2020 级《计算机工程综合实践》期末报告

题目：母语识别

F12019100001 Tomasulo Joseph John

Table of Contents

Introduction

在计算语言学中，母语识别是确定文本作者的母语的任务。该技术的两个主要应用是在语言学习和法医语言学中。对于前者，因为学习者的语言背景会影响他学习目标语言时可能犯的错误，所以啊如下是有益的。对于后者，母语识别被认为是作者身份识别和剽窃检测的一个子任务。

本报告所介绍的母语识别应用可归类为一种语言学习工具。不管他们的母语是什么，大多数领域的研究人员都用英语发表他们的发现。因此，重要的是英语的质量不减损研究的质量。为此，本报告提供了一个工具，帮助非母语作家确定那些方面的英语强烈表示非母语的作者。本项目开发了三个组建：数据采集管道，机器学习模型，和一个浏览器应用程序。

Background

2013 年，第一个为母语识别而创造的数据集发布（Blanchard, et al.）[i]。 它包含 11 个不同母语北京的中高级学生为托福英语考试写的论文，一共有六千个。同年，有了一场关于这个数据集的竞赛（Tetrault, et al.）[ii]。有 29 个团队参加，最成功的团队都使用支持向量机。他们选择不同的特征作为输入，其中最常用的选项包括字符、单词和词性 n-grams。在进行向量化，大多数都使用对数熵或 TF-IDF 加权形式。2017 年，重新举行此竞赛（Malmasi, et al.）[iii]。在此期间，自然语言处理和计算语言学发生了重大变化，就是神经网络被用来有效的创建分布式表示（Mikolov, et al.）[iv]。尽管取得了这些进展，第二届比赛中最成功的团队仍然使用支持向量机和基于统计计算的向量，而没有用新的神经网络生成的向量。相反，大部分团队在不同特征上进行训练然多个模型然后用集成技术来提高性能。

虽然这个项目使用不同数据集，但是这个项目的模型很想的一次比赛中的模型，在支持向量机中使用 TF-IDF 向量。我们没有使用更先进的向量化方法因为在第二比赛中没有提高单模型的性能，没有用集成技术因为这个项目只是为了证明概念。

Data

因为这个工具是为了研究人员使用的，所以模型需要根据研究人员写的论文上进行训练。arXiv[1]是一个收录科学文献预印本的在线数据库[2]。它创建与 1991 年，至 2016 年每月投稿率超过一万篇[3]。在请求者付费模式下[4]，可以使用 s3cmd[5]工具下载整个数据库。

数据集一共有 2807 个.tar 文件（以后称为 tarball），每个包含几百论文，论文是个压缩（tar.gz）的 LaTeX 文件。从这个数据集中，获取了论文的 id，内容，和主要作者附属的大学[6]。因为文件的名称就是其在 arXiv 数据库中的 id，这很容易获取。

大学是用来做标签的，美国或英国大学出来的论文有母语是英语的人写出来的，中国的有非母语的作者。为了实现这一点，我使用了一份按国家划分的大学名单[7]，然后在论文的题目附近搜索了所有中国、英国、美国大学的名字[8]。这个并不是完美的获取标签方式，在英美大学肯定有很多来自中国的研究人员发布论文。原来打算用作者的名字加上一层筛选，但是这版本没有这个功能。因为大学名字太多了，每个名字扫描一次文本

1 https://arxiv.org/
2 https://blog.csdn.net/ctrigger/article/details/90264383
3 https://en.wikipedia.org/wiki/ArXiv
4 下载了 1200（一共有 2807）压缩文件的价格是大约$27 美元。
5 https://s3tools.org/s3cmd
6 process.py 里的 process_tar 函数
7 https://github.com/endSly/world-universities-csv
8 get_uni.py

太低效了，所以用了一个叫做 Aho Corasick[9]的算法来执行一个基于索引的搜索，可以在一次扫描的过程中搜索多个字符串。

因为数据集中的论文没有标准的形式，所以没有简单的方式获取内容。内容的开始和结束通常都是由一小组 LaTeX 标签来表示的（例如/introduction 和/end{abstract}），所以这里也用了同样的字符搜索算法[10]。这些 LaTeX 标签是检查数据集几十个论文手动找到的，这里也有机会做的更全面。拿到内容之后，用了一个叫做 opendetex[11]的工具来删除 LaTeX 的标签。这个工具最近（2020 年 6 月）有人在优化，可能过了一会重新跑代码结果会更干净，但是这第一版本有的标签没有删除、一些标签删除后没有留空格、还有的数学公式只删除了一部分。虽然看来没有太大的影响，如果要分享该数据集，一定要面对这些问题。

除此之外，还下载了一个干净的元数据数据集[12]，叫做 arxiv_archive，它包含了从 1993 年到 2019 年的所有 arXiv 上的论文的元数据。其实，为了完成母语识别的任务，不需要这个数据集，在这个原型的软件中，它唯一的用处是在训练模型的时候，可以筛选科学领域。有可能如果训练数据都来自一个科学领域那么在该领域模型的准确率将高点。在原始的 LaTeX 文件中没有指定科学领域，因此如果没有这个元数据数据集，这是不可能的。此外，此数据集还支持其它改进，例如前面提到的使用作者姓名数据来提高标签准确性的方法。

NLI Dataset

来源	类别	内容
arxiv_archive (zenodo)	标题	Microscopic explanation for...
	筛选	Charged dilatonic black hole...
	arxiv_id	1812.11765
	作者（列表）	Yong Chen, Haitang Li...
	主要领域	hep-th
	科学领域（列表）hep-th,gr-qc	hep-th, astro-ph.CO, gr-qch
	创建日期	2018-12-31
arxiv.org (s3cmd)	内容	In the past decades, black hole thermodynamics has become...
	作者附属的大学	Zhejiang University of Technology
	标签	Non-native

Table 1. 该表显示了数据的结构以及每个属性的来源。arxiv_archive 还有其他类别，这里不显示。

合并两个数据集使用一个简单的方法[13]。首先，加载已处理的 tarball 列表和已添加的数据点，如果没有，创建这两个列表。然后，如果有任何未处理的 tarball，逐个处理，将文件名添加到文件名的列表中、数据添加到数据列表中。这样，当下载更多数据时，可以直接将其添加到现有数据集中，而无需重新处理任何数据。处理每个数据点可以使用 id 将元数据与内容和大学链接起来，因为 id 在两个数据集中都存在[14]。

9 https://pypi.org/project/pyahocorasick/
10 aho_get_content.py
11 https://github.com/pkubowicz/opendetex
12 https://github.com/staeiou/arxiv_archive/tree/v1.0.1 （文档），
https://zenodo.org/record/2533436#.XwGbUd-YUUF（下载）
13 process.py 里的 add_tars 函数
14 merge.py

Model

模型的建立[15]有三个步骤，首先对数据进行预处理，然后进行向量化，最后训练一个线性支持向量机。预处理使用 Python 的 pandas[16]模块，先选择一个或者几个科学领域，然后筛选几句只会在中国大学出来的论文里出现的话。

向量化和训练模型都使用 Python 的 scikit-learn[17]（以后 sklearn）模块。向量化有两步骤。地一个是用一个叫做 CountVectorizer 函数把文本的列表转换为词频矩阵，一个论文一行，列数等于 max_features 参数的值，否则等于整个预料库词汇的大小。这个函数也支持 ngram_range，通过此参数的值控制词的大小。提交的模型设置 max_features 为 5000，ngram_range 设置为（2,3）。第二部是把上一步的矩阵转换为一个 TF-IDF 表示形式。TF 表示一个文档里的词频；IDF 就是一个词的逆文档频率，也就说在多个文档最常见词给予最小的权重。TF-IDF 是他们的相乘，表示如果一个词在一个文档很常见但在所有其它文档很少见，那么他的权重会很高。这两个步骤在 sklearn 函数 TfidfTransformer 中结合在一起。

建立模型第二步骤是一个支持向量机（SVM）。SVM 是一个监督学习方式，它构造的决策边界是对学习样本求解最大边距超平面[18]。这里使用 sklearn 的线性支持向量分类器，还设置 multi_class 参数为 crammer_singer 和 class_weight 为 balanced。Crammer & Singer 的方法将优化所有类的联合目标[v]。Balanced 模式使用标签值来自动调整输入数据中的类频率成反比的权重。

Web App

这个应用程序是使用一个名为 Flask[19]的 Python 模块开发的。Flask 是一个 web 微框架，让用户能够链接 Python 和 HTML。这个项目的网页由一个 Flask 脚本，一个 python 脚本，几个 HTML 模板组成。Flask 代码[20]十分简单，只负责路由和在 Python 和 HTML 之间传输数据。提交的版本只支持上传.txt 文件和粘贴内容，以后如果支持.tex 和.docx 那就对用户更方便。Python 的脚本[21]负责加载经过训练的模型，使用它对输入进行预测，并格式化结果。在输入中显示的模型的所有特征都将亮显并标记其权重。第三部分使用 HTML，CSS 和 Bootstrap[22]来显示信息并提供基本的 UX。

Discussion

通过分析错误的预测，可以理解这个工具的缺点。False Negatives 是模型判断有非母语的作者但是数据集中标签是母语的作者。记得标签来自作者附属的大学。如果很多以中文为母语的作者在美国大学写论文，我们会看很多中国名字在 False Negatives 里面。

False Negatives	作者的名字	False Positives	主要作者的名字
1404.3811	Zhiqiang Xu	1406.4802	Charles Soussen, Jerome Idier
1003.5305	Sreekanth Malladi	1511.04123	Luis Barba, Otfried

15 run.py
16 https://pandas.pydata.org/
17 https://scikit-learn.org/stable/
18 https://baike.baidu.com/
19 https://flask.palletsprojects.com/en/1.1.x/
20 site/app.py
21 site/single.py
22 https://getbootstrap.com/

			Cheong
1411.0294	Andrea Grigorescu	1602.04706	Kyeong Soo Kim
1307.1756	Cheng Bo	0909.1428	Daowen Qiu, Paulo Mateus
1401.3582	Xiaomin Bao	1203.4455	Xiaohui Bei
0707.1905	Yuri L. Zuev	1509.05480	Pingzhong Tang
1602.02991	Saeed Akhoondian Amiri	1308.2027	Tao Huang
1602.1948	Cristina Benea	1306.6265	Hervce Chabanne, Gerard Cohen
1602.05292	Zhenhao Ge	1111.1328	Xueying Duan
0911.1564	Tony Cai, Lie Wang	1010.3348	Szilard Andras
1512.08493	Lina Yao, Quan Z. Sheng	1506.00743	Xiaowang Zhang
		1309.3052	Ping Cao

Table 2. 这表表示所有预测错的样本和其作者的名字。

在 Table 2 左边可以看 11 个作者中有 6 个有中国的名字。这些可能是标签的问题，而不是模型的问题，如果想避免这种情况，选择标签时应该考虑作者的名字。右边有一些名字看来不是中国的，因此，从中国大学发表的论文列表中筛选出作者姓名非中文的论文也应该提高数据集标签的准确性。

Conclusion

这个项目非常有教育意义。 在第一部分中，我了解了数据集的创建还有文本处理的一些基本概念。 在第二部分中，我发现尝试使用 Scikit-Learn 非常有用，以了解一些基本的 NLP 技术。 在第三部分中，我学习了一种将 Python 代码链接到网站的简单方法。

如前所述，该项目具有比较大的发展潜力。 可以清理数据集，并且标签可以更准确。 可以扩展为多分类的模型，针对更多语言背景的用户。可以提高该模型的准确率。还有，现在超过一两百兆的模型导致 Flask 的加载时间很慢，如果设置服务器，可能不用太担心模型的大小。 该网站可以为用户提供针对不同应用不同模型的选择，也可以支持上传更多的文件形式。

这个项目是我爸推荐的，他是医生没有计算机经验，不知道他怎么想出来，可能是应为我之前跟他说过我做的另外一个 NLP 项目。虽然我使用很多模块，所有提交的代码是我为了这门课自己写的。我的 github[23]上有所有代码和一些相关的文件，还有在本地计算机上运行的说明。模型、数据集的一小部分和测试文件都可以在我的腾讯微云[24]上找到。

23 https://github.com/j-toma/nli
24 链接：https://share.weiyun.com/kbrvsZai 密码：pufvc4

Appendix A: process.py

```
import tarfile
import os
import sys
import pickle
from get_unis import unis_in_countries
from aho_get_content import get_content
from get_uni import get_uni

def process_tar(tar_name):
    directory = '/home/jtoma/nli/arxiv/dump'
    os.chdir(directory)
    ### countries specified in get_unis file
    unis = unis_in_countries()
    ### open file, get files in tar
    tar = tarfile.open(tar_name)
    members = tar.getmembers()
    ### initialize return value
    ret = []
    for member in members:
        if not member.isfile():
            continue
        else:
            article_id = member.name.rstrip('.gz')[5:]
        ### extract individual paper
        f = tar.extractfile(member)
        try:
            ### get uni associated with primary author of individual paper
            unis_without_country = [ el[1] for el in unis ]
            hit = get_uni(f,unis_without_country)
            if hit:
                ### get content of paper
                content = get_content(f)
                if content and type(content) == bytes:
                    content = content.decode('utf-8','ignore')
                    content = content.replace('\n','')
                    ### content
                    obj = {'article_id': article_id, 'uni': hit, 'content': content}
                    ret.append(obj)
            else:
                pass
        except OSError:
            continue
        f.close()
    tar.close()
    return ret
```

Appendix B: get_uni.py

```
import ahocorasick as ahc
from get_unis import unis_in_countries
import io, gzip, time


def make_automaton(unis):
    A = ahc.Automaton()
    for index, uni in enumerate(unis):
        A.add_word(uni, (index,uni))
    A.make_automaton()
    return A

def aho_corasick(text, unis):
    '''
        return example
        [(1874, (1, 'University of Massachusetts'))]
        loc, (index?, uni)
    '''
    A = make_automaton(unis)
    found_keywords = []
    for item in A.iter(text):
        #print(item)
        #print(line)
        found_keywords.append(item)
        break
    return found_keywords

def get_uni(f, unis):
    f.seek(0)
    title_loc = f.read().find(b'\\title{')
    f.seek(title_loc - 2000)
    text = io.TextIOWrapper(io.BufferedReader(gzip.open(f)), \
            encoding='utf8', errors='ignore').read(6000)
    hit = aho_corasick(text, unis)
    if hit:
        return hit[0][1][1]
    else:
        return False
```

Appendix C: get_aho_content.py

```
import ahocorasick as ahc
import io, gzip
import subprocess
import time
from bs4 import BeautifulSoup as bs


def make_keywords():
    start_strs = [
            #'\\end{abstract}',
            'Introduction',
        ]
    end_strs = [
            '\\begin{thebibliography}',
            '\\bibliography',
            '\\sub{\\bf{References}}',
        ]
    return start_strs, end_strs


def analyze_hits(hits):
    s_strs, e_strs = make_keywords()
    start = (float('inf'),'')
    end = (0,'')
    has_start = False
    has_end = False
    for hit in hits:
        match = hit[1][1]
        loc = hit[0]
        if match in s_strs:
            if loc < start[0]:
                start = (loc, match)
        elif match in e_strs:
            if loc > end[0]:
                end = (loc, match)
    return start, end


def make_automaton(kws):
    A = ahc.Automaton()
    for index, kw in enumerate(kws):
        A.add_word(kw, (index,kw))
    A.make_automaton()
    return A

def aho_corasick(text, kws):
    '''
        return example
        [(1874, (1, 'University of Massachusetts'))]
        loc, (index?, uni)
    '''
    A = make_automaton(kws)
    found_keywords = []
    for item in A.iter(text):
        found_keywords.append(item)
    return found_keywords


def get_content(f):
    f.seek(0)
    s_strs, e_strs = make_keywords()
    kws = s_strs + e_strs
    text = io.TextIOWrapper(
            io.BufferedReader(gzip.open(f)),
            encoding='utf8', errors='ignore').read()
    hits = aho_corasick(text, kws)
    start, end = analyze_hits(hits)
    size = end[0] - start[0]
    if size > 5000:
        content = text[start[0]:end[0]].encode()
        detex_content = subprocess.run(
                ['detex'],
                input=content,
                stdout=subprocess.PIPE
                )
        content = detex_content.stdout
        return content
    else:
        return False
```

Appendix D: merge.py

```
import pandas as pd
import glob, os, pickle
from process import add_tars
from get_unis import unis_in_countries, native

def merge(n):
    STORE_PATH = '/home/jtoma/nli/data'
    f = 'metadata.pickle'
    metadata_path = os.path.join(STORE_PATH, f)
    print('running merge')
    try:
        df_all = pd.read_pickle(metadata_path)
        print("Loading metadata from pickle")
    except (OSError, IOError) as e:
        print("Creating metadata pickle")
        dumpdate = "20190101"
        datadir = "arxiv_archive-1.0.1/processed_data/" + dumpdate + "/per_year/"
        files = glob.glob(datadir + "*.tsv.zip")
        len(files)
        files.sort()
        dtypes = {
            "abstract": object, "acm_class": object,
            "arxiv_id": object, "author_text": object,
            "categories": object, "comments": object,
            "created": object, "doi": object,
            "num_authors": int, "num_categories": int,
            "primary_cat": object, "title": object,
            "updated": object,  "created_ym": object
        }
        df_all = pd.DataFrame()
        for f in files:
            print(f)
            yearly_df = pd.read_csv(
                    f,
                    sep="\t",
                    index_col=0,
                    compression='zip',
                    dtype=dtypes,
                    parse_dates=["created","updated"])
            df_all = df_all.append(yearly_df)
        df_all.to_pickle(metadata_path)

    print('length before drop', len(df_all))
```

Appendix D (Cont.): merge.py

```
    # every time we run merge, we increase the size of the dataset
    additional_data = add_tars(n)

    keys = [additional_data[i]['article_id'] for i in
            range(len(additional_data))]

    is_hit = df_all['arxiv_id'].isin(keys)
    df_hit = df_all[is_hit]

    ### add uni
    uni_d = { additional_data[i]['article_id']: additional_data[i]['uni'] for
            i in range(len(additional_data)) }
    df_hit.loc[:,'uni'] = df_hit['arxiv_id'].map(uni_d)

    ### add content
    content_d = { additional_data[i]['article_id']: additional_data[i]['content'] for
            i in range(len(additional_data)) }
    df_hit.loc[:,'content'] = df_hit['arxiv_id'].map(content_d)

    ### add country of uni
    country_d = { el[1]:el[0] for el in unis_in_countries() }
    df_hit.loc[:,'country'] = df_hit['uni'].map(country_d)

    ### give a column for native or non-native
    native_d = { el[1]:el[0] for el in native() }
    df_hit.loc[:,'native'] = df_hit['uni'].map(native_d)

    print('length after drop', len(df_hit))
    print('df_hit.head():', df_hit.head())
    STORE_PATH = '/home/jtoma/nli/data/'
    file_name = 'ds1.pickle'
    pickle_file = os.path.join(STORE_PATH, file_name)
    df_hit.to_pickle(pickle_file)
    return df_hit

merge(10)
```

Appendix E: run.py

```
import pandas as pd
from sklearn import metrics
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import Normalizer, LabelEncoder
from sklearn.calibration import CalibratedClassifierCV
from sklearn.svm import LinearSVC, SVC
from sklearn.utils import resample
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from joblib import dump
import re
import time


def pretty_print_cm(cm, class_labels):
    row_format = "{:>5}" * (len(class_labels) + 1)
    print(row_format.format("", *class_labels))
    for l1, row in zip(class_labels, cm):
        print(row_format.format(l1, *row))

def remove_bigram_stops(doc):
    doc = re.sub('[^a-z\s]', '', doc.lower())               # get rid of noise
    stop_phrases = [
            "of(\s?\\.?\s?)china",
            "national(\s?\\.?\s?)natural",
            "science(\s?\\.?\s?)foundation",
            "eqnarray",
            "et al.",
            ]
    for phrase in stop_phrases:
        doc = re.sub(phrase, "", doc, flags=re.IGNORECASE)
    return doc

def under_sample(df):
    print('Original counts:')
    print('non native:', df[df.native == False].shape[0], '| native:', df[df.native == True].shape[0])
    df_maj = df[df.native==True]
    df_min = df[df.native==False]
    df_maj_under = resample(df_maj, replace=False, n_samples=df_min.shape[0],
            random_state=123)
    df_under = pd.concat([df_min, df_maj_under])
    print('count of native after resample:', df_under.native.value_counts()[0])
    print('----')
    return df_under
```

Appendix E (Cont. 2): run.py

```
def filter_categories(df):
    # only cs and not physics
    df1 = df[df['categories'].str.contains('cs.') & ~(df['categories'].str.contains('ics.'))]
    return df1

def cut_content(df):
    mean_len_content_before = df.content.apply(lambda x:len(x)).mean()
    df.content = df.content.str[:20000]
    mean_len_content_after = df.content.apply(lambda x:len(x)).mean()
    print('Average character count in content:')
    print('before cut:', mean_len_content_before, '| after cut:',
          mean_len_content_after)
    print('----')
    return df

def run():
    #CLASS_LABELS = ['CN', 'GB', 'IN', 'US']
    CLASS_LABELS = ['NON', 'NAT']

    # get data
    df = pd.read_pickle('data/ds1.pickle')

    # filter categories
    df = filter_categories(df)

    # undersample majority class to size of minority
    df_under = under_sample(df)

    # trim content length for faster training
    #df_under = cut_content(df_under)

    # remove nonlinguistic indicators
    df_under.content = df_under.content.apply(remove_bigram_stops)

    # set data
    X = df_under.content
    y = df_under.native
    print('length of X:', len(X))

    clf = Pipeline(
          steps=[
             ('tfidf', TfidfVectorizer(ngram_range=(2,3), analyzer='word',
                binary=True, max_features=5000)),
             ('svc', LinearSVC(multi_class='crammer_singer',
                class_weight='balanced')),
          ]
12)
```

Appendix E (Cont. 3): run.py

```
    X_train, X_test, y_train, y_test = train_test_split(X, y,
            test_size=0.2, random_state=7)

    s = time.time()
    print("Training LinearSVC...")
    clf.fit(X_train, y_train)
    print("Training Time" + str(time.time() -s ))
    s= time.time()
    print("Linear SVC Score- "+ str(clf.score(X_test, y_test)))
    predicted = clf.predict(X_test)

    print('Results')
    print("\nConfusion Matrix:\n")
    cm = metrics.confusion_matrix(y_test, predicted).tolist()
    pretty_print_cm(cm, CLASS_LABELS)
    print("\nClassification Results:\n")
    print(metrics.classification_report(y_test, predicted, target_names=CLASS_LABELS, digits=4))

    dump(clf, 'data/pipe1.joblib')
    print('pipeline dumped to pipe1.joblib'

    #feature_names = clf.named_steps.tfidf.get_feature_names()
    # if using calibratedclassifierCV
    #coef_avg = 0
    #for i in clf.calibrated_classifiers_:
    #    coef_avg = coef_avg + i.base_estimator.coef_
    #coef_avg  = coef_avg/len(clf.calibrated_classifiers_)
    #coefs_with_fns = sorted(zip(coef_avg, feature_names))

    #coefs_with_fns = sorted(zip(clf.named_steps.svc.coef_[0], feature_names))
    #df=pd.DataFrame(coefs_with_fns)
    #df.columns='coefficient','n-gram'
    #df.sort_values(by='coefficient')
    #print('top 50 ngrams indicating non native:')
    #print(df[:50])
    #print('top 50 ngrams indicating native:')
    #print(df[-50:])
    #print('----')
    #pickle.dump([clf, vectorizer], open('data/clf1.pickle', 'wb'))
    #print('classifier, vectorizer, and features dumped to pickle clf1.pickle')
    print('run completed')

run()
```

Appendix E: site/app.py

```
from flask import Flask, render_template, request, request, url_for, flash, send_from_directory
import sys
sys.path.append("/home/jtoma/nli")
from single import display_text

UPLOAD_FOLDER = '/home/jtoma/nli/static'
ALLOWED_EXTENSIONS = {'txt'}

app = Flask(__name__)
app.config["SEND_FILE_MAX_AGE_DEFAULT"] = 1
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

@app.route('/', methods=['GET','POST'])
def upload():
    if request.method == 'POST':
        #if 'file' not in request.files:
        #    flash('No file part')
        #    return redirect(request.url)
        if 'file' in request.files:
            f = request.files['file']
            if f.filename == '':
                flash('no selected file')
                return redirect(request.url)
            if f and allowed_file(f.filename):
                f.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
                return redirect(url_for('pred_up'))
        if 'doc' in request.form:
            return redirect(url_for('pred_paste'))

    return render_template('index.html')

@app.route('/pred_up', methods=['GET','POST'])
def pred_up():
    up_content = request.files['doc']
    pred, prob, mkdn = display_text(up_content)
    data = [pred[0],prob, mkdn]
    return render_template('prediction.html', data=data)

@app.route('/pred_paste',
methods=['GET','POST'])
def pred_paste():
    paste_content =  request.form['paste']
    pred, prob, mkdn = display_text(paste_content)
    data = [pred[0],prob, mkdn]
    return render_template('prediction.html',
data=data)

@app.route('/<string:page_name>/')
def render_static(page_name):
    return render_template('%s.html' % page_name)

if __name__ == '__main__':
    app.run(host='0.0.0.0', debug=True)
```

Appendix F: site/single.py

```
from joblib import load
from markupsafe import Markup
import nltk
nltk.download('punkt')
from nltk.tokenize import sent_tokenize, word_tokenize
from pathlib import Path

#clf, vectorizer = pickle.load(open('data/clf1.pickle','rb'))
model_file = Path('../data/pipe1.joblib')
pipeline = load(model_file)
clf = pipeline.named_steps.svc
vectorizer = pipeline.named_steps.tfidf

def feature_in_sentence(l, s):
    '''
        input
            list of sentences
            query
        output
            index of sentences that contain query
    '''
    return [index for index, value in enumerate(l) if s in value]

def display_text(content):
    if type(content) == str:
        pass
    else:
        content = content.read().decode('utf-8')
    content_vectorized = vectorizer.transform([content])
    prediction = clf.predict(content_vectorized)
    #print('prediction:', prediction)
    proba = clf.decision_function(content_vectorized)
    proba = round(proba[0],4)
    #print('proba:', proba)
    content = display_list(content, vectorizer)
    return prediction, proba, content
```

Appendix F (Cont. 2): site/single.py

```
def display_list(c,v):
    feature_names = vectorizer.get_feature_names()
    coefs_with_fns = sorted(zip(clf.coef_[0], feature_names))

    strong_non = coefs_with_fns[:100]
    count_strong_non, count_weak_non, count_weak_nat, count_strong_nat = 0, 0, 0, 0
    strong_non_list, weak_non_list, weak_nat_list,  strong_nat_list = [], [], [], []
    weak_non = coefs_with_fns[100:1000]
    weak_nat = coefs_with_fns[-1000:-100]
    strong_nat = coefs_with_fns[-100:]

    sent_tokens = sent_tokenize(c)
    #print('sent tokens:', sent_tokens)
    for phrase in strong_non:
        spaced_phrase = ' ' + phrase[1] + ' '
        indices_of_occurences = feature_in_sentence(sent_tokens, spaced_phrase)
        #print('strong non indices of occurences:', indices_of_occurences)
        for i in indices_of_occurences:
            sent = sent_tokens[i]
            count_strong_non += 1
            replacement = '<span style="color:red">' + spaced_phrase + '</span>'
            sent = sent.replace(spaced_phrase, replacement)
            sent = Markup(sent)
            strong_non_list.append([i, sent, phrase[1], round(phrase[0],4)])
    for phrase in weak_non:
        spaced_phrase = ' ' + phrase[1] + ' '
        indices_of_occurences = feature_in_sentence(sent_tokens, spaced_phrase)
        for i in indices_of_occurences:
            sent = sent_tokens[i]
            count_weak_non += 1
            replacement = '<span style="color:orange">' + spaced_phrase + '</span>'
            sent = sent.replace(spaced_phrase, replacement)
            sent = Markup(sent)
            weak_non_list.append([i, sent, phrase[1], round(phrase[0],4)])
    for phrase in weak_nat:
        spaced_phrase = ' ' + phrase[1] + ' '
        indices_of_occurences = feature_in_sentence(sent_tokens, spaced_phrase)
        for i in indices_of_occurences:
            sent = sent_tokens[i]
            count_weak_nat += 1
            replacement = '<span style="color:yellowgreen">' + spaced_phrase + '</span>'
            sent = sent.replace(spaced_phrase, replacement)
            sent = Markup(sent)
            weak_nat_list.append([i, sent, phrase[1], round(phrase[0],4)])
    for phrase in strong_nat:
        spaced_phrase = ' ' + phrase[1] + ' '
        indices_of_occurences = feature_in_sentence(sent_tokens, spaced_phrase)
        for i in indices_of_occurences:
            sent = sent_tokens[i]
            count_strong_nat += 1
            replacement = '<span style="color:green">' + spaced_phrase + '</span>'
            sent = sent.replace(spaced_phrase, replacement)
            sent = Markup(sent)
            strong_nat_list.append([i, sent, phrase[1], round(phrase[0],4)])
    return [strong_nat_list, weak_nat_list, weak_non_list, strong_non_list]
```

Appendix G: site/templates/index.html

```
<html>
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title> 母语识别 </title>
  <link href="{{ url_for('static', filename='../static/bootstrap.min.css') }}" rel="stylesheet">
  <link href="../static/style.css" rel="stylesheet">
  <link rel="shortcut icon" href="{{ url_for('static', filename='favicon.ico') }}">
  <link href="{{ url_for('static', filename='../static/bootstrap.min.css') }}" rel="stylesheet">
  <link href="../static/bootstrap.min.css" rel="stylesheet">
  <link href="../static/style.css" rel="stylesheet">
  <script src="../static/bootstrap.min.js"></script>
</head>
<body>
  <div class="container-fluid">
    <div class="jumbotron">
      <h1 class="display-4"> 母语识别工具 </h1>
      <p class="lead"> 这个工具试图分别本地人和中国人写的英文文本。 </p>
    </div>
    <a href="try_it_out">Try it out!  直接试试！ </a>
  </div><br>

  <div class="container">

    <ul class="nav nav-tabs" id="myTab" role="tablist">
      <li class="nav-item" role="presentation">
        <a class="nav-link active" id="home-tab" data-toggle="tab" href="#home"
          role="tab" aria-controls="home" aria-selected="true"><h3>Home  主页 </h3></a>
      </li>
      <li class="nav-item" role="presentation">
        <a class="nav-link" id="upload-tab" data-toggle="tab" href="#upload"
          role="tab" aria-controls="profile" aria-selected="false">
          <h3>Upload  上载 </h3>
        </a>
      </li>
      <li class="nav-item" role="presentation">
        <a class="nav-link" id="paste-tab" data-toggle="tab" href="#paste"
          role="tab" aria-controls="contact" aria-selected="false">
          <h3>Paste  输入 </h3>
        </a>
      </li>
    </ul>
```

Appendix G (Cont. 2): site/template/index.html

```
  <div class="tab-content" id="myTabContent">
    <div class="tab-pane fade show active" id="home" role="tabpanel" aria-labelledby="home-tab">

      <br>
      <h5> 介绍 </h5>
      <p> 该网站提供了一种机器学习模型，该模型试图将母语为英语的人的文字与母语不是英语的中国作者的文字区分开。 </p>
      <h5> 数据 </h5>
      <p> 该模型使用来自 arxiv.org 数据进行训练。下载之后，需要进行处理，主要有两步骤。第一步是确定主要作者所属的大学（如果有），把这个大学当作标签。第二步是获取论文的内容。 </p>
      <h5> 模型 </h5>
      <p> 定义此模型时，我决定用最小最简单的模型能够完成任务，我随便要求 90% 以上正确率。这个模型主要有两部分。第一是个 TF-IDF
      Vectorizer 。第二步是将向量化的内容输入到个线性支持向量分类器。 </p>
      <h5> 网页 </h5>
      <p> 这个网页提供的 sklearn 模型到 html 内容的链接用 python 模块 flask 而完成的。设计方面使用 bootstrap 。 </p>
    </div>
    <div class="tab-pane fade" id="upload" role="tabpanel" aria-labelledby="upload-tab">
      <br>
      <form method="POST" action="{{url_for('pred_up')}}" enctype="multipart/form-data">
        <div class="input-group">
          <div class="custom-file">
            <input type="file" name="doc" class="custom-file-input" id="myInput" aria-describedby="myInput">
            <label class="custom-file-label" for="myInput">Choose file</label>
          </div>
          <button type="submit" class='btn btn-primary'>Submit</button>
        </div>
      </form>
    </div>
    <div class="tab-pane fade" id="paste" role="tabpanel" aria-labelledby="paste-tab">
      <br>
      <form method="POST" action="{{url_for('pred_paste')}}" enctype="multipart/form-data">
        <div class="form-group">
          <label for="exampleFormControlTextarea1"></label>
          <textarea class="form-control" name="paste" id="exampleFormControlTextarea1"
rows="10"></textarea></br>
        </div>
        <button type="submit" class='btn btn-primary'>Submit</button>
      </form>
    </div>
  </div>
 </div>
  <footer class="footer">
    <div class="container-fluid">
      <a href="https://www.ynu.edu.cn/"> 云南大学  Yunnan University</a>
    </div>
  </footer>
 <script src="{{ url_for('static', filename='../static/jquery.js') }}"></script>
 <script src="{{ url_for('static', filename='../static/bootstrap.min.js') }}"></script>
</body>
</html>
```

Appendix H: site/templates/prediction.html

```
<html>
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Prediction</title>

  <link rel="shortcut icon" href="{{ url_for('static', filename='favicon.ico') }}">
  <link href="../static/bootstrap.min.css" rel="stylesheet">
  <script src="../static/jquery.js"></script>
  <script src="../static/bootstrap.min.js"></script>
  <link href="../static/style.css" rel="stylesheet">

</head>
<body>
  <div class="container">
    {% if data[0] %}
    <div class="alert alert-success">
      <h1 class="display-4"> 作者的母语：英语 </h1>
      <p class="lead"> 成绩 : {{ data[1] }}</h4>
      <hr class="my-4">
      <p>
        <a class="btn btn-primary" data-toggle="collapse" href="#collapseExample" role="button" aria-
expanded="false" aria-controls="collapseExample"> 解释 </a>
      </p>
      <div class="collapse" id="collapseExample">
        <div class="card card-body">
              一篇文章的等成绩高，以英语为母语的人写出来的可能性就越大。 负值表示该模型预测文本是由
第一语言为中文的人编写的。
        </div>
      </div>
    </div>
    {% else %}
    <div class="alert alert-danger">
      <h1 class="display-4"> 作者的母语：中文 </h1>
      <p class="lead"> 成绩 : {{ data[1] }}</h4>
      <hr class="my-4">
      <p> 一篇文章的等级越高，以英语为母语的人写出来的可能性就越大。
          负值表示该模型预测文本是由非英语母语者撰写的。 </p>
    </div>
    {% endif %}
  </div>
```

Appendix H (Cont. 2): site/template/prediction.html

```
<div class="container">
  <div class="accordion" id="accordionExample">
    <div class="card">
      <div class="card-header" id="headingOne">
        <h2 class="mb-0">
          <button class="btn btn-link btn-block text-left"
                  type="button"
                  data-toggle="collapse"
                  data-target="#collapseOne"
                  aria-expanded="true"
                  aria-controls="collapseOne">
            Strong Non Nat ({{ data[2][3]|length }})
          </button>
        </h2>
      </div>
      <div id="collapseOne"
          class="collapse"
          ariea-labelledby="headingOne"
          data-parent="#accordionExample">
        <div class="card-body">
          <h2>Strong Non Nat</h2>
          <p>some text</p>
          <table class="table table-striped">
            <thead>
              <tr>
                <th>Sentence Index</th>
                <th>Sentence</td>
                <th>Feature</th>
                <th>Feature Strength</th>
              </tr>
            </thead>
            <tbody>
              {% for item in data[2][3] %}
              <tr>
                <td>{{ item[0] }} </td>
                <td>{{ item[1] }} </td>
                <td>{{ item[2] }} </td>
                <td>{{ item[3] }} </td>
              </tr>
              {% endfor %}
            </tbody>
          </table>
        </div>
      </div>
    </div>
```

Appendix H (Cont. 3): site/template/prediction.html

```
<div class="card">
 <div class="card-header" id="headingTwo">
  <h2 class="mb-0">
   <button class="btn btn-link btn-block text-left"
           type="button"
           data-toggle="collapse"
           data-target="#collapseTwo"
           aria-expanded="true"
           aria-controls="collapseTwo">
    Weak Non Nat ({{ data[2][2]|length }})
   </button>
  </h2>
 </div>
 <div id="collapseTwo" class="collapse" data-parent="#accordionExample">
  <div class="card-body">
   <h2>Weak Non Nat</h2>
   <p>some text</p>
   <table class="table table-striped">
    <thead>
     <tr>
      <th>Sentence Index</th>
      <th>Sentence</td>
      <th>Feature</th>
      <th>Feature Strength</th>
     </tr>
    </thead>
    <tbody>
     {% for item in data[2][2] %}
     <tr>
      <td>{{ item[0] }} </td>
      <td>{{ item[1] }} </td>
      <td>{{ item[2] }} </td>
      <td>{{ item[3] }} </td>
     </tr>
     {% endfor %}
    </tbody>
   </table>
  </div>
 </div>
</div>
```

Appendix H (Cont. 4): site/template/prediction.html

```
<div class="card">
  <div class="card-header" id="headingTwo">
    <h2 class="mb-0">
      <button class="btn btn-link btn-block text-left"
              type="button"
              data-toggle="collapse"
              data-target="#collapseThree"
              aria-expanded="true"
              aria-controls="collapseThree">
        Weak Nat ({{ data[2][1]|length }})
      </button>
    </h2>
  </div>
  <div id="collapseThree" class="collapse" data-parent="#accordionExample">
    <div class="card-body">
      <h2>Weak Nat</h2>
      <p>some text</p>
      <table class="table table-striped">
        <thead>
          <tr>
            <th>Sentence Index</th>
            <th>Sentence</td>
            <th>Feature</th>
            <th>Feature Strength</th>
          </tr>
        </thead>
        <tbody>
          {% for item in data[2][1] %}
          <tr>
            <td>{{ item[0] }} </td>
            <td>{{ item[1] }} </td>
            <td>{{ item[2] }} </td>
            <td>{{ item[3] }} </td>
          </tr>
          {% endfor %}
        </tbody>
      </table>
    </div>
  </div>
</div>
```

Appendix H (Cont. 4): site/template/prediction.html

```
        <div class="card">
          <div class="card-header" id="headingFour">
            <h2 class="mb-0">
              <button class="btn btn-link btn-block text-left"
                      type="button"
                      data-toggle="collapse"
                      data-target="#collapseFour"
                      aria-expanded="true"
                      aria-controls="collapseFour">
                Strong Nat ({{ data[2][0]|length }})
              </button>
            </h2>
          </div>
          <div id="collapseFour" class="collapse" data-parent="#accordionExample">
            <div class="card-body">
              <h2>Strong Nat</h2>
              <p>some text</p>
              <table class="table table-striped">
                <thead>
                  <tr>
                    <th>Sentence Index</th>
                    <th>Sentence</td>
                    <th>Feature</th>
                    <th>Feature Strength</th>
                  </tr>
                </thead>
                <tbody>
                  {% for item in data[2][0] %}
                  <tr>
                    <td>{{ item[0] }} </td>
                    <td>{{ item[1] }} </td>
                    <td>{{ item[2] }} </td>
                    <td>{{ item[3] }} </td>
                  </tr>
                  {% endfor %}
                </tbody>
              </table>
            </div>
          </div>
        </div>
      </div>
    </div>
    <footer class="footer">
    <div class="container-fluid">
      <a href="https://www.ynu.edu.cn/"> 云南大学  Yunnan University</a>
    </div>
    </footer>
  </body>
```

i Blanchard, Daniel, et al. ETS Corpus of Non-Native Written English LDC2014T06. Web Download. Philadelphia: Linguistic Data Consortium, 2014.

ii Joel Tetreault, et al. A Report on the First Native Language Identification Shared Task. W13-1706 48-57. Association for Computation Linguistics, 2013.

iii Malmasi, Servin, et al. A Report on the 2017 Native Language Identification Shared Task. W17-5007 62-75. Association for Computation Linguistics, 2017.

iv Mikolov, Thomas, et al. Efficient Estimation of Word Representations in Vector Space. 1301.3781. Arxiv, 2013

v Crammer, K., Singer, Y. On the Learnability and Design of Output Codes for Multiclass Problems. *Machine Learning* 47, 201-233, 2002.