Kaldi Setup Documentation

Prerequisites Check and Installation

Tools Installation

To check the prerequisites for Kaldi, run:

```
extras/check_dependencies.sh
```

Review the output carefully for any system-level installations you might need. If your system's default C++ compiler is not supported, you can set the CXX environment variable to use another compiler:

```
CXX=g++-4.8 extras/check_dependencies.sh
```

After confirming the prerequisites, run:

```
make
```

This will install ATLAS headers, OpenFst, SCTK, and sph2pipe. Ensure your C++ compiler supports C++11, such as:

- `g++ >= 4.7`
- `Apple clang >= 5.0`
- `LLVM clang >= 3.3`

If your system's default compiler does not support C++11, specify a compliant compiler as shown:

```
make CXX=g++-4.8
```

To speed up the build process using multiple CPUs, use the `-j` option:

```
make -j 4  # Adjust the number based on the available CPUs
```

For additional components used by example scripts, there are various scripts in `extras/` that you may need to run based on the example requirements.

Source Installation

These steps are valid for UNIX-like systems (e.g., Linux distributions, Darwin, Cygwin).

Make sure you have completed the installation steps in `tools installation` (compiling OpenFst; obtaining ATLAS and CLAPACK headers).

Run the following commands for installation:

```
./configure --shared
make depend -j 8
make -j 8
```

The `-j 8` option runs the build in parallel using 8 jobs, which may be too many for systems with fewer cores.

For more information, see the Kaldi documentation and refer to "The build process (how Kaldi is compiled)".

Path Setup

In `backend/path.sh`, replace `/path/to/kaldi` with your absolute path to the Kaldi model:

```
export KALDI_ROOT=/path/to/kaldi
export PATH=$PWD/utils/:$KALDI_ROOT/tools/openfst/bin:$PWD:$P
ATH
[ ! -f $KALDI_ROOT/tools/config/common_path.sh ] && echo >&2
"The standard file $KALDI_ROOT/tools/config/common_path.sh is
not present -> Exit!" && exit 1
. $KALDI_ROOT/tools/config/common_path.sh
```

```
export LC_ALL=C

# Python version for LM training and G2P-related scripts
PYTHON='python2.7'

# Paths for optional parts of the recipe (text normalization
for LM-training)
FEST_ROOT=tools/festival
NSW_PATH=${FEST_ROOT}/festival/bin:${FEST_ROOT}/nsw/bin
export PATH=$PATH:$NSW_PATH

# SRILM needed for LM model building
SRILM_ROOT=$KALDI_ROOT/tools/srilm
SRILM_PATH=$SRILM_ROOT/bin:$SRILM_ROOT/bin/i686-m64
export PATH=$PATH:$SRILM_PATH

# Sequitur G2P executable path
sequitur=$KALDI_ROOT/tools/sequitur-g2p/g2p.py
sequitur_path="$(dirname $sequitur)/lib/$PYTHON/site-package
s"

# LM training corpus directory
LM_CORPUS_ROOT=./lm-corpus
export PYTHONUNBUFFERED=1

export PATH=/path/to/kaldi/tools/python:${PATH}
export LIBLBFGS=/path/to/kaldi/tools/liblbfgs-1.10
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH:-}:${LIBLBFGS}/lib/.
libs
export SRILM=/path/to/kaldi/tools/srilm
export PATH=${PATH}:${SRILM}/bin:${SRILM}/bin/i686-m64
```

Backend Directory

To start the Kaldi backend, follow the instructions provided in the `backend` folder.

Frontend Directory

All the frontend scripts and HTML files are placed in the `frontend` folder.

Testing the Setup

In `backend/path.sh`, replace `/path/to/kaldi` with your absolute path to the Kaldi

To verify if the model is working correctly:

1. Run the frontend and backend.
2. After signing up, record your audio, then upload and save it by answering the complete question.
3. The audio should be saved in the `backend/uploaded_audio` directory.
4. GoP scores will be saved in `backend/Goodness-of-Pronounciation/GoP_scores.txt`
5. You should see processing on the interface, and within 2-3 seconds, the model should respond with either `Wrong` or `Correct` based on the GoP scores. If you encounter `Error`, recheck your installation setup.

Reference code in `backend/main.py`

```
from fastapi import FastAPI, File, Form, UploadFile
from fastapi.middleware.cors import CORSMiddleware
from fastapi.responses import JSONResponse
from pydantic import BaseModel
import os
import pickle
import numpy as np
from datetime import datetime
import logging
```

```
logging.basicConfig(level=logging.INFO)

app = FastAPI()

# Configure CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Function to create data folders and files
def create_data_folder(audio_path, qid, id_to_sent, data_dir
='data/data_temp'):
    os.makedirs(data_dir, exist_ok=True)
    with open(data_dir+'/wav.scp','w',encoding='utf8') as f:
        f.write('temp cat '+audio_path+' | /usr/bin/sox -t wa
v - -c 1 -b 16 -t wav - rate 16000 |\n')
    with open(data_dir+'/text','w',encoding='utf8') as f:
        f.write('temp '+id_to_sent[qid]+'\n')
    with open(data_dir+'/utt2spk','w',encoding='utf8') as f:
        f.write('temp temp\n')
    with open(data_dir+'/spk2utt','w',encoding='utf8') as f:
        f.write('temp temp\n')

# Function to extract GOP scores
def extract_gop_scores():
    os.system('./extract_gop_scores.sh')

# Function to validate answers
def validate_answer(gop_file_path, target_word, dict_lexico
n):
    with open(gop_file_path) as gop_file:
        gop_file = gop_file.readlines()
```

```
    phones = []
    phn_gop_scores = []
    for line in gop_file:
        l = line.split()
        l[0] = l[0].split("_")[0]
        if l[0] == "SIL":
            continue
        if l[0][-1].isdigit():
            l[0] = l[0][:-1]
        phones.append(l[0])
        phn_gop_scores.append(float(l[1]))

    for j in dict_lexicon[target_word]:
        phn_indices = [ind for ind, val in enumerate(phones)
if val == j[0]]
        for k in range(len(phn_indices)):
            if j == phones[phn_indices[k]:phn_indices[k]+len
(j)]:
                return np.mean(phn_gop_scores[phn_indices[k]:
phn_indices[k]+len(j)])
```

Team Meetings

Meeting 1:

- Downloaded the code and skimmed through the backend and frontend folders, mainly focusing on the `main.py` file, which contains the logical part of the website.
- Along with that, we looked at some `.sh` files like `path.sh` and `extract_gop_scores.sh`.
- Tried to understand what GOP (Goodness of Pronunciation) scores mean and ran the website to check its functionality.
- Initially encountered some permission and directory issues on submitting our speech. After that, got to know that the model is outputting wrong/correct

answers based on some default values stored in the `gop_scores.txt` file.

Meeting 2 and 3 (with Kumar sir):

- Took help from Kumar sir in our first meeting with him to install the Kaldi module on our local machines so that the model can process our speech and output (all done locally).
- Tried installing the Kaldi model using instructions given in the GitHub repo and ran the model after resolving some issues, but it was throwing an error.
- Sorted out the error after discussing with Kumar sir in our third meeting by changing some paths in `.sh` files. Able to use the functionality of the website locally on our computers.

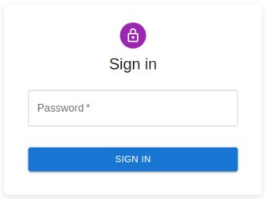
CHANGING THE UI PHASE

Meeting 4:

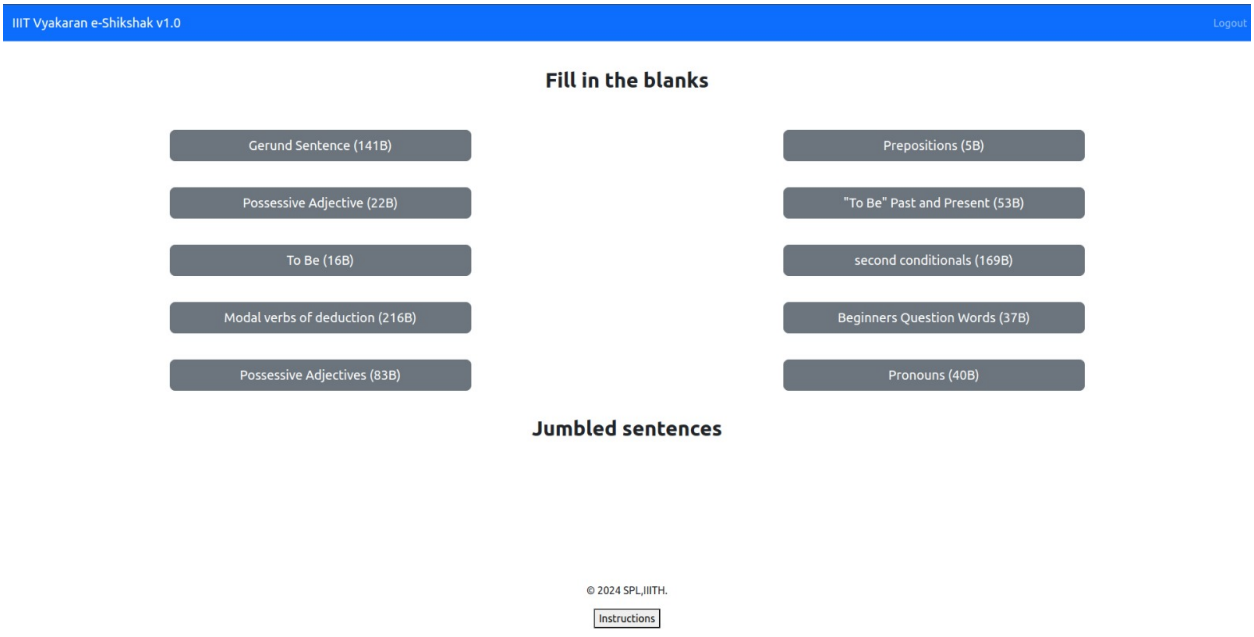
- Discussed the changes that should be made to each page on a higher level and what theme the website should follow.
- Used Bootstrap components to change the UI.

Here is the updated UI for each page:

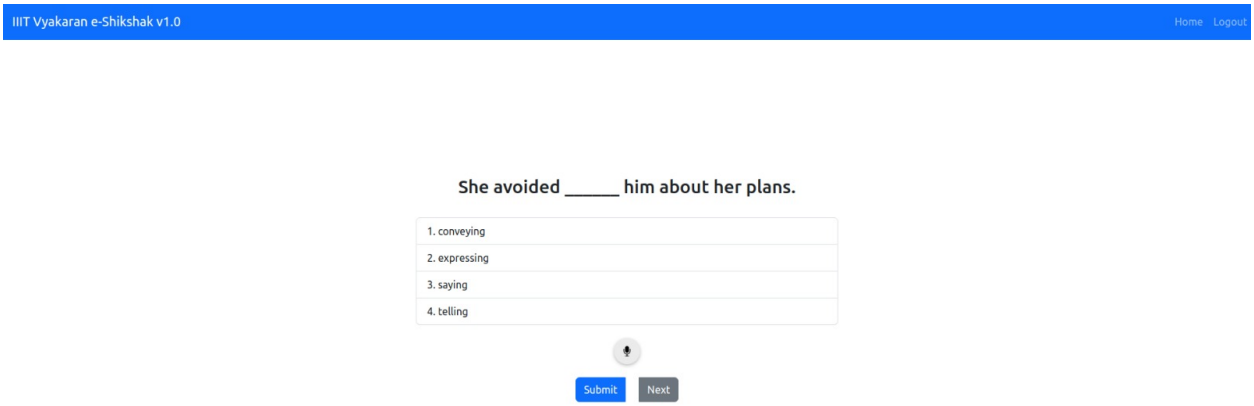
- **LOGIN/SIGNUP Page**



- **DASHBOARD Page**



- **QUESTIONS Page**



NEW FEATURE

Meeting 5:

- In this meeting, we discussed adding jumbled sentences as a new feature to better serve the purpose of the website. The user is supposed to arrange the words to make a sentence and speak out the sentence. This will check the grammatical and eloquence ability of the user.
- Discussed that the feature will be added to the Dashboard page. The page, after clicking on the button, will be similar to the Questions page, except there will be words given to form a sentence. The flow after that will be similar.