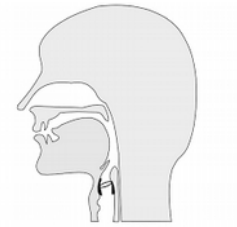
Распознавание речи.
Практическое знакомство с пакетом CMU Sphinx

Цель работы – ознакомиться с построением системы распознавания речи с использованием открытых данных и программного пакета CMU Sphinx (**http://cmusphinx.sourceforge.net**).

В процессе выполнения будут достаточно подробно рассмотрены следующие вопросы:
- обучение модели и настройка её параметров;
- оценка качества работы системы;
- улучшение акустической и языковой моделей.

Отчет по лабораторной работе должен включать:
- демонстрацию записанного тестового корпуса;
- результаты работы моделей на тестовых данных;
- защиту (ответ на 2 случайных контрольных вопроса).

ЧАСТЬ 1. Теория



CMU Sphinx – современный и весьма популярный пакет для разработки систем распознавания речи, на котором можно реализовать как высокоточные системы управления голосовыми командами, так и системы распознавания слитной речи с большим словарем.

Стоит отметить, что распознавание речи – не самая простая задача. Решения, которые мы имеем на сегодняшний день, стали возможными благодаря многолетним исследованиям в самых разных областях науки: обработке сигналов, лингвистике, фонетике, информатике, статистике и машинному обучению.

Настоящая работа ставит перед собой задачу лишь ознакомить с практическими аспектами этой занимательной науки.

Перед выполнением требуется самостоятельно прочитать следующие ресурсы:
- **презентацию theory/speech_recognition_fundamentals.pdf**
- **методичку по основным командам linux theory/linux_must_know.txt уметь отвечать на вопрос, для чего какая команда нужна**
- **заметку блога компании Яндекс (материал включает модели на нейронных сетях, с которыми в данной работе мы не столкнемся, но понимание которых важно): https://habrahabr.ru/company/yandex/blog/198556/**

Все описанное в практической работе является дружелюбной адаптацией официальной документации Sphinx (http://cmusphinx.sourceforge.net/wiki/tutorial), перенесенной на небольшой корпус русского языка и дополненной упрощающими (надеюсь) жизнь скриптами и материалами.

ЧАСТЬ 2. Изучение структуры проекта

Для обучения акустической и языковой моделей мы будем использовать данные проекта www.voxforge.org – аудиокниги, записанные волонтерами на нескольких языках.

Перед работой рекомендуется ознакомиться с детальным описанием процесса обучения:
http://cmusphinx.sourceforge.net/wiki/tutorialam

Замечание: при работе с виртуальной машиной (ВМ) удобно активировать режим копии-вставки между основной опереционной системой и ВМ.
Для активации в VirtualBox зайти в *Machine* → *Settings* → *Advanced* и установить *Shared Clipboard & Drag'n'Drop* на *bidirectional*

Материалы к данной работе находятся в папке lab_sphinx на рабочем столе распространяемой виртуальной машины.

Папка **lab_sphinx** содержит:
- настоящее руководство (**lab_sphinx/readme.pdf**)
- теоретические материалы (**lab_sphinx/theory**)
- необходимые для выполнения работы скрипты (**lab_sphinx/scripts**)
- базу (корпус) русских аудио книг (**lab_sphinx/ru_base**)
- исходный код модулей sphinx (**lab_sphinx/sphinx**)

Замечание: папка sphinx создается при установке, описанной в приложении. Если вы работаете с виртуальной машины, все программы должны быть у вас уже установлены.

Изучите состав папки **lab_sphinx/ru_base**.

Убедитесь, что все данные на месте. Во-первых, у вас должен быть файл **lab_sphinx/ru_base/lm_train_data.txt** с текстами для тренировки языковых моделей. Во-вторых, должна быть непустая папка **lab_sphinx/ru_base/wav,** в каждой директории которой находятся данные отдельных дикторов (их около 300). Для каждого диктора есть файл **etc/PROMPTS** с транскрипцией и **.wav** файл с соотвествующим именем.

Рекомендуется поизучать данные и послушать несколько аудио прежде чем переходить к следующему этапу работы.

Внимание: некоторые текстовые файлы, с которыми мы будем работать – достаточно большие. Читать их рекомендуется только с помощью команды *more* командной строки.

ЧАСТЬ 3. Подготовка данных

Наверно самым важным шагом в создании любой системы машинного обучения является аккуратная подготовка данных. К этому шагу нужно отнестись максимально ответственно.

3.1. Подготовка данных, словаря и языковых моделей

На данном этапе проект должен содержать следующие директории (команда ls */):

```
ru_base/:
    lm_train_data.txt  wav
scripts/:
    install_sphinx.sh  prepare_data.sh  prepare_lang_large.sh
    prepare_lang.sh  prepare_lang_test.sh  words2prons.py
sphinx/:
    cmuclmtk  pocketsphinx  sphinxbase  sphinxtrain
```

Важно: убедитесь, что содержимое папки ru_base соответствует описанному в **п.2.1**
Также убедитесь, что все необходимые программы работают, как описано в **п.2.2**

Откройте скрипт **prepare_data.sh** и изучите его работу. Наша задача – подготовить данные для обучения и тестирования моделей. Для этого у нас есть папка **ru_base/wav**, содержащая аудио файлы и транскрипции к ним. Тестовые данные скрипт возьмет из папки **test-speaker** (пока что оставим их без изменения).

Убедитесь, что вы находитесь в корне папки эксперимента и запустите скрипт из командной строки

```
cd ~/Desktop/lab_sphinx
bash scripts/prepare_data.sh
```

В паке **ru_base** теперь появится папка **etc**, содержащая несколько файлов, без которых ничего не получится. Просмотрите их содержимое:

- ***etc/ru_base_{train,test}.fileids*** – пути к аудио файлам
- ***etc/ru_base_{train,test}.transcription*** – транскрипции

```
# важно: транскрипты и id должны идти в одном и том же порядке. К счастью, эту работу за вас делает скрипт
```

- ***etc/ru_base_train.text*** – тексты тренировочных данных. По ним будем делать словарь и базовую языковую модель.

Создадим словарь и языковую модель. Если не помните, что это, просмотрите еще раз слайды из папки theory. Нужно понимать, зачем они нужны и уметь это объяснить

Изучите содержимое скрипта prepare_lang.sh и запустите его **из основной папки**

```
cd ~/Desktop/lab_sphinx
bash scripts/prepare_lang.sh
```

Просмотрите созданный скриптом словарь **ru_base/etc/ru_base.dic**. Создание словаря произношений – отдельная задача. К счастью, для русского языка правил не так много. Тем не менее, вы можете заметить, что произношения некоторых слов не идеальны. Скрипт также создал языковую модель **ru_base/etc/ru_base.lm** – триграммы. Файл содержит слова, биграммы и триграммы с вероятностями (в логарифмической шкале). Слева записывается вероятность последовательности, справа – вес back-off, использующийся для сглаживания. Для интересующихся деталями подробное описание можно почитать тут https://web.stanford.edu/class/cs124/lec/languagemodeling.pdf

В программе поиска (декодере) будет использоваться сжатая версия языковой модели для улучшения производительности. Это файл с расширением DMP

Замечание: если файлы языковых моделей не были созданы, что-то пошло не так с установкой cmuclmtk. Попробуйте переустановить самостоятельно, следуя описанию по ссылке http://cmusphinx.sourceforge.net/wiki/cmuclmtkdevelopment

Если не выходит, свяжитесь с автором

Скрипт так же создаст 2 списка фонем (юнитов):
- ***ru_base/etc/ru_base.phone*** – речевые фонемы
- ***ru_base/etc/ru_base.filler*** – заполняющие фонемы (у нас только тишина, может быть еще, например, шум, музыка, смех, и т.п.)

3.2. Запись тестовых данных

Основываясь на предположении о том, что любому заинтересованному студенту непременно захочется потестировать систему на своем голосе, а так же на том, что незаинтересованных среди нас с вами здесь нет, **обязательной частью лабораторной является создание собственной тестовой базы.**

Задача – изменить папку **ru_base/wav/test-speaker** под себя

Проще всего начать с файла ***ru_base/wav/test-speaker/etc/PROMPTS***. Фразы можно оставить те же, можно придумать свои. Отметим, что **на первом этапе система не сможет распознавать слова, которых не было в изначальной транскрипции** (т.к. система пока не знает для них произношения, а языковая модель не знает, с какой вероятностью их ожидать).

Изначальная тестовая выборка была составлена с неким хитрым умыслом: первые несколько фраз довольно просты – многие из них даже повторяются в тренировочных текстах (но произнесены другими дикторами).

test-speaker/mfc/ru_0061 Гусев и Маша жили в одной комнате в когда-то роскошном огромном теперь заброшенном доме

Последние несколько фраз – сложные, т.к. в них практически бесполезна языковая модель, обученная на речи адекватного русскоговорящего человека.
Согласитесь, вряд ли в текстах книжек будет что-то в духе:

test-speaker/mfc/ru_1006 нимфа да она всё выглядывала глубины города и еще зовёт её зубр знаю точно

Итак, в файле ***ru_base/wav/test-speaker/etc/PROMPTS*** мы можем **добавить/удалить/заменить** транскрипции любых текстовых фраз, которые вам предстоит записать. Желательно пока что использовать слова из словаря, **лучше в не самой логичной последовательности (иначе слишком просто)**. Первая колонка содержит имя соответствующего файла. Цифры в имени можно ставить любые

Теперь идем в папку **ru_base/wav/test-speaker/wav**
В ней находятся записанные автором аудио. Их нужно удалить

```
cd ~/Desktop/lab_sphinx/ru_base/wav/test-speaker/wav/
rm -rf *.wav
```

По аналогии нужно **записать файлы** и назвать их именами, соответствующими транскрипту: ru_0052.wav, ru_0054.wav, и т.д.

Важно! файлы должны быть mono, 16kHz, PCM, 16bit LE

В linux очень удобно делать такие простые записи командой arecord.

```
cd ~/Desktop/lab_sphinx/ru_base/wav/test-speaker/wav/
arecord  -r 16000 -c 1 -f S16_LE  ru_0052.wav
# говорим, жмем Ctr+C для остановки
# для прослушивания запускаем
play ru_0052.wav
```

Для любителей оконных редакторов можно воспользоваться, например, **Audacity** (бесплатна как под Windows, так и под linux)

```
sudo add-apt-repository ppa:ubuntuhandbook1/audacity
sudo apt-get update
sudo apt-get install audacity
```

В случае с аудасити не забудьте установить параметры **16kHz mono** и сохранить как **wav** командой **export**.

Для проверки совместимости с sphinx можно использовать команду консоли ''file''

```
file ru_base/wav/test-speaker/wav/ru_0052.wav
> ru_base/wav/test-speaker/wav/ru_0052.wav: RIFF (little-endian) data, WAVE audio,
Microsoft PCM, 16 bit, mono 16000 Hz
```

Данные готовы, можно начинать эксперименты!

Примечание: тем, кто работает с виртуальной машины, возможно, придется столкнуться с нерабочим динамиком или микрофоном. Должно помочь следующее:
- выключить виртуальную машину
- задать в свойствах settings → audio → enable audio
- выбрать драйвер PulfhfseAudio и системный контроллер типа Intel HD audio (если не доступен, можно попробовать что предлагается и потестировать разные варианты драйвера, у меня cfhf ICH AC97 не работала запись)
- запустить машину
- настроить уровень громкости в меню *sound settings* → *recording*

На крайний случай остается решение записать файлы где угодно и перенести их на виртуальную машину через флешку или Интернет.

После того, как данные для тестового диктора переделаны, необходимо “пересобрать” файлы нашего корпуса. Для этого возвращаемся в исходную папку проекта и запускаем два ранее выполненных скрипта

```
cd ~/Desktop/lab_sphinx
bash scripts/prepare_data.sh
bash scripts/prepare_lang.sh
```

Теперь файлы ***etc/ru_base_test.fileids*** *и* ***etc/ru_base_test.transcription*** относятся к новомым тестовым данным. Отметим, что тестовые данные при тренировке не используются (за этим проследил скрипт). Это важно, т.к. система должна работать на новых для нее данных.

4. Обучение акустической модели

Обучение и настройка акустической модели – не самая простая задача. Sphinx обладает огромным количеством настроек и алгоритмов обучения, позволяющих сильно увеличить точность распознавания.

Нам скорее важны основы самого процесса и его этапы. Каждый из этапов создаст свою папку в logdir. Опишем их вкратце и далее разберем каждый из этапов подробнее.

Основные этапы обучения акустической модели sphinx:
000.comp_feat - вычисление кепстральных признаков MFCC
20.ci_hmm - обучение модели на контекстно-независимых фонемах
30.cd_hmm_untied - обучение модели трифонов
40.buildtrees - построение дерева фонем для объединения близких по звучанию трифонов
45.prunetree – уменьшение числа трифонов (кластеризация)
50.cd_hmm_tied – обучение модели с кластерами трифонов
decode - распознавание тестовой выборки и оценка ошибки (Word Error Rate)

4.1. Создание и изучение конфигурационного файла тренировки модели

Для начала работы нужно создать конфигурационный файл для системы обучения sphinxtrain. Вы наверно уже догадались, что тут нам снова поможет скрипт.

Зайдите в папку с данными

```
cd ~/Desktop/lab_sphinx/ru_base
```

Из этой папки запустите

```
sphinxtrain -t ru_base setup
```

Скрипт создаст файл **etc/sphinx_train.cfg**
Давайте познакомимся с некоторыми его частями поподробнее, т.к. он содержит основные настройки, которые необходимо менять для настройки моделей и качества распознавания.

То, что мы в дальнейшем будем изменять, помечено желтым цветом

Пути к файлам проекта и исполняющим программам
Config начинается с указания путей к файлам нашего проекта для исполнительных программ, а так же путей к этим программам.

Параметры обработки сигнала
В нашем случае сигнал достаточно стандартный, но на практике бывает нужно изменить параметры извлечения признаков и фильтрации:

```
$CFG_WAVFILE_SRATE = 16000.0; # частота дискретизации (8кГц для телефонных записей)
$CFG_NUM_FILT = 25;       # - число фильтров (лучше 15 для телефонной речи)
$CFG_LO_FILT = 130;       # - частота фильтра нижних частот
                          # (например, можно срезать фонящую составляющую)
$CFG_HI_FILT = 6800;      # - частота фильтра верхних частот (не превышает SRATE/2
                          # по понятным Котельникову, Найквисту и вам причинам)
$CFG_VECTOR_LENGTH = 13; # - число кепстральных признаков (например, для
                          # идентификации диктора используют 19-20 признаков
```

Настройки дополнительной обработки признаков (например, нормализация среднего), параметров тренировки модели (количество итераций алгоритма обучения Baum-Welch) модели с оптимизацией MMIE (максимизации критерия взаимной информации) – дискриминативным обучением, которое позволяет прилично повысить точность, но требует слишком большого количества настроек и потому нами опускается в данной работе.

Пути к файлам корпуса, которые мы создавали на ранних этапах

```
$CFG_DICTIONARY     = "$CFG_LIST_DIR/$CFG_DB_NAME.dic";
$CFG_RAWPHONEFILE   = "$CFG_LIST_DIR/$CFG_DB_NAME.phone";
$CFG_FILLERDICT     = "$CFG_LIST_DIR/$CFG_DB_NAME.filler";
$CFG_LISTOFFILES    = "$CFG_LIST_DIR/${CFG_DB_NAME}_train.fileids";
$CFG_TRANSCRIPTFILE = "$CFG_LIST_DIR/${CFG_DB_NAME}_train.transcription";
$CFG_FEATPARAMS     = "$CFG_LIST_DIR/feat.params";
```

Настройки параллельных вычислений

Если вы работаете не с виртуальной машины и если у вас больше 1 CPU
Первый параметр уже сейчас можно смело поменять на **“Queue::POSIX”**
число 1, соответственно, меняем на число CPU
Это значительно ускорит тренировку

```
$CFG_QUEUE_TYPE = "Queue";
$CFG_NPART = 1;
$DEC_CFG_NPART = 1;
```

Настройки классификатора

Число Гауссиан в смеси для каждой фонемы (или сенона)

```
$CFG_FINAL_NUM_DENSITIES = 8;
```

Нужно ли тренировать контекстно-зависимые модели.
Начинать мы будем как раз без них:

```
$CFG_CD_TRAIN = 'yes';
```

Количество сенонов – кластеров контекстно–зависимых фонем. Чуть позже об этом.

```
$CFG_N_TIED_STATES = 200;
```

Настройки декодера

После выполнения всех наших экспериментов, будут созданы папки для каждой модели. Сюда можно будет просто прописать, на какой модели считать точность

Какую акустическую модель использовать декодеру после тренировки

```
$DEC_CFG_MODEL_NAME = "$CFG_EXPTNAME.cd_${CFG_DIRLABEL}_${CFG_N_TIED_STATES}";
```

Вес языковой модели (насколько при распознавании будет влиять акустика)

```
$DEC_CFG_LANGUAGEWEIGHT = "10";
```

Путь к файлу языковой модели

```
$DEC_CFG_LANGUAGEMODEL  = "$CFG_BASE_DIR/etc/${CFG_DB_NAME}.lm.DMP";
```

Настройки **beam search** (эвристического поиска) и штрафа за вставку слов, которые мы трогать не будем

```
$DEC_CFG_BEAMWIDTH = "1e-80";
$DEC_CFG_WORDBEAM = "1e-40";
$DEC_CFG_WORDPENALTY = "0.2";
```

4.2. Тренировка небольшой модели с контекстно-независимыми фонемами

Мы начнем с создания самой простой модели. Каждая фонема будет представляться в виде **3 состояний скрытой Марковской модели (CMM)**, каждое из которых будет моделироваться смесью **Гауссиан (если сейчас ничего не понятно, посмотрите еще раз слайды)**. Такой подход не учитывает ко-артикуляцию, или эффект различия акустических признаков фонем в разных контекстах (например для звука “а” в словах “сад” и “фан” признаки будут отличаться довольно сильно). С другой стороны, это упрщение позволит достаточно быстро обучить небольшую модель, которую легко встраивать в системы с малым количеством памяти и вычислительных ресурсов.

Создайте копию текущего конфигурационного файла (на всякий случай)

```
cd ~/Desktop/lab_sphinx/ru_base
cp etc/sphinx_train.cfg etc/sphinx_train.cfg.bkp
```

Открываем ~/Desktop/lab_sphinx/ru_base/etc/sphinx_train.cfg и меняем следующие строки (для правки файла удобно пользоваться командой gedit)
(# отвечает за комментарий)

```
# $CFG_CD_TRAIN = 'yes';
$CFG_CD_TRAIN = 'no';
# $DEC_CFG_MODEL_NAME = "$CFG_EXPTNAME.cd_${CFG_DIRLABEL}_${CFG_N_TIED_STATES}";
$DEC_CFG_MODEL_NAME = "$CFG_EXPTNAME.ci_${CFG_DIRLABEL}";
```

Теперь нам остается, **находясь в папке ru_base, запустить тренировку**

```
sphinxtrain run
```

Можно немного передохнуть – тренировка займет от нескольких минут до получаса в зависимости от мощности машины. При тренировке будут появляться сообщения ERROR – на данном этапе они не страшны.

4.3. Анализ результатов тренировки первой модели

Если все прошло удачно, скрипт выдаст что-то в духе

```
SENTENCE ERROR: 41.2% (7/17)   WORD ERROR RATE: 25.1% (49/195)
```

В данном случае, это говорит о том, что в 41% предложений сделана хотя бы одна ошибка. Если смотреть по словам (в моем случае их 195), то из них правильно было распознано 55.

Давайте посмотрим, что еще насоздавал Sphinx во время тренировки первой модели:
- файл отчета ru_base.html, в котором сохраняются истории всех запусков sphinxtrain. Его можно отрыть из браузера и посмотреть, как прошел тот или иной шаг тренировки, какие запускались команды и где что-то можно поправить;
- файлы логов для каждого из этапов logdir.

Например, если вы увидели сообщение

```
ERROR: This step had 10 ERROR messages and 0 WARNING messages. Please check the log file for details.
```

То можно заглянуть в соответствующий лог, анализ которого скажет нам, что произошло

```
ERROR: "backward.c", line 421: Failed to align audio to trancript: final state of the search is not reached
ERROR: "baum_welch.c", line 324: wav/ru_0065 ignored
```

На самом деле, ничего страшного. Данная аудио запись и соответствующий транскрипт не получилось “выровнять”. Это говорит о том, что либо транскрипция, либо аудио, либо словарь неточны или зашумлены, либо модель пока еще не достаточно точна. Иногда бывает полезно послушать проблемные файлы. Если таких пропусков немного, система должна выучиться нормально.

В самом начале этой части мы говорили том, что обучение состоит из последовательного вызова внутренних скриптов и программ sphinx. В данном случае были выполнены три программы:
000.comp_feat - вычисление кепстральнхы признаков MFCC
20.ci_hmm - обучение модели на контекстно-независимых фонемах
decode - распознавание тестовой выборки и оценка ошибки (Word Error Rate)

4.4. Анализ натренированных моделей

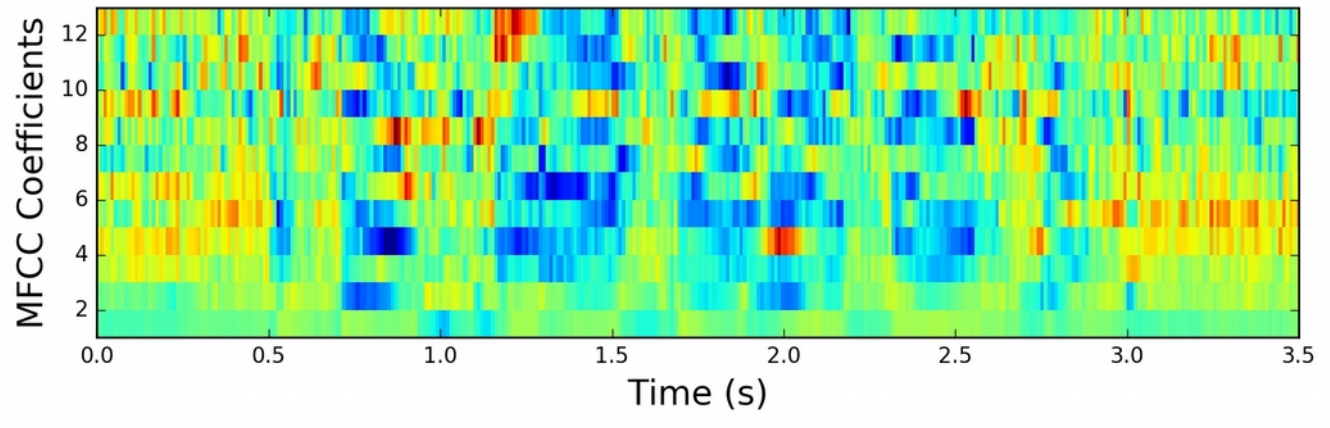
В результате, помимо логов, по которым вы конечно уже вдоволь нагулялись, создались **файлы признаков и моделей**. Признаки (MFCC для каждой аудио записи) находятся в папке **feat** – они записаны в бинарном формате. Прочитать их мы можем с помощью команды **sphinx_cepview**:

```
sphinx_cepview -d 13 -describe 1 -f feat/test-speaker/wav/ru_0052.mfc | more

    0: 57.397 -21.300 -11.883  20.390 -29.060  -6.600  -4.971 -17.877  -4.298  -0.001  -4.789 -10.140  11.017
    1: 46.706  -4.212   2.922  -8.208 -10.996  -0.331  -3.510   0.755   7.516 -10.791   4.334   7.510   7.109
    2: 47.283  -4.150  -0.394 -10.870  -6.460  -2.542  -5.947   1.350  16.483  11.696  12.321  15.527   9.837
    3: 47.332  -1.552   1.418  -4.305  -0.750   3.976  -2.256  -0.473   6.640  -1.902   8.814  15.854   8.186
    4: 47.607   0.499  -4.088  -2.182  -0.914   3.680  -0.479  12.805  15.848   1.988   9.720   5.255  -5.043
    5: 47.492  -4.342  -2.386  -8.035   2.640   2.288  -0.766   4.229  12.383   1.487  -7.022  -2.629   1.705
    6: 47.799  -2.363  -1.937 -11.272   2.426  -0.100  -6.280  -3.471   8.775   1.214   7.396  10.014  10.158
    7: 47.906  -2.933  -1.901 -12.021  -3.020  -6.458 -12.269  -5.026  13.506   2.860   0.499  -7.437   6.979
```

В этой таблице столбцы – кепстральные признаки (их 13), строки – перекрывающиеся окна, по которым бралось преобразование Фурье (25мс с шагом 10мс). Чтобы освежить понимание признаков, см презентацию.

Знающим matlab не составит труда построить по этим данным кепструм одного из файлов

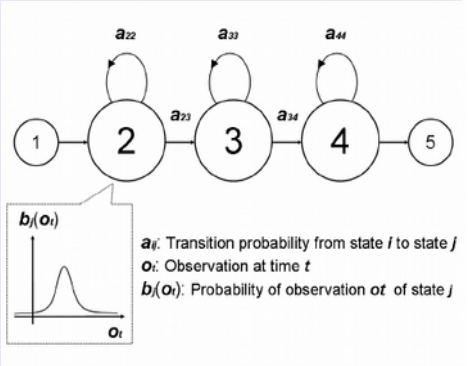


По этим признакам sphinx обучил скрытые Марковские модели (CMM) и заботливо уложил все необходимое в соответствующие папки.

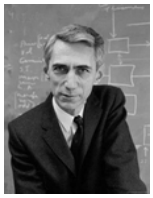
Сперва создается архитектура модели. Это текстовый файл
model_architecture/ru_base.ci.mdef
Давайте немного поизучаем, что внутри:

```
43 n_base     # в нашем языке всего 43 фонемы
0 n_tri       # сюда будут записываться контекстно-зависимые
трифоны. Пока что их нет

129 n_tied_state  # 3 состояния с Гауссовыми смесями (state ids)
129 n_tied_ci_state
43 n_tied_tmat
#
# Columns definitions
#base lft  rt p attrib tmat      ... state id's ...
  SIL   -   - - filler    0    0    1    2    N
   a0   -   - -    n/a    1    3    4    5    N
    b   -   - -    n/a    2    6    7    8    N
   bj   -   - -    n/a    3    9   10   11    N
    c   -   - -    n/a    4   12   13   14    N
   ch   -   - -    n/a    5   15   16   17    N
```



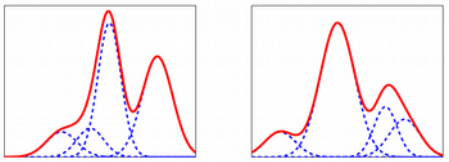
Проще говоря *(настолько просто, что любым ценителям точных определений, фанатам статистики и просто уважающим себя и двух парней справа людям стоит пропустить данный параграф и почитать книжку)*:
для каждой фонемы архитектурой задается 3 состояния, или вершины конечного автомата (начало фонемы, середина и конец), в каждом из которых будет задано вероятностное распределение в виде смеси Гауссиан (параметры которых мы учим в процессе тренировки). На этапе распознавания система будет оценивать, на какое распределение (на какую часть какой фонемы) похож текущий фрейм признаков (25 мс сигнала). Состояния скрытой Марковской модели задают длительность фонем, а также порядок. Таким образом, чтобы кусок сигнала был признан похожим на фонему “a0” из нашего словаря, минимум 3 фрейма должны быть похожими на распределения 3, 4 и 5 (согласно state_ids mdef файла). В Марковской модели недаром есть циклы – фонема может (и часто будет) звучать больше 3 фреймов.



Как уже было сказано ранее, обучение заключается в оценке параметров Гауссовых смесей и вероятностей перехода СММ. Начинается оно с предположения о равномерной сегментации фонем. Для каждой фонемы создаются одинаковые смеси Гауссиан и считается, что длительности фонем так же равномерно распределены. Начальная модель записывается в папку **model_parameters/ru_base.ci_cont_flatinitial**

Напомню, что каждая смесь задает функцию правдоподобия признаков фрейма o в момент t и состояния (state согласно mdef архитектуре) q. Параметры у этой функции (и соответствующие файлы модели) следующие:

$$P(\mathbf{o}_t|q_t = j) = \sum_{l=1}^{M} \omega_{jl}\mathcal{N}(\mathbf{o}_t|\boldsymbol{\mu}_{jl}, \boldsymbol{\sigma}_{jl})$$



means (μ) – средние значения для каждого MFCC признака
variances (σ) - дисперсия
mixture_weights (ω) – веса Гауссиан

Также в файле модели находятся
transition_matrices – вероятности цикла и перехода Марковской модели

Давайте посмотрим на изначальные параметры модели в файлах model_parameters/ru_base.ci_cont_flatinitial

```
printp -gaufn model_parameters/ru_base.ci_cont_flatinitial/means | more
```

Если команда printp не найдена, попробуйте выполнить из абсолютного пути. Похоже на баг установщика

```
~/Desktop/lab_sphinx/sphinx/sphinxtrain/src/programs/printp/.libs/printp -gaufn
model_parameters/ru_base.ci_cont_flatinitial/means | more
```

Выведет на экран что-то в духе:

```
param 129 1 1
mgau 0     # индекс из state ids предыдущего файла (на этом этапе каждой фонеме сопоставляется 1
распределение)
feat 0     # далее идут средние для состояния с индексом 0 (SIL)
           # имеем 39 средних значений: 13 для каждого параметра MFCC + 13 первых и вторых производных
density    0 -3.266e-02 -1.267e-02 -2.884e-03 -1.470e-02 2.961e-03 -1.099e-02 6.187e-04 8.596e-04
-2.154e-03 2.475e-03 -2.208e-03 -1.890e-03 5.259e-03 -2.663e-02 2.295e-02 5.553e-03 8
.893e-03 -5.822e-03 8.889e-03 8.958e-03 1.411e-02 -2.441e-03 6.562e-03 6.274e-03 -1.467e-04 -2.039e-03
-6.044e-04 4.556e-04 1.445e-03 -2.537e-03 -1.244e-03 -2.040e-03 -2.373e-03 -2.85
8e-03 4.516e-04 -5.864e-04 -1.588e-03 -5.911e-04 3.538e-04
mgau 1     # аналогичные параметры для второго состояния (середина SIL)
feat 0
density    0 -3.266e-02 -1.267e-02 -2.884e-03 -1.470e-02 2.961e-03 -1.099e-02 6.187e-04 8.596e-04
-2.154e-03 2.475e-03 -2.208e-03 -1.890e-03 5.259e-03 -2.663e-02 2.295e-02 5.553e-03 8
.893e-03 -5.822e-03 8.889e-03 8.958e-03 1.411e-02 -2.441e-03 6.562e-03 6.274e-03 -1.467e-04 -2.039e-03
-6.044e-04 4.556e-04 1.445e-03 -2.537e-03 -1.244e-03 -2.040e-03 -2.373e-03 -2.85
8e-03 4.516e-04 -5.864e-04 -1.588e-03 -5.911e-04 3.538e-04
...
```

Внимательный читатель / запускатель заметит, что параметры для различных фонем в этом файле одинаковы. Верно! Именно в этом и заключается идея инициализации модели. Сравните с файлом обученной модели **model_parameters/ru_base.ci_cont/means**

Теперь у нас мало того, что параметры все разные, так еще и каждому состоянию соответствуют целых 8 Гауссиан (каждая по 39 параметров)

Почему 8?
Простой ответ – потому что $CFG_FINAL_NUM_DENSITIES = 8; в конфиге.
Более логичный ответ – потому что так у модели есть возможность обобщать сильно различающиеся данные: например, акустические признаки одной и той же фонемы, произнесенной мужчиной и женщиной, сильно отличаются. 2 Гауссовы компоненты как раз решают эту проблему. Если тренировочных данных много, можно использовать более 100 компонент для охвата всевозможных вариаций звуков. В нашем случае 8 – в самый раз.

Аналогично можно посмотреть на параметры variances.

```
printp -gaufn model_parameters/ru_base.ci_cont_flatinitial/variances | more
```

Веса Гауссиан и матрицы вероятностей переходов состояний просматриваются немного другой командой

```
printp -mixwfn model_parameters/ru_base.ci_cont/mixture_weights | more
printp -tmatfn model_parameters/ru_base.ci_cont/transition_matrices | more
```

С весами вроде ежу понятно – каждой компоненте - свой вес, их сумма 1. А вот на матрицы переходов стоит посмотреть чуть внимательнее

```
…
tmat [4] # матрицы фонемы с полем tmat=4 из файла mdef (фонема /c/)
 8.243e-01 1.757e-01
          8.498e-01 1.502e-01
                    7.748e-01 2.252e-01
```

Эта матрица моделирует длительность фонем (в терминах CMM). Вспомните, что у нас есть 3 состояния, через которые должны пройти фреймы аудио сигнала с достаточно высоким значением функций правдоподобия. Сама функция правдоподобия при распознавании будет как раз умножаться на вероятность перехода. В данном случае, находясь в 1м состоянии (state id=12, если верить ранее просмотренному файлу архитектуры mdef), вероятность в нем остаться будет 8.2, а вероятность перейти в другое состояние – середина фонемы (где властвует уже другая функция правдоподобия) равна 1.8.

Для желающих копнуть еще глубже есть еще вот эта довольно объемная документация по содержимому различных файлов моделей CMU Sphinx:
http://www.speech.cs.cmu.edu/sphinxman/scriptman1.html

4.5. Более детальный анализ результата

Отлично! Если вы дошли до сюда, значит уже немного разбираетесь с linux, с акустическими признаками, моделями и конфигурационным файлом.

Честно, это уже большой шаг, который, судя по форуму CMU Sphinx, ежедневно мучает пару десятков человек с разных уголков Земного шара.

Наверно, пусть отдаленно, у вас возник вопрос:
а где, собственно, результат?

Точность систем распознавания речи традиционно измеряется мерой **Word Error Rate**, которая складывается из числа **замен (substitutions)**, **вставок (insertions)** и **удалений (deletions)**, которые нужно сделать, чтобы сопоставить результат распознавания и исходный транскрипт (расстояние Левинштейна).

Полный отчет теста находится в файле **result/ru_base.align.** Несколько примеров:

```
# Идеально распознанная фраза (чуть позже станет ясно, почему так получилось)
гусев и   маша жили в   одной комнате в   когда то  роскошном огромном теперь заброшенном доме  (wav-RU_0061)
гусев и   маша жили в   одной комнате в   когда то  роскошном огромном теперь заброшенном доме  (wav-RU_0061)
Words: 15 Correct: 15 Errors: 0 Percent correct = 100.00% Error = 0.00% Accuracy = 100.00%
Insertions: 0 Deletions: 0 Substitutions: 0

# Гораздо хуже
ГЛАЗ   СДЕЛАЛ грустно УЛЫБАЯСЬ ШОФЕР     гусев и   *** ЗАКРУЖИЛСЯ  (wav-RU_1004)
ВОЛОСЫ СИДЕЛ  грустно ***      НЕБОЛЬШАЯ гусев и   ЗА  КОНУСЕ      (wav-RU_1004)
Words: 8 Correct: 3 Errors: 6 Percent correct = 37.50% Error = 75.00% Accuracy = 25.00%
Insertions: 1 Deletions: 1 Substitutions: 4

# Совсем плохо
***  ВООБРАЖЕНИИ ОПЯТЬ УГЛЕКИСЛОТЫ ЗНАЧИТ ТАК         ВОТ    УБЕДИТЕЛЬНО  (wav-RU_1002)
НАДО ЖЕ          МНЕ   ПОД         ЭТИМ   УГЛЕКИСЛОТЫ ЗНАЧИТ МУЧИТЕЛЬНО   (wav-RU_1002)
Words: 7 Correct: 0 Errors: 8 Percent correct = 0.00% Error = 114.29% Accuracy = -14.29%
Insertions: 1 Deletions: 0 Substitutions: 7
```

Как видно, фразы с нестандартной грамматикой практически не распознались. Это происходит из-за того, что акустическая модель у нас на данном этапе совсем слабая и в основном распознавание получается за счет языковой модели. Хорошо распознанные фразы были взяты из тренировочных данных. Хотя мы и исключили тестируемого диктора из тренировки, языковая модель позволила безупречно отработать эти примеры. Именно поэтому управление роботом спокойной речью через хороший микрофон – уже хорошо решенная задача, пусть и требующая некоторых знаний для ее решения.

В следующей части мы постараемся улучшить акустическую модель, чтобы система максимально четко отрабатывала любые последовательности слов.

- Занесите полученные результаты в первую строчку таблицы отчета;

- Скопируйте файл ru_base.align в файл ru_base_ci.align

5. Улучшение точности акустической модели

5.1. Контекстно-зависимые трифоны

Вы наверно уже заметили, что предыдущая система, пусть и распознавала знакомые ей фразы достаточно успешно, не могла похвастаться точностью именно акустической части.

Серьезной проблемой моделирования речи является **многообразие (или variability)** акустических признаков одной и той же фонемы, произнесенных в разных условиях.
Большие проблемы создают:
- Пол и возраст дикторов (частично решается Гауссовыми смесями)
- Микрофон (частично решается обработкой сигнала – нормализацией среднего)
- Шум и реверберация (до сих пор не до конца ясно как решать)
- Коартикуляция (различие звуков в разных контекстах)

Как раз о коартикуляции пойдет речь в этой части работы. Раз звуки так отличаются в разных контекстах, **почему бы вместо фонем не рассматривать трифоны**: например, звуку /т/ соответствовали бы несколько моделей в зависимости от того, какие звуки находятся рядом: “**т**ук”, “**т**ам”, “лис**т**”, “рас**т**ение”. Идея хорошая, но таких сочетаний

было бы слишком много (кстати, сколько?), а значит для тренировки понадобились бы тысячи часов аудио.
На практике Sphinx работает с **сенонами** – сначала действительно учит большую модель для всех сочетаний трифонов, а затем группирует (кластеризует с использованием решающих деревьев) те, звучание которых похожи.

Давайте вернемся к нашему конфигурационному файлу.

```
# ставим флаг тренировки CD (context-dependent моделей)
# $CFG_CD_TRAIN = 'no';
$CFG_CD_TRAIN = 'yes';

# говорим декодеру в тесте использовать CD модель
# $DEC_CFG_MODEL_NAME = "$CFG_EXPTNAME.ci_${CFG_DIRLABEL}";
$DEC_CFG_MODEL_NAME = "$CFG_EXPTNAME.cd_${CFG_DIRLABEL}_${CFG_N_TIED_STATES}";
```

На самом деле предыдущая модель зря не пропадет. Тренировка моделей всегда происходит в несколько этапов, т.к. иначе алгоритм просто не сойдется (по большей части из-за предположения о равномерном начальном выравнивании и отсутствии транскрипций на уровне фонем и времени их звучания).

Первые два шага, помеченные звездочкой, мы уже выполнили.

*** 000.comp_feat** - вычисление кепстральных признаков MFCC
*** 20.ci_hmm** - обучение модели на контекстно-независимых фонемах
30.cd_hmm_untied - обучение модели трифонов
40.buildtrees - построение дерева для объединения близких по звучанию трифонов
45.prunetree – уменьшение числа трифонов (кластеризация)
50.cd_hmm_tied – обучение модели с кластерами трифонов
decode - распознавание тестовой выборки и оценка ошибки (Word Error Rate)

Чтобы продолжить тренировку с учетом изменений в конфигурационном файле, остается **перейти в папку ru_base и ввести простую команду**

```
sphinxtrain -f cd_hmm_untied run
```

При таком запуске сфинкс пропустит этап извлечения признаков и сразу перейдет к шагу **30.cd_hmm_untied**.

Можно пойти попить чаю. После тренировки появятся еще несколько записей в html логах, а также файлы моделей.

Скорее всего, результат, полученный после этого шага хоть немного лучше предыдущего.

Изучите файлы архитектуры моделей для реальных трифонов model_parameters/ru_base.cd_cont_untied/mdef и для сенонов. Обратите внимание на 3 колонки state id's. Напомню, что каждому state id (состоянию) ставится в соответствии набор из 8 Гауссиан (по 39 значений средних и 39 дисперсий каждая).

- Занесите полученные результаты из ru_base.align во вторую строчку таблицы
- Занесите в таблицу размер папок с ci и cd_200 моделей
- Скопируйте файл ru_base.align в файл ru_base_cd_200.align

5.2. Увеличение числа параметров

Не всегда, но часто увеличение числа параметров позволяет улучшить точность (если у нас достаточно данных). Попробуем увеличить число Гауссиан до 16 и число сенонов до 2000, изменив конфигурационный файл

```
# $CFG_N_TIED_STATES = 200;
$CFG_N_TIED_STATES = 2000;
# !!! тот что после $CFG_DIRLABEL = 'cont';
# $CFG_FINAL_NUM_DENSITIES = 8;
$CFG_FINAL_NUM_DENSITIES = 16;
```

Это последняя длительная тренировка, которую мы выполним в данной работе. Шаг с обучением первичной модели трифонов можно пропустить. Таким образом, команда из папки ru_base будет выглядеть следующим образом:

```
sphinxtrain -f prunetree run
```

На этот раз обучение будет долгим. Можно даже и на обед прерваться.

- Занесите полученные результаты в третью строчку таблицы, запишите также размер папки с моделью
- Аналогично внесите данные о размере получившейся модели
- Скопируйте файл ru_base.align в файл ru_base_cd_2000.align

6. Обучение языковой модели

6.1. Вес языковой модели

После того, как последняя акустическая модель построена, результат распознавания должен выглядеть уже более адекватным даже для сложных фраз.

Следующий этап – настройка языковой модели. Иногда это может дать сильный прирост точности. К тому же это займет гораздо меньше времени и усилий.

Дело в том, что **при распознавании вес той или иной гипотезы (фонемы / слова) складывается из функции правдоподобия (акустической модели) и вероятности слова в контексте (языковой модели), умноженной на некоторый вес.**

Если акустическая модель улучшается, а языковая – не очень, этот вес полезно уменьшить. Для этого в конфигурационном файле просто меняем значение одного параметра:

```
# $CFG_LANGUAGEWEIGHT = "11.5";
$CFG_LANGUAGEWEIGHT = "8";
```

И запускаем только распознавание

```
sphinxtrain -f decode run
```

- Попробуйте несколько значений в интервале 7-12

- Лучший результат занесите в 4ю строку таблицы

6.2. Расширение языковой модели

До сих пор и акустическая и языковая модели обучались нами на совсем небольшом корпусе (8 часов речи). Для сравнения, современные системы обычно учат на 100-1000 часах данных; google периодически публикует работы о системах, обученных на 10-100 тысячах часов. Транскрипция такого объема речи – очень дорогое удовольствие.

В то же время находить тексты для языковых моделей гораздо проще (википедия, новостные сайты, субтитры). Множество открытых ресурсов даже предлагают готовые очищенные бесплатные данные, измеряющиеся гигабайтами текстов. При грамотном использовании такие ресурсы могут сильно улучшить системы общего назначения (general purpose speech recognition), такие как google voice search.

Внимательный пользователь уже заметил довольно большой файл **lm_train_data.txt**, который был скачан с сайтов субтитров (на самом деле это меньше 0.1 части корпуса, предназначенная для демонстрации работы). Также в папке scripts лежит волшебный скрипт **prepare_lang_large.sh**, превращающий текст в языковую модель. Заодно он расширяет словарь, так что система теперь не ограничена только словами из малой тренировочной базы.

Изучите работу скрипта scripts/prepare_lang_large.sh, перейдите в корневую папку лабораторной работы, запустите скрипт и после его выполнения снова вернитесь в ru_base

```
cd ~/Desktop/lab_sphinx
bash scripts/prepare_lang_large.sh
cd ru_base
```

Теперь в папке etc у нас есть 2 языковые модели **ru_base.lm и ru_base_large.lm**, а также их бинаризованные версии .DMP.

```
! Открывать файл  ru_base_large.lm только из терминала командой (никаких окон, иначе все зависнет)
more etc/ru_base_large.lm
```

Давайте укажем декодеру путь к новой языковой модели

```
# $DEC_CFG_LANGUAGEMODEL  = "$CFG_BASE_DIR/etc/${CFG_DB_NAME}.lm.DMP";
$DEC_CFG_LANGUAGEMODEL  = "$CFG_BASE_DIR/etc/${CFG_DB_NAME}_large.lm.DMP";
# можно повысить вес языковой модели, если на предыдущем шаге он был понижен
```

и перезапустим распознавание

```
sphinxtrain -f decode run
```

Не знаю, как у вас, у меня точность повысилась порядочно (с 22.6 до 18.5 WER).
Если разницы не видно, попробуйте удалить папку logdir/decode и перезапустить

- Сравните размер (в Мб) бинарных версий этих моделей и занесите в первую и пятую строки таблицы в последнюю колонку
- На сколько увеличился объем словаря?
- Запишите результат распознавания в пятную строку
- Скопируйте файл ru_base.align в файл ru_base_cd_2000_llm.align

6.3. Немного жульничества

А что если нам нужна довольно простая система, которая как можно точнее распознает довольно узкий набор текстов (или команд)?

Для этой задачи обычно используют другой вид языковых моделей – **грамматики или режим поиска ключевых слов**, реализованные в Sphinx. В рамках этой работы упростим задачу и просто по аналогии с предыдущим шагом смешаем тексты тренировочной выборки с текстами (но не аудио) из тестовых данных.

Отвечает за это скрипт **scripts/prepare_lang_test.sh**
Он создаст языковую модель ru_base_test.lm

Прикрутите модель к декодеру и повторите эксперимент. У меня ошибка теперь 1.5%

- Проделайте аналогичный предыдущему эксперимент с моделью из scripts/prepare_lang_test.sh
- Запишите результат распознавания в шестую колонку
- Скопируйте файл ru_base.align в файл ru_base_cd_2000_tlm.align

Можно сказать, что система теперь работает без ошибок. Но не стоит забывать, что задача, которую мы ей поставили, для современного мира уже в принципе достаточно давно решена: последний пример – это фактически распознавание читаемой речи с малым словарем.

Попробуйте, например, записать те же тестовые данные с играющей на фоне музыкой, запишите их быстрым темпом, как если бы вы участвовали в дебатах, в ванной комнате на расстоянии 1м от микрофона, создав легкую реверберацию, и вы обязательно поймете, что не все так уж и гладко.

Что дальше?

После того, как модель готова, систему можно настраивать под свои нужды, встраивать в приложения на android, запускать на Raspberry PI, управлять роботом в ROS, запускать онлайн сервер транскрипций наподобие google.

Pocketsphinx обладает всеми этими возможностями благодаря достаточно большому сообществу программистов, пишущих открытый код.

Следующей частью лабораторной работы будет использование pocketsphinx и обученных нами моделей для управления мобильным роботов в ROS.

Bonus

Распознавание записанных файлов – штука полезная для отладки и написания научных статей. Наверно вам хотелось бы поиграть с real-time распознаванием с микрофона. Самой простой реализацией такого взаимодействия является режим поиска ключевых слов в потоке аудио сигнала, реализованный в pocketsphinx_continuous.

Заинтересованным оставляю на самостоятельную работу раздел Keyword lists
из документации http://cmusphinx.sourceforge.net/wiki/tutorialIm

Модель, кстати, можно использовать и ту, что мы обучили для русского языка (смотрите также logs/decode/* чтобы разобраться, куда прописывать пути к моделям)

Отчет по работе студента _____

1. Сводная таблица результатов

№	Модель	Результат (WER/Ins/Del/Sub)	Размер акустической модели, Мб (model_params/mdl)	Размер языковой модели, Мб (etc/ru_base*.DMP)
1	CI, 8 DEN			
2	CD, 8 DEN, 200 sen			----
3	CD, 16 DEN, 2000 sen			----
4	CD, 16 DEN, 2000 sen + LW tune		----	----
5	CD, 16 DEN, 2000 sen + lm_large		----	
6	CD, 16 DEN, 2000 sen + lm_test		----	

2. Скопировать папки model_parameters и wav/test-speaker и файлы ru_base.align для каждого из экспериментов

ПРИЛОЖЕНИЕ. Установка необходимых программ

На выданной виртуальной машине были заранее предустановлены все необходимые программы. В случае если появится необходимость установить систему на свою машину, можно воспользоваться данным руководством.

Для выполнения работы использовались следующие программы:
- **sphinxbase** – основные пакеты CMU Sphinx
- **sphinxtrain** – пакет для тренировки акустических моделей
- **pocketsphinx** – система распознавания (декодер)
- **cmuclmtk** – система тренировки языковых моделей

Примечание: установка – достаточно трудоемка часть работы. Для упрощения были созданы установочные скрипты, которые не всегда работают стабильно. После установки требуется проверить работоспособность различных модулей и при возникновении проблем попытаться перезапустить установку недостающих пакетов вручную.

Находясь в папке lab_sphinx, запустите из терминала:

```
bash scripts/install_sphinx.sh
source ~/.bashrc
```

Примечание: если скрипт не запускается, можно просто копировать команды из его содержимого в терминал

После ввода пароля администратора (123 в случае виртуальной машины) система должна установить все необходимое.
Установка займет приличное количество времени. Если все прошло нормально, в папке с лабораторной работой появится каталог sphinx, а из консоли должны запускаться нижеследующие команды.

Проверьте наличие установленных программ. Ниже приведены команды, которые система должна находить при вводе части команды и нажатии клавиши TAB. При выполнении обычно возникает справка, или просто что-то отличное от сообщения ошибки ненайденной команды.

```
sphinx_ # нажатие TAB должно печатать:
  sphinx_cepview     sphinx_fe          sphinx_lm_convert  sphinx_pitch
  sphinx_cont_seg    sphinx_jsgf2fsg    sphinx_lm_eval
  (если нет, что-то не так с sphinxbase)

sphinxtrain -h         # если не находит, переустанавливаем sphinxtrain
pocketsphinx_batch     # если не находит, переустанавливаем pocketsphinx
text2wfreq -help       # если не находит, проблема с cmuclmtk (см ниже)
```

Известные проблемы

1. linux видит программы, но выводит ошибку вида:
```
sphinx_fe: error while loading shared libraries: libsphinxbase.so.3: cannot open
shared object file: No such file or directory
```

Решение: добавить папку lib в область видимости linux, выполнив:
```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib/
```

Чтобы проблема не возникала при открытии нового терминала, эту строчку нужно добавить в файл ~/.bashrc
открыть в текстовом редакторе (например **nano ~/.bashrc**) и дописать в конец файла
```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib/
```
нажать Ctrl+X

2. linux не видит программы, хотя вроде все сделано:
Возможно, программы были скомпилированы, но linux их не видит. Можно попробовать запустить напрямую:
```
/usr/local/bin/sphinx_fe
/usr/local/bin/sphinxtrain
/usr/local/bin/pocketsphinx_continuous
/usr/local/bin/wfreq2vocab
```

Если эти команды на месте, скорее всего, проблема в переменной PATH системы linux
Выполнить
```
export PATH=$PATH:/usr/local/bin/
```
Открываем ~/.bashrc и добавляем
```
export PATH=$PATH:/usr/local/bin/
```

3. Ничего не помогло
Установка программ в linux может преподносить сюрпризы. При возникновении ошибок не бойтесь их “гуглить”. Зачастую решением является доустановка какого-то недостающего пакета с помощью ***sudo apt-get <имя_пакета>***

4. Все еще не помогло (advanced level)
Если что-то с установкой все еще идет не так, ознакомьтесь с
http://jrmeyer.github.io/installation/2016/01/09/Installing-CMU-Sphinx-on-Ubuntu.html

5. И так не помогло
Если не помогло, обращайтесь к автору работы.
Дальнейшая работа будет невозможной при неправильном функционировании любой из вышеперечисленных программ.