自动化工具复现说明

191250189 袁军平

1.论文理解 & 核心算法

问题背景

这篇文章关注的是减轻语言中的性别刻板印象的反事实数据扩增。比如英语中提到工程师往往是用he。在NLP领域中，对于此类语料的学习，可能会传播或者放大这种现象，所以这篇作者提出了一种办法，通过对原有的数据扩增来减轻这种现象。

目前主流的方法主要用于英语中的处理。但主流方法在处理形态丰富的语言如西班牙语时，会产生不合语法的现象。如考虑句子*el ingeniero experto*（*the skilled engineer*）,只替换ingeniero会导致语法错误不仅需要将ingeniero替换为*ingeniera*，还要替换el以及experto。在替换一个词的时候，还需要考虑替换对周围词造成的影响

这篇文章提出了一个解决办法，主要步骤为以下4步。

1.分析句子

2.替换一个特定的性别词

3.推导出句子新的形态句法标签

4.将句子映射为新形态（完成数据扩增）

下面简要介绍这4步

分析句子

对于某个特定的句子，需要分析三部分

T 依赖树(i,j,l) i,j是词的位置。l是i与j之间的修饰关系 如advmod，amod等

morpho-syntactic tags : 记录词的单复数以及性别词如词 *ingeniero*的tag是[MSC; SG]

POS tags：记录词的词性 如词 *ingenier*od的tag是[noun] 因为这个词是名词

现在一个句子可以用[T,M,P]来代替，记为SC(**去词法化，只关心依赖结构和语法形态标签**)

作者定义了一个马尔可夫随机场(MRF)来表示给定T与P，序列M($m_1 m_2 ... m_n$)的联合概率分布

2.替换（干涉）性别词

对于某个句子，可以通过反转其中词语的性别属性来得到新的SC（只改变了M）,如[MSC;SG]->[FEM,SG]。

对于某个句子来说，可能有很多性别词，通过选择其中一定的性别词（组合数）来转换完成数据扩增。

3.推导句子新的形态句法标签

显然，要判断一种转换mi是不是合法的，可以考虑在给定T与P的情况下，mi的可能性 作者给出了以下公式：

$$\Pr(\mathbf{m} \mid T, \mathbf{p}) \propto \prod_{(i,j,\ell)\in T} \phi_i(m_i) \cdot \psi(m_i, m_j \mid p_i, p_j, \ell),$$

其中二元因子 ψ(·,· | ·,·,·)衡量了mi,mj,pi,pj的匹配程度 一元因子φi(·) ≥ 0表示了一个形态句法标签与句子内容的不匹配程度。作者通过使用这些一元因子来实现一个对句子的干涉。

对于衡量句子整体，作者给出了以下公式

$$Z(T, \mathbf{p}) = \sum_{\mathbf{m}'\in\mathcal{M}} \prod_{(i,j,\ell)\in T} \phi_i(m'_i) \cdot \psi(m'_i, m'_j \mid p_i, p_j, \ell).$$

Z(T,P)给出了在指定T与P的基础上，标签序列M的可能性

作者对于二元因子 ψ给出了线性参数化的表达方式

$$\psi(m_i, m_j \mid p_i, p_j, \ell) = \exp\left(\overline{m}_i^\top W(p_i, p_j, \ell)\overline{m}_j\right)$$

最终的联合概率通过置信传播来计算。

下面是我对置信传播的理解：

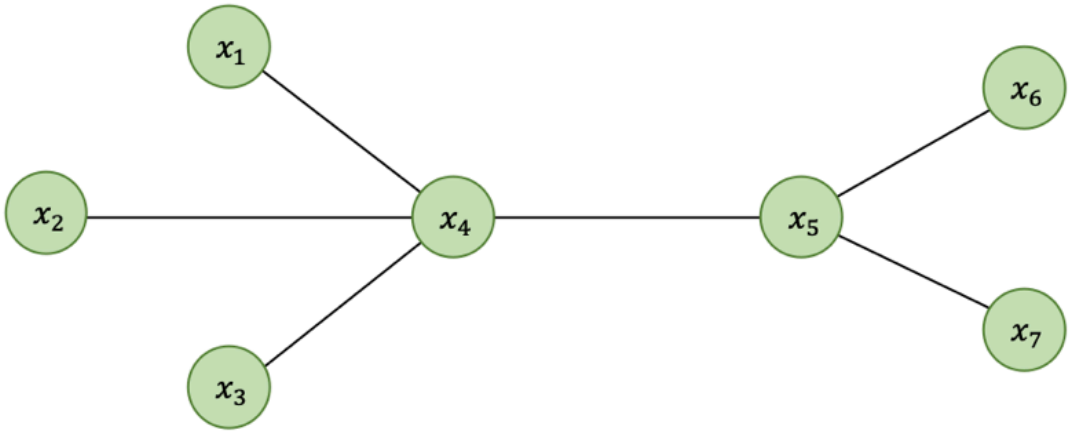
对于马尔科夫随机场中的每一个节点，通过消息传播，把该节点的概率分布状态传递给相邻的节点，从而影响相邻节点的概率分布状态，经过一定次数的迭代，每个节点的概率分布将收敛于一个稳态.所有的决策都可以通过直接计算通信节点的和解决。个人感觉有点像动态规划。消息传播通过累加来解决。

举个通俗的例子，

以前古代士兵报数，存在一个问题 若干士兵排成**一个队列**，每个士兵**只能与他相邻的士兵交流**，问如何才能**使每个士兵都知道总的士兵数**

从左到右传播一遍每个人知道自己左边有多少人，再从右到左传播一遍。每个人知道自己右边有多少人，最终人数就是L+R+1

考虑以下成对马尔可夫随机场



$P(x4)=$

$$Z^{-1}\mu_{1\to4}(x_4)\mu_{2\to4}(x_4)\mu_{3\to4}(x_4)\mu_{5\to4}(x_4)$$

其中 消息$\mu i\to j$是用来寻找边际概率而定义的可重用部分和

在有了以上基础知识和置信传播的概念后，要计算Z(T,p),我们还需要具体的ψ与φi计算方式。

对于一元因子变量φ的计算我们可以通过强制更改具体标签对应的信息权重来做到（以下称为phi参数）。作者用 multi-hot encoding来表示一个形态语法标签mi。具体来说，一个单词的phi是[0,1,0]代表它是阳性词，一个单词的psi是[0，0，1]代表它是阴性词。[1,0,0]代表它没有阳性阴性这一标签。如果我们对某个单词强制改变了其词性。可以通过改变其phi为[0,100,0]或[0，0，100]来使其在传播中具有更大权重。

对于二元因子变量ψ的计算，实际上就是要训练一个模型。根据两个词的标签给出一个量化的匹配程度。

作者的优化方法是基于梯度的优化。处理负对数− log（Pr（m | T，p)来作为树T的损失函数

具体的训练方法会在原理解析（5.1.7)里面介绍。

$$\psi(m_i, m_j \mid p_i, p_j, \ell),$$

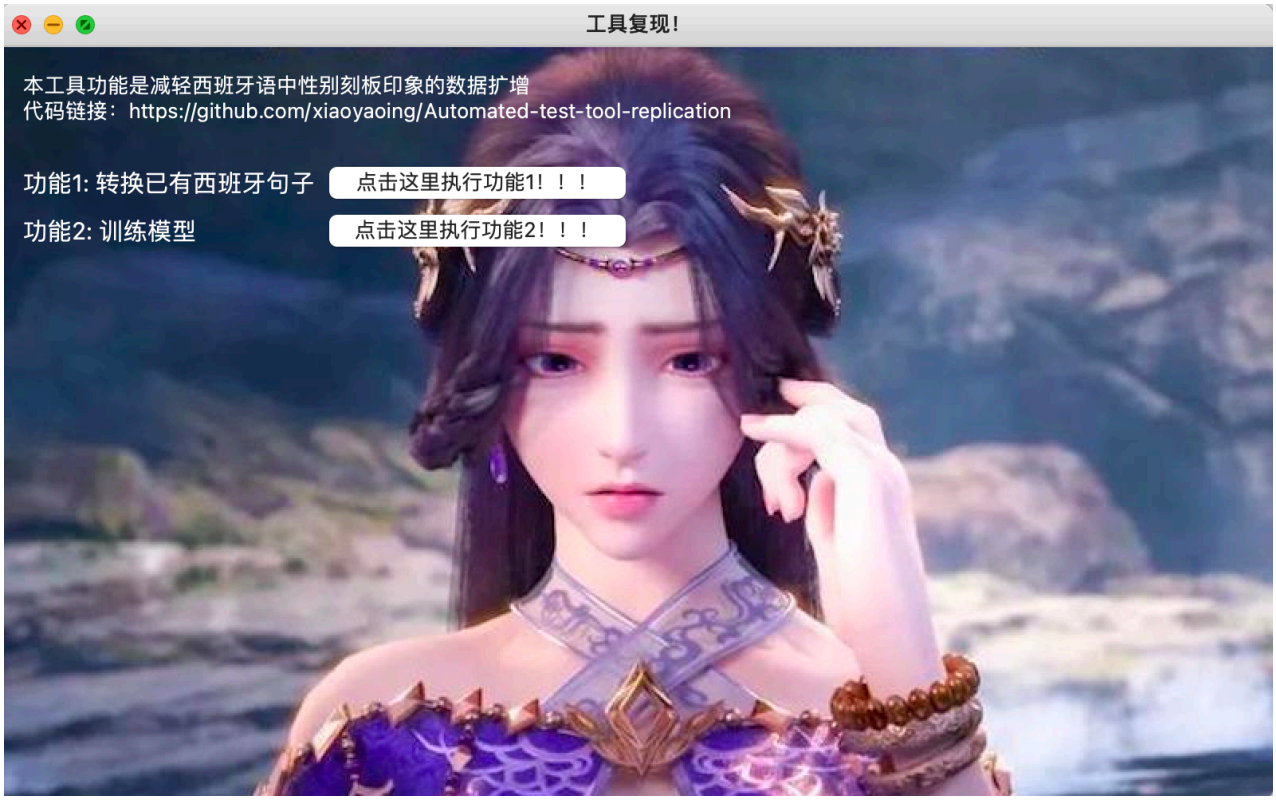
4.将句子映射为新形态，完成数据扩增

因为对于句子的转换是并非针对句子本身，而是针对[T，POS，M]，在改变M后，需要重新映射句子。论文作者提到这部分工作已经有人做的很好了，就直接用了现有模型。该模型的功能是给定一个单词的原型以及对应的语法形态标签，给出正确的单词形态。

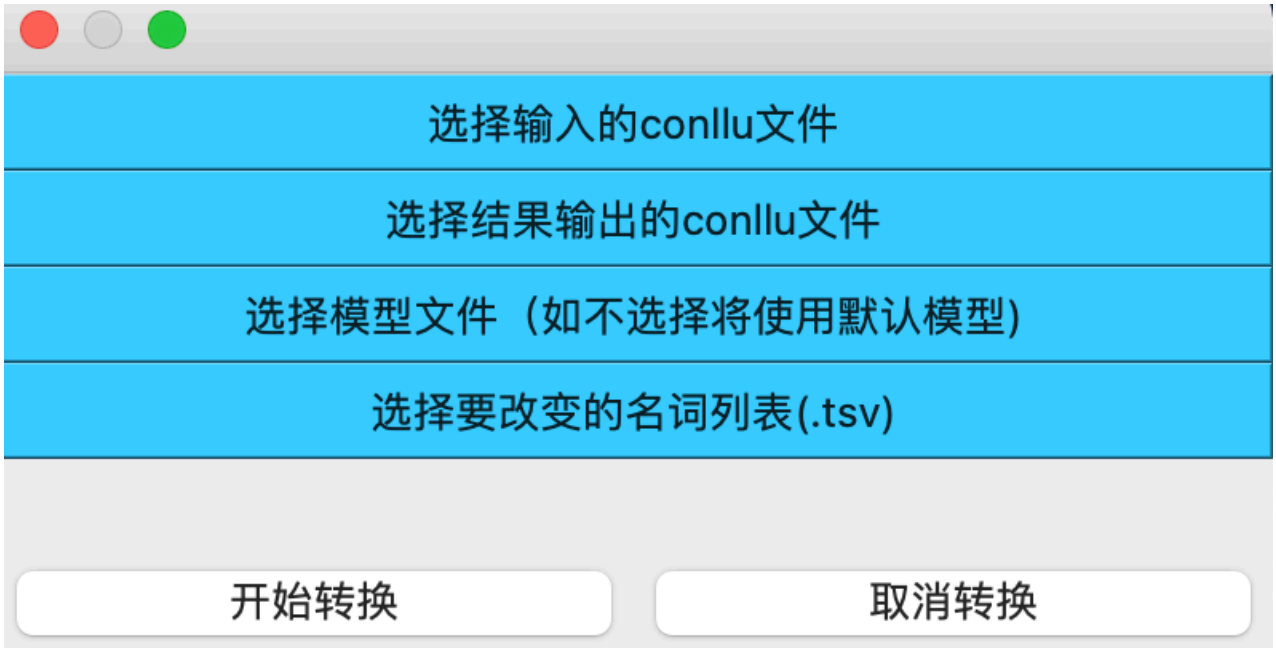
如：给定单词experto和标签[A;FEM,PL]该模型高概率会给出单词exper-tas，对于整个句子的每个单词应用，就可以完成句子的形态转换。

这边算是一个边角工作。作者用的是Wu等人提出的映射模型，所以我也偷懒用的这个用好的模型。

2.工具完成



2.1 功能一 转换已有西班牙语句子（数据扩增）



程序输入:输入的conllu（依存句法表示）文件， 以及要输出的conllu文件路径和性别化词列表 ,转换模型（可选，如不选择将使用自带模型）

程序输出：扩增转换后的conllu文件

示例 具体操作请见视频 这里只展示结果

反事实结果展示：

在input.conllu文件里 某一句子对应的部分：

```
3   atras   atras   ADV _   _   2   advmod  _   _
4   el  el  DET _   Definite=Def|Gender=Masc|Number=Sing|PronType=Art   5   det _   _
5   maestro maestro NOUN    _   Gender=Masc|Number=Sing 8   nsubj   _   _
6   Alberto alberto PROPN   _   Gender=Masc|Number=Sing 5   appos   _   _
7   Trujillo    trujillo    PROPN   _   Gender=Masc|Number=Sing 6   flat    _   _
8   promueve    promover    VERB    _   Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin   0   root    _
```

在output.conllu文件里，该句子对应的部分：

```
4   la  el  DET _   Definite=Def|Gender=Fem|Number=Sing|PronType=Art   5   det _   _
5   maestra maestra NOUN    _   Gender=Fem|Number=Sing  8   nsubj   _   _
6   alberta alberto PROPN   _   Gender=Fem|Number=Sing  5   appos   _   _
7   trujilla    trujillo    PROPN   _   Gender=Fem|Number=Sing  6   flat    _   _
8   promueve    promover    VERB    _   Mood=Ind|Number=Sing|Person=3|Tense=Pres|VerbForm=Fin   0   root
9   10  el
```

对应的阳性词转换为了阴性词

从整体看输入文件中的阳性词/阴性词=7861/5739

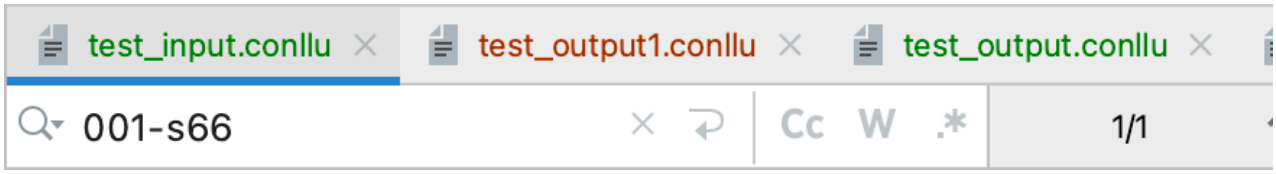
输出文件中的阳性词/阴性词=2104/1631

降低了性别刻板印象 该部分功能完成

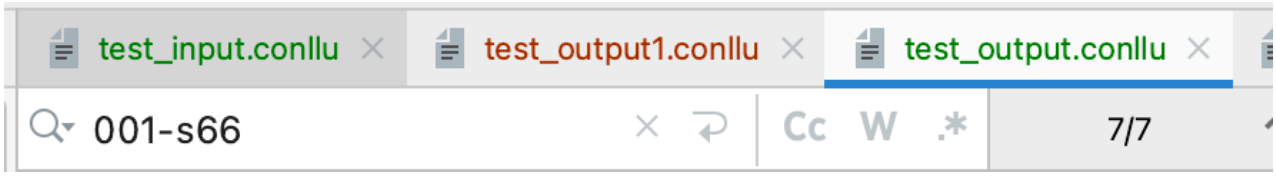
数据扩增结果展示

在输入文件中的一个句子被扩增为了输出文件中的多种形态 如例子：、

input文件中id为s66的句子出现次数

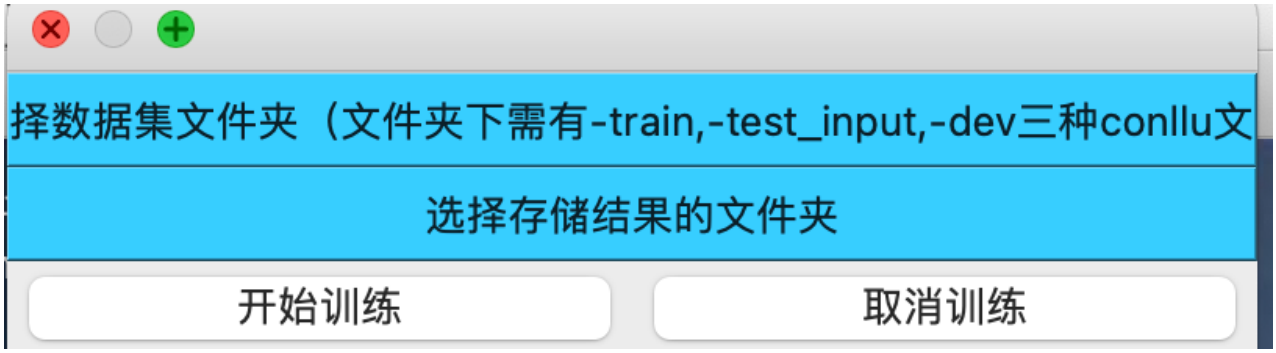


输出文件：



完成了数据扩增

2.2 功能二 训练模型



一中的功能需要依靠模型来寻找最优的替换策略来保证整句的语法结构特征以及正确性不受到影响，我提供了一个已经训练好的模型。不过用户也可以自行训练（需要提供数据集）

本程序会自动训练100次，并计算训练误差和生产误差，存储到结果文件夹 如图

3.工具验证

对于数据扩增的验证：

在输出文件中，原先的句子会有多种形态（因为转化序列有多种）。数据扩增比较两个conllu文件就可以明显的看出来。作者没有提到对这部分的验证，重点的验证在保证句子的形态和语法不能改变。在下面：

句子形态没有大程度改变的验证：

作者的验证办法：

作者在这篇文章里面做的工作之前没有研究过，作者说没有标准答案（即一个成对的句子语料库 一个句子的初始形态，和改变后的形态）。作者的做法是请了一些native speaker来给句子打注解。然后作者对一些包含性别词的句子做出干涉并转换句子形态。然后作者对打过注解的西班牙语句子计算了一个值（硬编码）作为基准线（硬编码就是把一个东西写死，不会因为输入改变）。然后对于改变后的句子，作者将其与原始标签句子进行比较（只关注除了性别词以外的词，**因为本工具很重要的一点是不只消除性别刻板形象，还要考虑在形态丰富的语言中（spanish）转换词性的时候，对于周围词的影响**）。然后计算出一个分数来验证转换的准确性。

我的验证办法：

考虑到我这也不太可能去找精通西班牙语的几个人帮我打注解。我就想了一种比较简单的验证。

对于转换前后的句子，我对相同id的句子来计算相似度。相似度的计算办法为对每个词计算相似度然后求平均。

对每个词的相似度计算办法为对每个词的feats集合 求交集大小与并集大小的比

下面是伪代码

```
def Test(path1,path2):
      sentences1=pyconll.load_from_file(path1)
      sentences2=pyconll.load_from_file(path2)
      simlaryity=0
      for s2 in sentences2:
            for s1 in sentences1:
              #if(isSameSentence(s1,s2)):
                  simlaryity+=computeSimlarity(s1,s2)
      simlaryity=simlaryity/len(sentences1)
      print("转换前后非性别词语法相似度为:" ,simlaryity)
```

验证结果是（我也不知道这个数值具体指高或者低，我这个验证办法太简陋了。。。）不过作者在他论文里面的验证结果很好！！！

> 转换前后非性别词语法相似度为： 0.9764

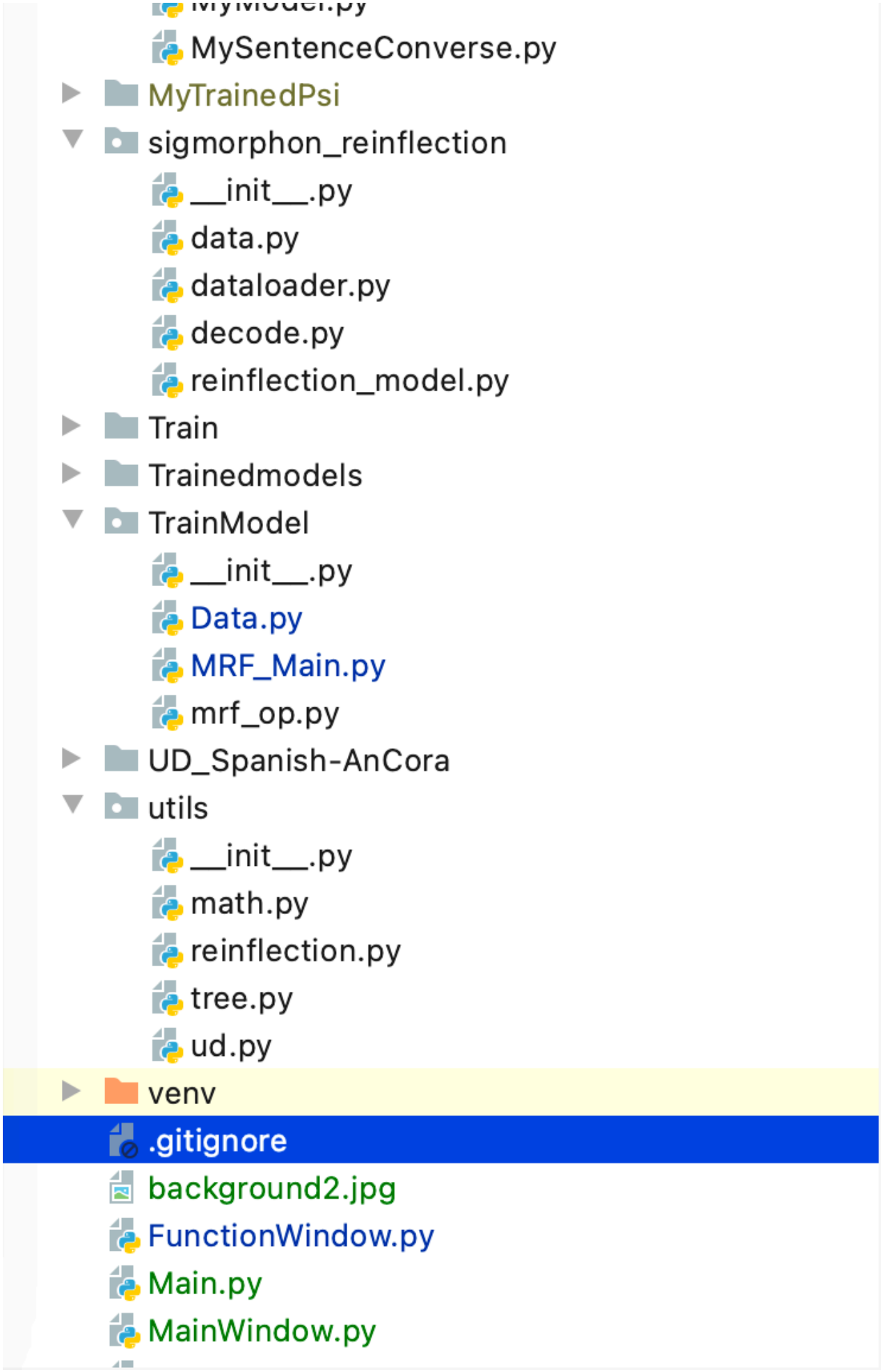
4.运行视频

详见https://www.bilibili.com/video/BV1Q44y1Y79v/

5.项目文档 & 原理解析

5.1项目目录

5.1.1 总览图


MySentenceConverse.py
MyTrainedPsi
sigmorphon_reinflection
__init__.py
data.py
dataloader.py
decode.py
reinflection_model.py
Train
Trainedmodels
TrainModel
__init__.py
Data.py
MRF_Main.py
mrf_op.py
UD_Spanish-AnCora
utils
__init__.py
math.py
reinflection.py
tree.py
ud.py
venv
.gitignore
background2.jpg
FunctionWindow.py
Main.py
MainWindow.py

下面逐步介绍每个模块（因为源码太多，放的话很臃肿，所以下面的代码均为伪代码 ）

5.1.2 MainFunction模块

这个模块是进行句子转换进行数据扩增的模块，也是功能一的实现模块。

antimate.py

主要功能是获取句子中可能的性别词转换序列

```
def getAnimates(conllu,animateF):
    """
    :param conllu: conllu对象
    :param animateList:  存储animate的文件
    :return: 返回可能的性别词变换排列
    """
    for sentence in conllu:
     for token in sentence:
       if  'Gender' not in token.feats or len(token.feats['Gender'])!=1 or
token.upos !="NOUN":
                continue
            changes.append([int(token.id),convertedWord,not masc])
                   for i in range(1,len(changes)+1):
            changesList.extend(combinations(changes,i))
           return  MySentenceConverse(deepcopy(sentence),change)
```

主要的处理逻辑为遍历每个句子中的每个token。

每个token持有一个成员属性feats。可以根据fetas来判断这个词是不是性别词，如果是性别词，就添加一种可能的替换，通过库函数combinations来组合各种替换的可能性。

组合的方法是这样的:

如果一个句子中有n个词是性别相关的。那么共有 C(n,1)+C(n,2)+...C(n,n)中变换序列（n个变换中选i个 1<=i<=n)

最终返回一个Mysentenceconverse类

MysentenceConverse.py

```
class MySentenceConverse:
    def __init__(self,sentence,changes):
        self.sentence=sentence
        self.changes=changes
    def applyModel(self,model,psi,reflectModel,device,decode_fn,decode_trg):
    def changeSentenceForm(self, tagsToChange, reflectModel, device, decode_fn,
decode_trg):
    def change_forms(self, form_idxs, reinflection_model, device, decode_fn,
decode_trg):
    def _change_line(self, token, reflectModel, device, decode_fn, decode_trg)
```

该类存有一个sentence成员变量来存储句子的原型（包括所有token，依存关系等）changes变量是一个三元组组成的列表（id,词，转换的性别）

applyModel则是通过MyModel.selectBestSequence来选择最佳的替换策略

得到最佳的替换策略以后通过changeform来改变句子的形态 完成最终的替换

MyModel.py

这部分代码主要功能是调用置信传播里的接口来获得最佳的转换序列

```
 class MyModel:
    def best_sequence(self, T, pos, psi, phi, fix_tags=[]):
      """
        :param T: tree
        :param pos: list of pos tags
        :param psi: psi potentials(二元变量)
        :param phi: phi potentials（一元变量）
        :param fix_tags: list of pairs of index of a word and the tag it should
be fixed to
        :return: 包含每个节点以及对应标签的字典
        """
        for idx, m in fix_tags:
            phi[idx - 1, m] = 100
        msgs, pointers = max_product(T, pos, psi, phi, True)
        tags_dict = get_best_tags(T, msgs, pointers)
        tags = []
        for i in range(1, len(T) + 1):
            tags.append(self.get_tag(tags_dict[str(i)]))
        return tags
```

对于需要修改的性别标签的干涉，这里是通过修改phi对应位置为100(phi创建时每一项都是1)。使其传播时具有更大权重。

maxproduct则是置信传播的一种累积方式（一个节点的信号量可以是周围信号量的累积作用（sum-product）也可以是选择影响最大的那个（maxproduct）

这个方法最终返回一个句子新的tags列表。

belief_propagation.py

这部分代码是置信传播的实现部分。

将信息分为4类:

```
# Types of messages
V_U = 0 变量到unary factor的信息
V_B = 1 变量到Binray factor的信息
U_V = 2 unary factor到变量的信息
B_V = 3 Binray factor到变量的信息
```

这里选的更新信息的方式是max-Product

置信传播：

信号的格式为

(Msg_type,i,j) -> Tensor(3)

先从叶到根部（句子的中心词）进行消息传播，从根到叶反向传播

```
def max_product(T, pos, psi, phi):
    """
    :param T:  Tree
    :param pos: list of pos tags
    :param psi: binary factor
    :param phi: unary  facotr
    :return: 因子图的信号量 以及最优标签的指向
    messages of nodes in the factor graph of the tree and backpointers to best
tags
    """
  msgs, pointers = _pass_msgs_from_leaves(T, pos, msgs, psi, phi, True,
pointers)
  msgs, pointers = _pass_msgs_from_root(T, pos, msgs, psi_transpose, phi, True,
pointers)
    return msgs, pointers
```

下面解释一下从叶到根部的传播算法原理

1.获得树中所有叶节点(变量节点以及根节点

2.构建所有（unary因子节点）到叶节点的消息栈（这是初始信息）

3.从栈中获取第一个可以用的信息(一个信息可用当切仅当其来自于一个因子而非变量，或者起点的所有子节点都已经被计算)

4.判断信息种类，根据信息来源做不同的传播。

这里以从变量到因子为例：

一个变量向因子传递信息，那么这个信息应该是这个变量收到的所有同类型的信息总和，然后进行更新。

这里是变量到因子传递的一个伪代码：

```
def _pass_msg_var(msgs, msg_type, i, j, use_log):
  ms = _get_msgs_to(msgs, i, False, [(U_V if msg_type == V_U else B_V, j)])
  msg = tr.sum(ms, 0) if use_log else tr.prod(ms, 0)
  msgs[(msg_type, i, j)] = msg
```

5.传播后向栈中加入下一个要计算的信息。

6.重复直到栈为空

从根到叶的传播类似

conllu.py

该文件负责将conllu句子从文件加载到内存

```
def loadSentences(n,file):
    """
    :param n:  加载几个句子
    :param file: file对象
    :return: conllu对象组成的list
    """
```

main.py 主要逻辑代码：

```
def main():
  conll=loadSentences(10000,f)
  sentenceConverseList=getAnimates(conll,opt.animate_list)
  for sentence in sentenceConverseList:

converted=sentence.applyModel(model,psi,reinflectModel,device,decode_fn,decode_
trg)
  return list[converted]
```

加载conllu文件->获取可能的变换->通过模型保证句子中性别词变换的同时句子的特征与语法不改变

5.1.3 MyTrainedPsi

放了跑的一些psi(两个数字分别代表训练误差和生产误差)

5.1.4 sigmorphon_reinflection

这个模块是完成重映射功能 这部分工作作者是借提到的Wu等人提出的模型来完成转换。不是论文的主要内容。于是我就直接用了Wu这些人的代码(QAQ)

Once the new tags have been inferred, the final step is to reinflect the lemmata to their new forms. This task has received considerable attention from the NLP community (Cotterell et al., 2016, 2017). We use the inflection model of Wu et al. (2018). This model conditions on the

5.1.5 Train

放了三个用来训练模型的文件（通用语料库中下载的西班牙语.conllu）

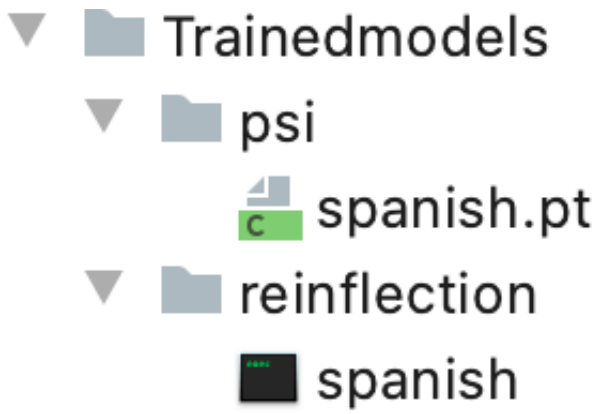
5.1.6 TrainedMoels

这个文件夹放了2个训练好的模型

本项目中用到了两个模型

模型1称为psi，对应论文中的二元因子 ψ，用来衡量两个单词的形态语法标签的匹配程度

模型2是用来重映射的



5.1.7 TraineModel

这个模块是用来训练psi模型的。目的是将二元因子 ψ参数化。作者提出了两种参数化的方式。一种是线性参数化一种是神经网络参数化。一方面我实在是看不太懂神经网络参数化那部分内容，另一方面作者最后的实验中提到了线性参数化的效果要明显好于神经网络参数化（作者也不知道这具体是为什么hhh），考虑到这两个问题我就只弄了线性参数化。

MRF_Main.py

这个文件是入口函数，接受参数以及读取文件等 首先我们关注NeuralMRF类的定义

```
class NeuralMRF(nn.Module):
    def __init__(self, data, out_dir, linear, use_v1, hack_v2):
        super(NeuralMRF, self).__init__()
        self.data = Data(data + "/-train.conllu", data + "/-dev.conllu", data +
"/-test_input.conllu", use_v1, hack_v2)
        self.num_pos = self.data.num_pos()
        self.num_labels = self.data.num_labels()
        self.num_tags = self.data.num_tags()
        self.out_dir = out_dir
        self.linear = linear
        self.mrf = MRF_Lin(self.data.tags)
        self.register_parameter('psi', None)
        self.psi = nn.Parameter(
            torch.randn(self.num_pos, self.num_pos, self.num_labels,
self.num_tags, self.num_tags,
                        dtype=torch.float64))
```

我们可以看到，该类的初始化函数会通过Data类加载conllu数据到内存中。并且获得所有句子中的所有pos，laebs，tags。然后会初始化一个psi参数。我们之前提到，psi的作用是计算以下公式

$$\psi(m_i, m_j \mid p_i, p_j, \ell),$$

是一个(Mi,mj,pi,pj,l)到val的映射。所以psi应该是一个五元化的参数。这里通过Torch库函数创建。

这个文件里除了创建模型，初始化参数，还需要承担优化模型的功能。优化模型是通过fit函数实现的。fit的实现逻辑如下：

```
def fit(self, epochs=100, precision=1e-5):
    #优化开启时，通过torch.optim.Adam创建优化器对象，与psi绑定
    self.optimizer = optim.Adam(self.parameters(), lr=0.001,
weight_decay=0.001)
     for i in range(epochs):
            print("Computing epoch", i + 1, "...")
            # 模型优化
            train_loss, dev_loss = step()
            # 保存现在的参数
            file = os.path.join(self.out_dir, "psi_" +
                                str(round(train_loss[0].item(), 6)) + "_" +
```

```
                                    str(round(dev_loss[0].item(), 6)) + "_epoch" +
str(i + 1) + ".pt")
            torch.save(self.psi, file)
  def step():
            train_loss = dev_loss = 0
            print("=优化参数ing")
            for sentence in self.data.test:
            #梯度
                self.optimizer.zero_grad()
                loss = self.forward(sentence)
                train_loss += loss
                loss.backward()
                self.optimizer.step()
                del loss
            print("  Calculating dev loss")
            for sentence in self.data.dev:
                count += 1
                dev_loss += self.forward(sentence)
            return train_loss / len(self.data.train), dev_loss /
len(self.data.dev)
```

#优化开启时，通过torch.optim.Adam创建优化器对象，与psi绑定。

然后在step函数中通过前向传播(具体过程请看下文mrf_op.py中的调用)对训练数据集计算训练损失。调用优化器的step（torch的库函数）来对psi参数进行优化。同时会计算生产训练集的训练损失。fit函数的参数epochs决定了优化多少次。在每次优化完毕后，通过torch.save存储模型。

mrf_op.py

主要逻辑为在mrf_op.py文件中定义一个线性参数化的马尔可夫模型（继承torch,nn.Module来实现）

```
class MRF_Lin(torch.nn.Module):
    """
    线性参数化初始化置信传播模型
    """
    def __init__(self, tags, sentence=None):
        super(MRF_Lin, self).__init__()
        self.model = Model(tags)
        self.sentence = sentence
    def forward(self, psi):
        """
        :param psi参数
        :return: 模型对当前句子的应用
        """
        val = -self.model.log_prob(self.sentence.T, self.sentence.pos,
self.sentence.m, psi)
        return torch.Tensor([val])
```

这个实际上就是通过psi对当前的句子进行条件概率计算（给定T，pos，m）。通俗点讲可以看成是**打分**因为这些句子都是没有处理过的。从通用语料库中下载的**非常地道，绝对正宗**的西班牙语句子。我们有理由认为:**一个psi对这些句子打分越好，这个psi就越准确** 这里加上一个负号就可以把它看成训练误差!

打分的具体函数在MainFunction/MyModel.py中

下面是给句子打分的逻辑以及伪代码

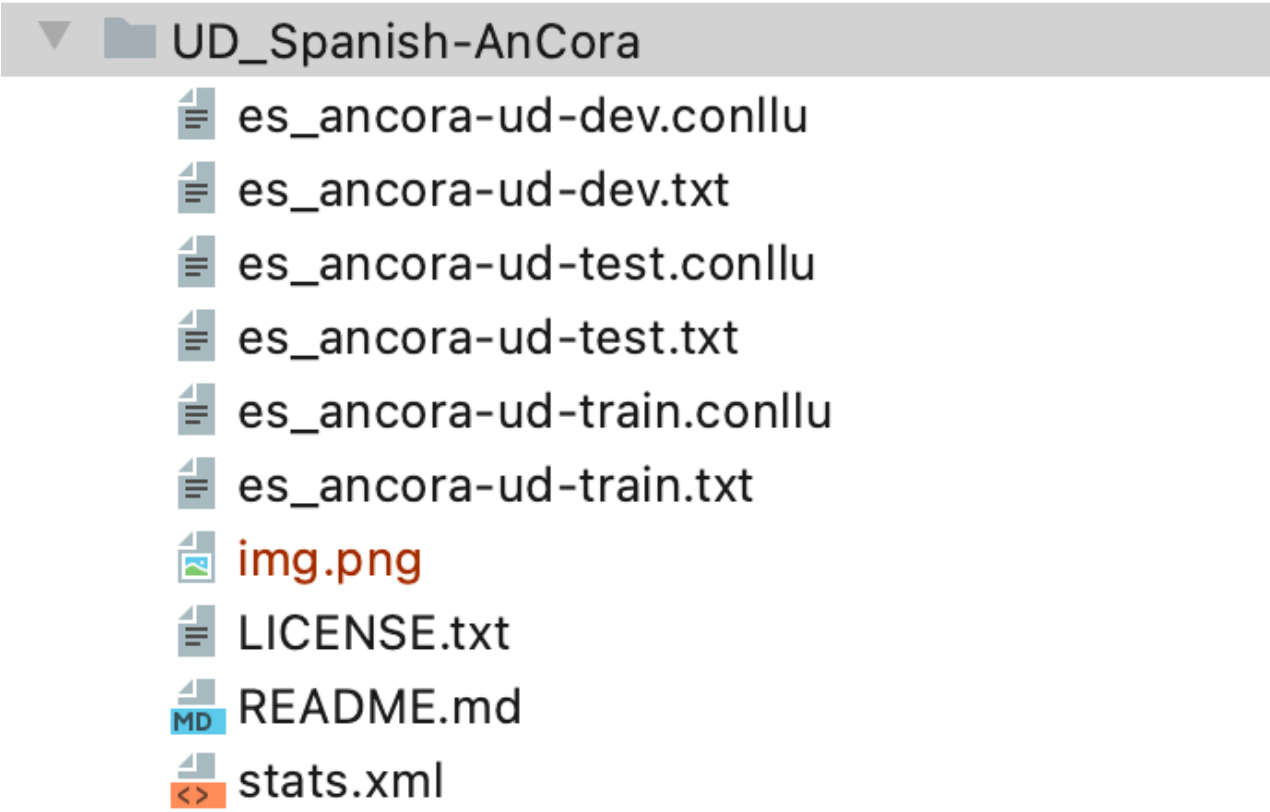
其中log_score是对句子整体的phi以及psi标签计算并加和 取对数

logZ是计算配分函数Z的对数的置信传播算法(目的是将概率归一化 因为是取了对数，所以是用减法)

```
    def log_prob(self, T, pos, m, psi, phi=tr.Tensor()):
        if phi.size() == tr.Size([0]):
            phi = self.create_phi(T, pos, m)
        return self.log_score(T, pos, m, psi, phi) - self.logZ(T, pos, psi,
phi)
```

5.1.8 UD_Spanish-AnCora

hhh 这里放的是我从通用语料依赖库中下的西班牙语的语料



5.1.9 utils

这是一个工具模块

主要包括对句子解析成三元组表示的树结构，进行重映射，获取词语的特征（如性别，词性）等

5.1.10 图形化部分

这部分主要是pyqt相关 主要是为了美化hhh，因为和论文内容没有相关性我就简单带过。

Main.py

这个文件是图形界面的入口。

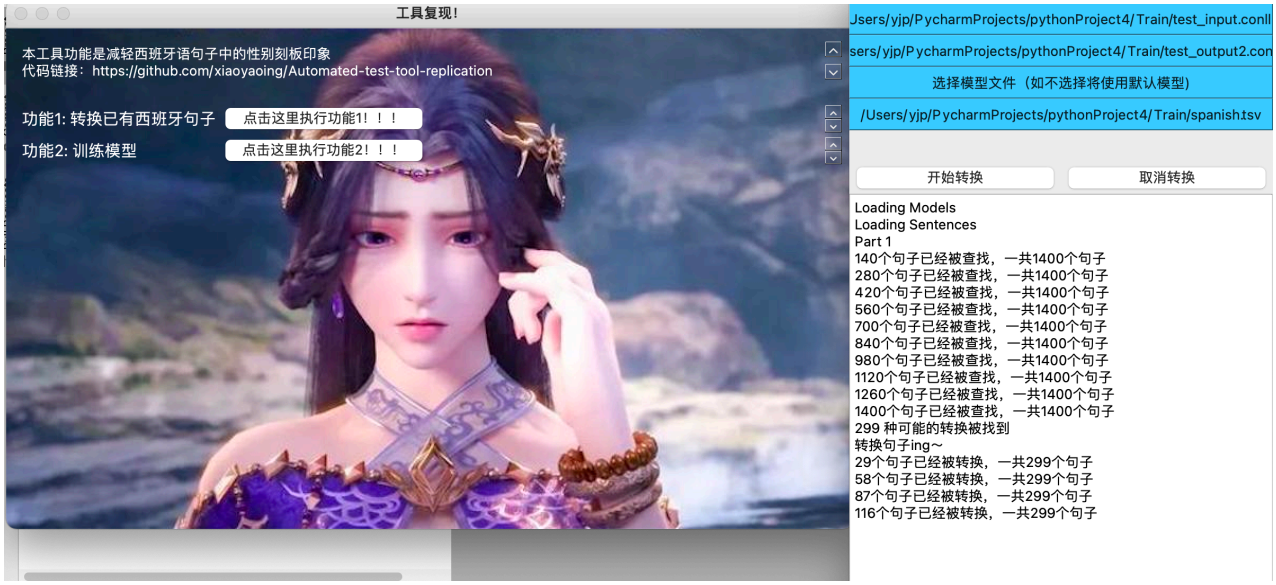
```
import sys
from PyQt5.QtWidgets import QApplication
from MainWindow import MyWindow
def main():
    App=QApplication(sys.argv)
    MainWindow=MyWindow()
    sys.exit(App.exec())
if __name__ == '__main__':
    main()
```

MainWindow.py

这个文件里面定义了程序的主窗口，就是工具完成那里放出的那个图表示的情况。主窗口里面有两个按钮，分别对应这个项目的两个功能。转换句子和训练模型。如果点的话会创建一个functionWindow。

FunctionWindow.py

这个文件里面定义了两个类，分别是执行功能1的窗口类和功能2的窗口类。在使用者按要求选择文件以后，点击相应的执行按钮时，程序会创建一个新线程来执行对应任务。并通过输出重定向来将原本输出到控制台的信息输出到窗口处。下图是一个例子（转换句子） **当然除了图形化界面用户也可以通过命令行直接单独地运行2个功能**



6. 运行说明

依赖说明：python3.7 torch1.3.1 如果需要图形化界面则还需要pyqt5的包

也可以用我打包好的程序，这样的话就不用下载乱七八糟的依赖了(macos 11.6.1可以运行，别的俺没有测试) 程序链接:这里是微云链接

图形化运行：在项目根目录运行 python Main.py 然后根据提示完成操作（以下两种功能都可以）

句子转换(命令行)：在 MainFunction/ 运行 **python main.py --in_files arg1 --out_file arg2 --animate_list arg3**

三个参数分别是输入输入出的conllu文件，以及名词列表 这三个参数是必需的

同时可以加入可选参数 --psi ，来选用自己的模型来转换。如果没提供这个参数，这里会用之前训练好的模型来转换

模型训练(命令行)

在/TrainModel下运行 **python MRF_Main.py --data arg1 --out-dir arg2**

其中arg1是数据文件夹路径，文件夹下需要有-train.conllu -dev.conllu -test.conllu三种数据集

arg2是模型存储的文件夹

7.代码规范

模块化我觉得还是蛮清楚的。注释这方面每个方法基本上都加了注释（QAQ）

```
def getAnimates(conllu,animateF):
    """
    :param conllu: conllu对象
    :param animateList:  存储animate的文件
    :return: 返回可能的性别词变换排列
    """
```

8.复现难度

俺觉得挺难滴，不知助教以为（QAQ）

9.最后的感想

很感谢自动化测试这门课，不仅学到了很多知识，也算是接触了科研，还极其地锻炼了我的心态（笑）。第一次读论文，第一次复现论文都是这门课带给我的。读论文的过程也确实是美妙又痛苦的。美妙的是里面的设计真的很精巧，算法很优雅，知识很丰富。痛苦的是很多专业名词，很多算法很难懂。我这个一点ai基础都没有的人，在这一个多月先后接触了马尔可夫随机场，因子图，置信传播，深度学习，梯度优化等等概念，也算是深度了解了conllu文件这种依存格式，体会到了调包pytorch的快乐hhh，忙里偷闲还学了pyqt来写个图形化界面。一开始是真的痛苦，完全不知道怎么搞，整个论文一遍遍的看头都秃了都觉得这不可能写的出来。

总之，这门课让我学到了很多，也进行了实践，前瞻了软件测试前沿技术，用自己的能力简单还原这样一份工作。

我不敢说对论文的理解足够透彻，对工具的复现足够好，这一个多月认真努力去完成这个任务，熬了不少夜，是真的收获了不少。

当然同时也非常感谢这门课的房春荣、陈振宇老师和所有助教老师们！！！要让一百多个人的工具复现大作业的题目各不相同，还要给每个人提供对应的论文，无疑这是工作量巨大的宏伟工程。我们读一篇论文尚且费劲，何况他们会读几十篇、甚至几百篇的论文。他们为了这一门课的教学质量牺牲了很多自己的时间。我能够选上这门课并且坚持到最后，何其有幸。