 適当に changelog 追加)
$ debuild -S -us -uc
 dpkg-buildpackage -rfakeroot -d -us -uc -S
dpkg-buildpackage: set CFLAGS to default value: -g -O2
dpkg-buildpackage: set CPPFLAGS to default value:
dpkg-buildpackage: set LDFLAGS to default value:
dpkg-buildpackage: set FFLAGS to default value: -g -O2
dpkg-buildpackage: set CXXFLAGS to default value: -g -O2
dpkg-buildpackage: source package ptex-bin
dpkg-buildpackage: source version 3.1.11+0.04b+3.0-0.1
dpkg-buildpackage: source changed by YOSHINO Yoshihito <yy.y.ja.jp@gmail.com>
 fakeroot debian/rules clean
dh_testdir
dh_testroot
rm -f build-stamp configure-stamp
# Add here commands to clean up after the build process.
# Remove teTeX source directory.
rm -rf tetex-src-3.0
dh_clean
dh_clean: Compatibility levels before 5 are deprecated.
 dpkg-source -b ptex-bin-3.1.11+0.04b+3.0
dpkg-source: info: using source format `3.0 (quilt)'
dpkg-source: info: building ptex-bin using existing ./ptex-bin_3.1.11+0.04b+3.0.
orig-jmpost.tar.gz ./ptex-bin_3.1.11+0.04b+3.0.orig-tetex-stripped.tar.gz ./ptex
-bin_3.1.11+0.04b+3.0.orig.tar.gz
dpkg-source: warning: ignoring deletion of file kanji.defines
dpkg-source: warning: ignoring deletion of file pconvert
dpkg-source: warning: ignoring deletion of file tftopl.ch

(snip)

dpkg-source: info: building ptex-bin in ptex-bin_3.1.11+0.04b+3.0-0.1.debian.tar
.gz
dpkg-source: info: building ptex-bin in ptex-bin_3.1.11+0.04b+3.0-0.1.dsc
 dpkg-genchanges -S >../ptex-bin_3.1.11+0.04b+3.0-0.1_source.changes
```

とりあえず空のソースツリーで作ったのでソースがないという warning はスルーします。

```
$ cd ..
$ dpkg-source -x ptex-bin_3.1.11+0.04b+3.0-0.1.dsc
dpkg-source: warning: extracting unsigned source package (ptex-bin_3.1.11+0.04b+3.0-0.1.dsc)
dpkg-source: info: extracting ptex-bin in ptex-bin-3.1.11+0.04b+3.0
dpkg-source: info: unpacking ptex-bin_3.1.11+0.04b+3.0.orig.tar.gz
dpkg-source: info: unpacking ptex-bin_3.1.11+0.04b+3.0.orig-jmpost.tar.gz
dpkg-source: info: unpacking ptex-bin_3.1.11+0.04b+3.0.orig-tetex-stripped.tar.gz
dpkg-source: info: unpacking ptex-bin_3.1.11+0.04b+3.0-0.1.debian.tar.gz
$ cd -
$ ls -F
COPYRIGHT      README.txt   jbibtex.ch       mkconf         ptexextra.h
COPYRIGHT.jis  configure*   jbibtex.defines  pconvert*      ptexhelp.h
Changes.txt    debian/      jmpost/          pdvitype.ch    tetex-stripped/
Files          jbibd.sed    kanji.c          pltotf.ch      tftopl.ch
INSTALL.txt    jbibextra.c  kanji.defines    ptex-base.ch   usage.c
Makefile.in    jbibextra.h  kanji.h.in       ptexextra.c    version.c
```

メインのソースのトップディレクトリでそのままビルドできるようなものはいいですが、 ptex-src-*.tar.gz はそうではないので微妙ですね... ともかく、今回はビルド用ディレクトリを掘ってビルドできるように debian/rules の変更が必要でした。

8.8.3 quilt 化

普段パッチシステム quilt を使っていればほとんど変更の必要はありませんが、このパッケージはパッチは分かれているものの自力で色々やっているのでまず quilt 化が必要でした。

全パッチを-p1 化した上で、 debian/rules に書かれている通りの順番でパッチを quilt import && quilt push しました。 quilt では同じパッチ名はダメのようですね。

```
$ cd debian/
$ mv patches patches.old
$ sed -i 's@^\(---\|+++\) @&ptex-bin-3.1.11+0.04b+3.0/tetex-stripped/@' patches.old/teTeX/*.patch
$ sed -i 's@^\(---\|+++\) @&ptex-bin-3.1.11+0.04b+3.0/@' patches.old/*.patch
$ sed -i 's@^\(---\|+++\) @&ptex-bin-3.1.11+0.04b+3.0/jmpost/@' patches.old/jmpost/*.patch
$ mv patches.old/Makefile.in.patch patches.old/pTeX_Makefile.in.patch
$ mv patches.old/jmpost/Makefile.in.patch patches.old/jmpost/jmpost_Makefile.in.patch
$ quilt import patches.old/teTeX/*.patch patches.old/*.patch patches.old/jmpost/*.patch
パッチ patches.old/teTeX/Makefile.in.patch を取り込んでいます (Makefile.in.patch として保存されます)
パッチ patches.old/teTeX/common.mk.patch を取り込んでいます (common.mk.patch として保存されます)
パッチ patches.old/teTeX/config.h.patch を取り込んでいます (config.h.patch として保存されます)
パッチ patches.old/teTeX/depend.mk.patch を取り込んでいます (depend.mk.patch として保存されます)
パッチ patches.old/teTeX/splitup.c.patch を取り込んでいます (splitup.c.patch として保存されます)
パッチ patches.old/teTeX/web2c.depend.mk.patch を取り込んでいます (web2c.depend.mk.patch として保存されます)
パッチ patches.old/pTeX_Makefile.in.patch を取り込んでいます (pTeX_Makefile.in.patch として保存されます)
パッチ patches.old/jmpost/jmpost_Makefile.in.patch を取り込んでいます (jmpost_Makefile.in.patch として保存されます)
$ cd ..
$ env QUILT_PATCHES=debian/patches quilt push -a
パッチ Makefile.in.patch を適用しています
patching file tetex-stripped/texk/web2c/web2c/Makefile.in

パッチ common.mk.patch を適用しています
patching file tetex-stripped/texk/make/common.mk

パッチ config.h.patch を適用しています
patching file tetex-stripped/texk/web2c/config.h

パッチ depend.mk.patch を適用しています
patching file tetex-stripped/texk/web2c/lib/depend.mk

パッチ splitup.c.patch を適用しています
patching file tetex-stripped/texk/web2c/web2c/splitup.c

パッチ web2c.depend.mk.patch を適用しています
patching file tetex-stripped/texk/web2c/web2c/depend.mk

パッチ pTeX_Makefile.in.patch を適用しています
patching file Makefile.in

パッチ jmpost_Makefile.in.patch を適用しています
patching file jmpost/Makefile.in

現在位置はパッチ jmpost_Makefile.in.patch です
$ rm -r debian/patches.old/
```

8.8.4 debian/rules, debian/control の変更

tarball をもう展開する必要はありません。ビルド用ディレクトリにコピーすることにします。パッチを当てる必要もないのでその辺も削ります。

```
--- debian/rules        2010-03-18 12:01:15.000000000 +0900
+++ debian/rules.new        2010-03-18 13:19:09.000000000 +0900
@@ -6,29 +6,14 @@
 # Uncomment this to turn on verbose mode.
 #export DH_VERBOSE=1

-####################
-# Things you should change when new upstream version is available.
-
-# Minimal teTeX source tarball.
-# You should install ptex-buildsupport package beforehand.
-TETEX_SRC_TARBALL=/usr/src/tetex-src-3.0-stripped.tar.gz
-
 # teTeX source directory.
-TETEX_SRC_DIR=tetex-src-3.0
-
-# pTeX source tarball.
-PTEX_SRC_TARBALL=ptex-src-3.1.11.tar.gz
+TETEX_SRC_DIR=tetex-src

 # pTeX source directory.
-PTEX_SRC_DIR=ptex-src-3.1.11
-
-# Japanized MetaPost tarball.
-JMPOST_SRC_TARBALL=jmpost-0.04b.tar.gz
+PTEX_SRC_DIR=ptex-src

 # Japanized MetaPost source directory.
-JMPOST_SRC_DIR=jmpost-0.04b
-
-####################
+JMPOST_SRC_DIR=jmpost-src

 DEB_HOST_GNU_TYPE        := $(shell dpkg-architecture -qDEB_HOST_GNU_TYPE)

@@ -42,31 +27,14 @@
 configure: configure-stamp
 configure-stamp:
         dh_testdir
-        # Unpack tarballs.
-        tar xfz $(TETEX_SRC_TARBALL)
+        cp -al tetex-stripped $(TETEX_SRC_DIR)
         mkdir $(TETEX_SRC_DIR)/texk/kpathsea_tetex
         mv $(TETEX_SRC_DIR)/texk/kpathsea/c-proto.h $(TETEX_SRC_DIR)/texk/kpathsea_tetex/
         rm -rf $(TETEX_SRC_DIR)/texk/kpathsea
         ln -s /usr/include/kpathsea $(TETEX_SRC_DIR)/texk/kpathsea
-        tar xfz $(PTEX_SRC_TARBALL) -C $(TETEX_SRC_DIR)/texk/web2c
-        tar xfz $(JMPOST_SRC_TARBALL) -C $(TETEX_SRC_DIR)/texk/web2c/$(PTEX_SRC_DIR)
-
-        # Apply patches to teTeX source
-        for f in debian/patches/teTeX/*.patch; do \
-                patch -p0 -d $(TETEX_SRC_DIR) < $$f; \
-        done
-
-        # Apply patches to pTeX source (should be named as *.patch)
-        # Put patches in debian/patches.
-        (for f in debian/patches/*.patch ; do \
-          patch -p0 -d $(TETEX_SRC_DIR)/texk/web2c/$(PTEX_SRC_DIR) < $$f ; \
-        done)
-
-        # Apply patches to Japanized MetaPost source (should be named as *.patch)
-        # Put patches in debian/patches/jmpost.
-        (for f in debian/patches/jmpost/*.patch ; do \
-          patch -p0 -d $(TETEX_SRC_DIR)/texk/web2c/$(PTEX_SRC_DIR)/$(JMPOST_SRC_DIR) < $$f ; \
-        done)
+        mkdir $(TETEX_SRC_DIR)/texk/web2c/$(PTEX_SRC_DIR)
+        for f in `find . -maxdepth 1 -type f`; do cp -al $$f $(TETEX_SRC_DIR)/texk/web2c/$(PTEX_SRC_DIR); done
+        cp -al jmpost $(TETEX_SRC_DIR)/texk/web2c/$(PTEX_SRC_DIR)/$(JMPOST_SRC_DIR)

         # Copy texmf.cnf from your system.
         cp /usr/share/texmf/web2c/texmf.cnf \
```

ptex-buildsupport は不要になりました。

```
--- debian/control        2010-03-18 12:01:15.000000000 +0900
+++ debian/control.new        2010-03-18 13:23:09.000000000 +0900
@@ -2,7 +2,7 @@
 Section: tex
 Priority: optional
 Maintainer: Masayuki Hatta (mhatta) <mhatta@debian.org>
-Build-Depends: debhelper (>> 4.0.0), texlive-binaries,
+Build-Depends: debhelper (>> 4.0.0), texlive-binaries,
 texlive-extra-utils, texlive-metapost, flex, bison, libkpathsea-dev, \
ptex-base (>= 2.4), ptex-buildsupport (>= 3.0), libtool
 Standards-Version: 3.7.3

 Package: ptex-bin
```

8.8.5 ビルド

```
$ debuild -S -us -uc
 dpkg-buildpackage -rfakeroot -d -us -uc -S
dpkg-buildpackage: set CFLAGS to default value: -g -O2

(snip)
```

試しに dpkg-source -x してみます。

```
$ dpkg-source -x ptex-bin_3.1.11+0.04b+3.0-0.1.dsc
dpkg-source: warning: extracting unsigned source package (ptex-bin_3.1.11+0.04b+3.0-0.1.dsc)
dpkg-source: info: extracting ptex-bin in ptex-bin-3.1.11+0.04b+3.0
dpkg-source: info: unpacking ptex-bin_3.1.11+0.04b+3.0.orig.tar.gz
dpkg-source: info: unpacking ptex-bin_3.1.11+0.04b+3.0.orig-jmpost.tar.gz
dpkg-source: info: unpacking ptex-bin_3.1.11+0.04b+3.0.orig-tetex-stripped.tar.gz
dpkg-source: info: unpacking ptex-bin_3.1.11+0.04b+3.0-0.1.debian.tar.gz
dpkg-source: info: applying Makefile.in.patch
dpkg-source: info: applying common.mk.patch
dpkg-source: info: applying config.h.patch
dpkg-source: info: applying depend.mk.patch
dpkg-source: info: applying splitup.c.patch
dpkg-source: info: applying web2c.depend.mk.patch
dpkg-source: info: applying pTeX_Makefile.in.patch
dpkg-source: info: applying jmpost_Makefile.in.patch
```

無事当たりました。バイナリパッケージをビルドします。

```
$ debuild -b -us -uc
 dpkg-buildpackage -rfakeroot -D -us -uc -b
dpkg-buildpackage: set CFLAGS to default value: -g -O2

(snip)

dpkg-buildpackage: warning: Build dependencies/conflicts unsatisfied; aborting.
dpkg-buildpackage: warning: (Use -d flag to override.)
debuild: fatal error at line 1330:
dpkg-buildpackage -rfakeroot -D -us -uc -b failed
$ mk-build-deps

(snip)

$ sudo dpkg -i ptex-bin-build-deps_1.0_all.deb

(snip)

$ sudo aptitude install ptex-bin-build-deps

(snip)

$ debuild -b -us -uc
 dpkg-buildpackage -rfakeroot -D -us -uc -b
dpkg-buildpackage: set CFLAGS to default value: -g -O2

(snip)

dh_builddeb
dh_builddeb: Compatibility levels before 5 are deprecated.
dpkg-deb: '../ptex-bin_3.1.11+0.04b+3.0-0.1_amd64.deb' にパッケージ 'ptex-bin' を構築しています。
dpkg-deb: '../jbibtex-bin_3.1.11+0.04b+3.0-0.1_amd64.deb' にパッケージ 'jbibtex-bin' を構築しています。
dpkg-deb: '../jmpost_3.1.11+0.04b+3.0-0.1_amd64.deb' にパッケージ 'jmpost' を構築しています。
 dpkg-genchanges -b >../ptex-bin_3.1.11+0.04b+3.0-0.1_amd64.changes
dpkg-genchanges: binary-only upload - not including any source code
dpkg-buildpackage: binary only upload (no source included)
Now running lintian...
W: jmpost: binary-without-manpage usr/bin/pmakempx
W: jbibtex-bin: copyright-without-copyright-notice
Finished running lintian.
```

無事できました。

8.9 References

- 第 41 回東京エリア Debian 勉強会資料(2008 年 6 月)。パッチシステムなどの使い方の詳細はこちらにまとまっています。
- dpkg-source(1)(man 1 dpkg-source)。3.0 (quilt) などのソースフォーマットの詳細。
- http://wiki.debian.org/ReleaseGoals/NewDebFormats
- http://release.debian.org/squeeze/goals.txt。 squeeze release goals.

Debian 勉強会資料

2010 年 3 月 20 日　　初版第 1 刷発行
東京エリア Debian 勉強会（ 編集・ 印刷・ 発行）