Problem 1

Team Members: 徐光正 2016012856, 郭峥岩 2018011340, 张鹤潇 2018011365

Problem: Chinese Emotion Classification

Data sources: NLPCC 2014 Evaluation Tasks, Emotion Analysis in Chinese Weibo Texts

Introduction:

Emotion Classification is a basic problem in the NLP field, where a lot of previous work can be used for reference. The data set we use is an open-source corpus from Sina Weibo, where the texts are divided into two categories according to whether or not opinionated. There are about 15,000 positive samples and 30,000 negative samples in the training set.

We team want to compare the performance of different methods from traditional machine learning models such as Naive Bayes, to deep learning models using CNN, RNN or even GNN, then to state-of-the-art pre-trained models. We hope to deepen our understanding of relevant methods by analyzing the advantages and disadvantages of them and try to make some improvements.

Problem 2

The pseudocode for the regression tree is as follows (in python syntax):

```
class Node:
    def __init__(self, value=None):
        self.lchild = None
        self.rchild = None
        self.cut_feature = None
        self.cut_point = None
        self.value = value

    def split(self, samples, labels):
        ''' samples: sample_size * feature_num
            labels : feature_num '''
        def do_split(self, col, labels, cut_point):
            '''col: the selected feature of training data'''
            labels_l = labels[col < cut_point]
            labels_r = labels[col >= cut_point]
```

```
            avg_l = mean(labels_l)
            avg_r = mean(labels_r)
                        mse = (sum((labels_l - avg_l)**2) +
                                sum((labels_r - avg_r)**2)) / len(labels)
            return avg_l, avg_r, mse

        min_loss = INF
        avg_l, avg_r = None, None
        for feature in features:
            for cut_point in cut_points:
                                avg_l_tmp, avg_r_tmp, loss =
                                        do_split(samples[:, ], labels,
                                            cut_point)
                    if min_loss > loss:
                        min_loss = loss
                        avg_l, avg_r = avg_l_tmp, avg_r_tmp
                        self.cut_feature = feature
                        self.cut_point = cut_point
        self.lchild = Node(avg_l)
                self.rchild = Node(avg_r)

class Tree:
    def __init__(self, X, y):
        self.root = Node()
        # BFS
        node_queue = queue.Queue()
        node_queue.put((self.root, X, y))
        while not node_queue.empty():
            node, samples, labels = node_queue.get()

            # terminal condition can be the max height of the tree
            # or the min samples needed to split a node
            if "terminal conditions are satisfied":
                break
```

```
            node.split(samples, labels)
            # Pick the samples that belong to the two subregions
            idx_l = (samples[:, node.cut_feature] < node.cut_point)
            idx_r = (samples[:, node.cut_feature] >= node.cut_point)

            node_queue.put((node.lchild, samples[idx_l, :], labels[idx_l]))
            node_queue.put((node.rchild, samples[idx_r, :], labels[idx_r]))

    def predict(self, X):
        def predict_one(sample):
            node = self.root
            while node.lchild and node.rchild:
                if sample[node.cut_feature] < node.cut_point:
                    node = node.lchild
                else:
                    node = node.rchild
            return node.value
        return [predict_one(sample) for sample in X]
```