人工智能Lab2实验报告

PB21000164 来泽远

决策树

训练

```
    def generate_node(self, X, y):
        # X: [n_samples_train, n_features],
        # y: [n_samples_train, ],
        # TODO: implement decision tree algorithm to train the model
        #生成节点
        self.tbar.update(1)
        X = X.reset_index(drop=True)
        y = y.reset_index(drop=True)
        new_node = self.Node()
        y_list = list(np.array(y).reshape(-1))
        #如果所有样本都属于同一类别，则标记为叶节点，return
        if len(set(y_list)) == 1
            new_node.value = y_list[0]
            return new_node
        #如果A为空，或者D中样本在A上取值相同，则标记为叶节点，值为D中样本数最多的类，return
        def check_eq(X):
            for i in X:
                if len(set(X[i])) != 1:
                    return False
            return True
        max_cnt = 0
        for i in set(y_list):
            if sum(y_list==i) > max_cnt:
                max_cnt = sum(y_list==i)
        if len(X.columns) == 0 or check_eq(X):
            new_node.value = max_cnt
            return new_node
        #选择最优划分特征
        best_feature, best_threshold = self.divide(X, y)
        print(f"best_feature: {best_feature}, best_threshold: {best_threshold}")
        new_node.feature = best_feature
        new_node.threshold = best_threshold
        if best_feature in continue_features:
            X1 = X[X[best_feature] <= best_threshold]
            y1 = y[X[best_feature] <= best_threshold]
            X2 = X[X[best_feature] > best_threshold]
            y2 = y[X[best_feature] > best_threshold]
            if len(y1) != 0: new_node.child.append(self.generate_node(X1, y1))
            else: new_node.child.append(self.Node(value=max_cnt))
            if len(y2) != 0: new_node.child.append(self.generate_node(X2, y2))
            else: new_node.child.append(self.Node(value=max_cnt))
        else:
            for i in range(discrete_features_size[best_feature]):
                X_i = X[X[best_feature] == i].drop(columns=[best_feature])
                y_i = y[X[best_feature] == i]
```

```
                if len(y_i) != 0: new_node.child.append(self.generate_node(X_i,
y_i))
                else: new_node.child.append(self.Node(value=max_cnt))
        return new_node
```

按照伪代码实现，相应的注释在代码当中。首先生成一个新节点，并且把X、Y的索引重新排序（为了后续顺序访问）。其次，判别是否为同一类别、是否取值相同、是否为空并做相应处理。再进行最优划分，按照西瓜书所述，如为离散特征则直接去除，若为连续特征则保留，以获得更高的准确率。

划分最优特征

```
    def divide(self, X, y):
        best_feature = None
        best_threshold = None
        best_gain = -1000
        #计算y的信息熵
        ent = 0
        for i in set(y['NObeyesdad']):
            p = sum(y['NObeyesdad']==i) / len(y['NObeyesdad'])
            ent -= p * np.log2(p)
        #计算信息增益
        for i in X:
            if i not in continue_features:
                gain = ent
                for j in set(X[i]):
                    #计算条件熵
                    ent_j = 0
                    for k in set(y['NObeyesdad']):
                        p = sum((X[i]==j) & (y['NObeyesdad']==k)) / sum(X[i]==j)
                        if p != 0: ent_j -= p * np.log2(p)
                    gain -= sum(X[i]==j) / len(X[i]) * ent_j
                if gain > best_gain:
                    best_gain = gain
                    best_feature = i
            else:
                record = 0
                th = 0
                for threshold in set(X[i]):
                    #计算信息增益
                    gain = ent
                    ent_1 = 0
                    ent_2 = 0
                    for k in set(y['NObeyesdad']):
                        if sum(X[i]<=threshold) > 0: p1 = sum((X[i]<=threshold)
& (y['NObeyesdad']==k)) / sum(X[i]<=threshold)
                        else : p1 = 0
                        if sum(X[i]>threshold) > 0:  p2 = sum((X[i]>threshold) &
(y['NObeyesdad']==k)) / sum(X[i]>threshold)
                        else : p2 = 0
                        if p1 != 0: ent_1 -= p1 * np.log2(p1)
                        if p2 != 0: ent_2 -= p2 * np.log2(p2)
                    gain -= sum(X[i]<=threshold) / len(X[i]) * ent_1 +
sum(X[i]>threshold) / len(X[i]) * ent_2
                    if gain > record:
                        record = gain
                        th = threshold
                    if gain > best_gain:
```

```
                        best_gain = gain
                        best_feature = i
                        best_threshold = threshold

        return best_feature, best_threshold
```

区分离散特征与连续特征。若为离散则枚举所有存在的值并分为一类，为他们计算信息增益；若为连续则枚举所有存在的值作为阈值，按照左右两边分类，将其中信息增益最大对应的阈值作为划分连续特征的阈值。

分类

```
    def next_node(self, X):
            if self.value is not None:
                return self.value
            if self.feature in continue_features:
                if X[self.feature] <= self.threshold:
                    return self.child[0]
                else:
                    return self.child[1]
            else:
                return self.child[int(X[self.feature])]
```

节点中内置next函数，如为非叶节点则按离散或者连续返回下一个节点；若为叶节点（有分类值），则返回分类值。在对一个样本分类只需要while循环进行next即可。

正确率

```
1:05,  2.02it/s]test 0.9479905437352246
train 1.0
```

在测试集正确率为94.7%，训练集接近100%。

PCA&KMeans

PCA

```
    def fit(self, X:np.ndarray):
        # X: [n_samples, n_features]
        # TODO: implement PCA algorithm
        X = X - np.mean(X, axis=0)
        X = X.T
        n_samples = X.shape[0]
        cov = np.zeros((n_samples, n_samples))
        for i in range(n_samples):
            for j in range(n_samples):
                cov[i, j] = self.kernel_f(X[i], X[j])
        eig_val, eig_vec = LA.eig(cov)
        idx = np.argsort(-eig_val)
        self.components = eig_vec[:, idx[:self.n_components]]
        X = X.T
```

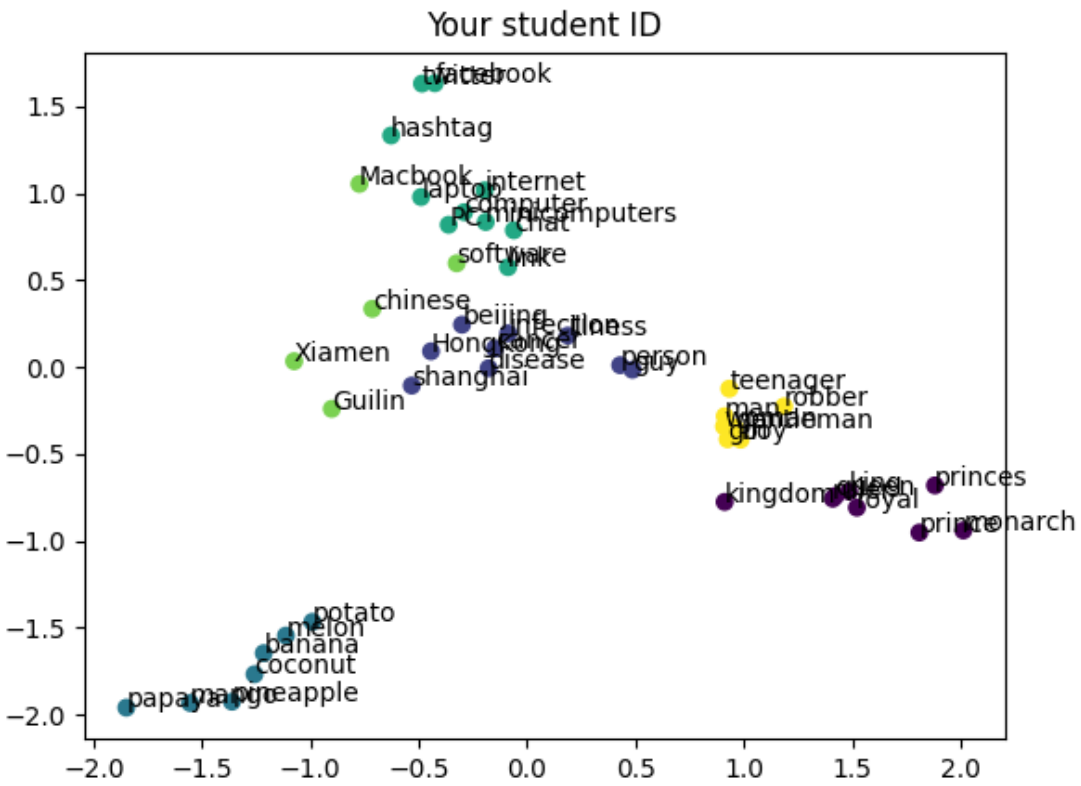
训练过程就是求投影矩阵，按照伪代码即可。

KMeans

```
    def fit(self, points):
        # TODO: Implement k-means clustering
        self.centers = self.initialize_centers(points)
        for i in range(self.max_iter):
            labels = self.assign_points(points)
            centers = self.update_centers(points)
            if np.all(labels == self.labels):
                break
            self.labels = labels
            self.centers = centers
```

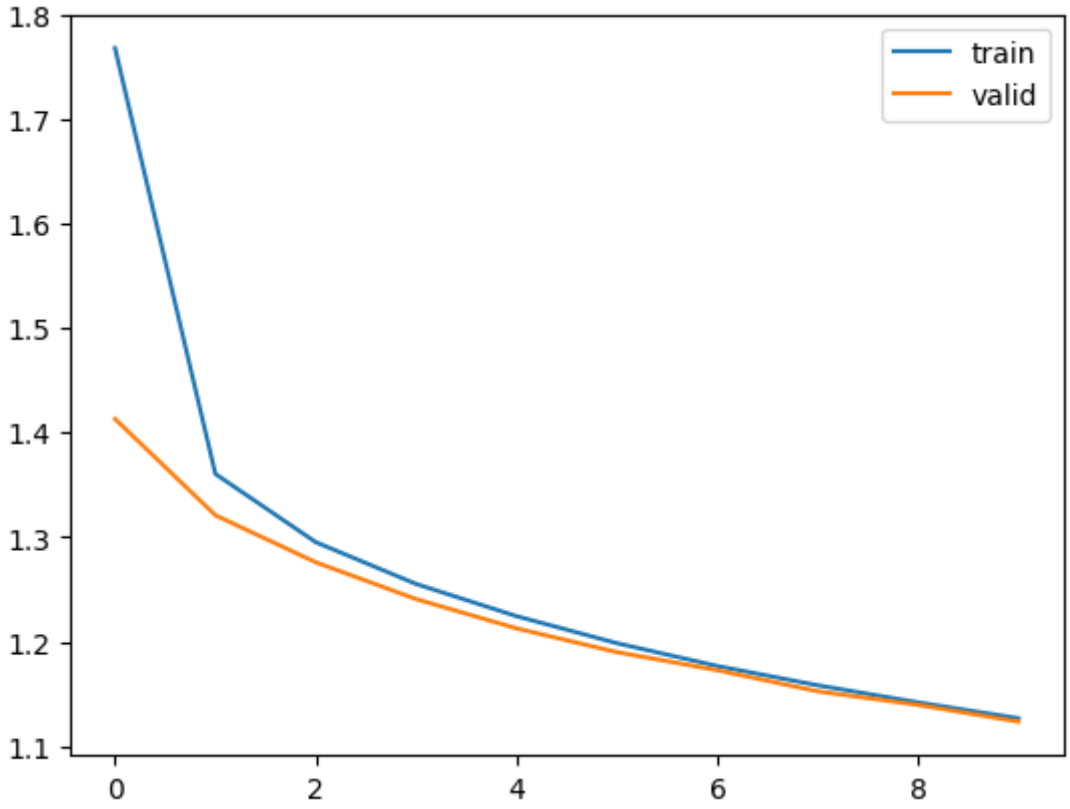
由于KMeans算法可以收敛，只需要不断将每一个样本归到k个类中的某一个（距离最近），再更新k类的中心（平权算术平均），直到没有变化即可。

散点图



MoE

Loss图像



训练了10个epoch，可以看到模型的收敛性还是不错的，我将其主要归功于以下几点：

1. 选段素材的数据量比较小且优质（内在统计规律好），这使得模型易于拟合
2. 正确使用了torch库函数完成一些操作，同时保证了可导性（如mask的实现、下标访问等，按照非torch的有些写法是不可导的，最好还是多用torch库函数）
3. SoftMax自带标准化，加上模型内添加的一些norm层，使得收敛更快。
4. MoE方法本身的优越性。

生成效果

```
I could pick my lance
That the state of the world will be so much
To seek to the senate, and the rest, the county strengt
```

由于是字符级别的训练，我们看到大部分单词都拼写正确足以证明其优越性。并且模型也学习到了基础的语法规则（主谓宾等），以及莎士比亚的用词习惯。

Bonus

前缀

```
J!!!!!!!! altru Marshall wild hust TV enjoyable living side reun flee window
pestPot outlook eleg Nora bu wanna mixer res  "
```

完整内容

```
 J!!!!!!!! altru Marshall wild hust TV enjoyable living side reun flee window
pestPot outlook eleg Nora bu wanna mixer res  "This is great! I love living on
the wild side!

From that day forth, Nora and Marshall the Fun Girl were great friends. Whenever
Josenzie's was at home, she spent most of her room and listened to all of her
gentleness.
```

Loss曲线