针对序列文本数据创建条件随机场

- import os # 文件管理包import argparse # 是python的一个命令行解析包
- import string
- import pickle # 该模块实现了用于序列化和反序列化Python对象结构的二进制协议
- import numpy as np # 科学计算包
- import matplotlib.pyplot as plot # 画图的包from pystruct.datasets import load_letters # 数据集
- from pystruct.models import ChainCRF # CRF模型
- from pystruct.learners import FrankWolfeSSVM
- # 定义一个函数来解析输入参数。我们将C的值作为输入参数# 参数C控制错误分类的惩罚力度，C的值越大意味着在训练中的错误分类罚力度越大# 但是我们可能会结束过拟合的模型。另一方面，如果我们选择一个较小的C值，我们将使得模型适应性更好# 但是这也意味着我们对错误分类施加了更低的惩罚
- def buid_arg_parser():
    - parser = argparse.ArgumentParser(description='train a Conditional\Random Field classifier')
    - parser.add_argument("--C", dest="c_val", required=False, type=float, default=1.0, help='C value to be used for training')
    - return parser
- # 定义一个处理所有构建CRF模型的类。我们将使用链CRF模型
- class CRFModel(object):
    - def __init__(self, c_val=1.0):
        - self.clf = FrankWolfeSSVM(model=ChainCRF(), C=c_val, max_iter=50)
    - def load_data(self):
        - alphabets = load_letters()
        - x = np.array(alphabets['data'])
        - y = np.array(alphabets['labels'])
        - folds = alphabets['folds']
        - return x, y, folds
- # 训练CRF
- def train(self, x_train, y_train):
    - self.clf.fit(x_train, y_train)
- # 定义评估函数：
- def evaluate(self, X_test, Y_test):
    - return self.clf.score(X_test, Y_test)
- # 使用未知数据点训练CRF
- def classify(self, input_data):
    - return self.clf.predict(input_data)[0]
- # 定义提取子串函数
- def convert_to_letters(indices):

```
    alphbets = np.array(list(string.ascii_lowercase))
    # 提取
    output = np.take(alphbets, indices)
    output = ''.join(output)
    return output
# 定义主函数
if __name__ == '__main__':
    args = buid_arg_parser().parse_args()
    c_val = args.c_val
    # 定义CRF模型
    crf = CRFModel(c_val)
    # 加载训练数据和测试数据
    x, y.folds = crf.load_data()
    X_train, X_test = x[folds == 1], x[folds != 1]
    Y_train, Y_test = y[folds == 1], y[folds != 1]
    # 训练CRF模型
     print('\n训练CRF模型')
    crf.train(X_train, Y_train)
    # 评估准度
    score = crf.evaluate(X_test, Y_test)
    print('\nAccuracy score =', str(round(score * 100, 2)) + '%')
    indices = range(3000, len(Y_test), 200)
    for index in indices:
        print("\nAccuracy = ", convert_to_letters(Y_test[index]))
        predicted = crf.classify([X_test[index]])
        print("Predict =", convert_to_letters(predicted))
```