任务一：基于机器学习的文本分类

实现基于logistic/softmax regression的文本分类

要求

1. 参考
    1. 文本分类
    2. 《神经网络与深度学习》 第2/3章
2. 数据集：Classify the sentiment of sentences from the Rotten Tomatoes dataset
3. 实现要求：NumPy
4. 需要了解的知识点：
    1. 文本特征表示：Bag-of-Word，N-gram
    2. 分类器：logistic/softmax regression，损失函数、（随机）梯度下降、特征选择
    3. 数据集：训练集/验证集/测试集的划分
5. 实验：
    1. 分析不同的特征、损失函数、学习率对最终分类性能的影响
    2. shuffle 、batch、mini-batch
6. 时间：两周

数据集划分

分别为 **训练集**、 **测试集**、 **验证集**

调用 `sklearn` 中的 `train_test_split` 方法划分

特征表示

- Bag-of-Word（将文本看为词的集合）
    - 建立词汇字典

      python实现为tuple，赋值为训练集合中的词下标索引

```
# 构建文档词汇字典
# 相当于sklearn 中 CountVectorizer.fit方法
def fit(self, data_list):
    for sentence in data_list:
        if self.do_lower_case:
            sentence = sentence.lower()
            words = sentence.strip().split(" ")
            for word in words:
                if word not in self.vocab:
                    # 元素值为下标索引值
                    self.vocab[word] = len(self.vocab)
```

    - 建立 document-term matrix

      每个样本表示为一个 $|v|$ 为向量，第 i 维的值表示词表中的第 i 个次在样本中出现的次数

```
# document-term matrix, count the frequency
# 相当于sklearn 中 CountVectorizer.transform方法
def transform(self, data_list):
    vocab_size = len(self.vocab)
    document_term_matrix = []
    for idx, sentence in enumerate(data_list):
        temp = np.zeros(vocab_size,dtype='int8')
        if self.do_lower_case:
            sentence = sentence.lower()
            words = sentence.strip().split(" ")
            for word in words:
                vocab_idex = self.vocab[word]
                temp[vocab_idex] += 1

        document_term_matrix.append(temp)
```

```
        return document_term_matrix
```

  - 以上过程调用 `CountVectorizer.fit_transform` 方法即可实现sklearn.feature_extraction.text.CountVectorizer — scikit-learn 1.0.2 documentation

- N-gram（考虑词序信息）
  - 多了邻近的几个单词的拼接过程

```
for i in range(len(words) - gram + 1):
    n_gram_word = "_".join(words[i : i + gram])
    if n_gram_word not in self.ngram_vocab:
    # 元素值为下标索引值
    self.ngram_vocab[n_gram_word] = len(self.ngram_vocab)
```

  - 以上过程调用 `CountVectorizer.fit_transform` 方法时设定参数 `ngram_range` 的值即可

模型建立

LinearRegression实现

- 风险函数

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right]$$

- 梯度下降

$$\frac{\partial J}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_j^{(i)}\right)$$

- 加入正则项
- 调用优化库 `import scipy.optimize as opt`，设定梯度下降函数，计算cost的函数即可

```
all_theta[i:], *unused = opt.fmin_cg(cost_func, fprime=grad_func, x0=theta_i,
                                     maxiter=100, disp=False, full_output=True)
```

- 进行预测

  采用 `One-hot` 编码，计算出每个lable的h_i(X)，取出最大值

```
# Add ones to the X data matrix
X = np.c_[np.ones(m), X]
# m个样本，每个样本都有5个预测输出（概率值）
result = sigmoid(np.dot(all_theta, X.T))  # (10,401) (5000,401)^T⇒(10,5000)
result = np.roll(result, -1, axis=0)    # 5000列同时在垂直方向向上滚动1个位置
# np.vstack:按垂直方向（行顺序）堆叠数组构成一个新的数组
result = np.vstack((np.zeros(m), result))   #  (1,5000) + (10,5000)  = (11,5000)
# 先上移在添一行全为0的数组，是因为算出0的概率在第一行，9在最后一行
# 而np.argmax，默认按列方向搜索最大值,返回的是索引所在的行数，这样能消除下标以及10代表0的作用
p = np.argmax(result, axis=0)   # p = (1,5000)
```

softmax实现

- softmax函数
  - 定义

    softmax函数将任意n维的实值向量转换为取值范围在(0,1)之间的n维实值向量，并且总和为1。
    例如：向量softmax([1.0, 2.0, 3.0]) ------> [0.09003057, 0.24472847, 0.66524096]
  - 性质：
    1. 因为softmax是单调递增函数，因此不改变原始数据的大小顺序。
    2. 将原始输入映射到(0,1)区间，并且总和为1，常用于表征概率。
    3. softmax(x) = softmax(x+c), 这个性质用于保证数值的稳定性。（避免输入的数值过大造成上溢）
  - 公式

$$\begin{aligned} p(y=c \mid \boldsymbol{x}) &= \operatorname{softmax}\left(\boldsymbol{w}_c^\top \boldsymbol{x}\right) \\ &= \frac{\exp\left(\boldsymbol{w}_c^\top \boldsymbol{x}\right)}{\sum_{c'=1}^{C}\exp\left(\boldsymbol{w}_{c'}^\top \boldsymbol{x}\right)}, \end{aligned}$$

- 代码

```
def softmax(x):
    # 减去最大值
    x -= np.max(x, axis = 1, keepdims = True)
    exp_x = np.exp(x)
    z = exp_x / np.sum(exp_x, axis = 1, keepdims = True)
    return z
```

- 标记值转化为独热编码

```
# 类别y转换为独热编码
y_one_hot = np.zeros((self.m, self.class_num))
for i in range(self.m):
    y_one_hot[i][y[i]] = 1
```

- 训练过程

  采用交叉熵损失函数, Softmax 回归模型的风险函数为

$$\begin{aligned}\mathcal{R}(\boldsymbol{W}) &= -\frac{1}{N}\sum_{n=1}^{N}\sum_{c=1}^{C}\boldsymbol{y}_c^{(n)}\log\hat{\boldsymbol{y}}_c^{(n)} \\ &= -\frac{1}{N}\sum_{n=1}^{N}\left(\boldsymbol{y}^{(n)}\right)^{\top}\log\hat{\boldsymbol{y}}^{(n)}\end{aligned}$$

  采用梯度下降法, Softmax 回归的训练过程为: 初始化 $\boldsymbol{W}_0 \leftarrow 0$, 然后通过下式进行迭代更新:

$$\boldsymbol{W}_{t+1} \leftarrow \boldsymbol{W}_t + \alpha\left(\frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}^{(n)}\left(\boldsymbol{y}^{(n)} - \hat{\boldsymbol{y}}_{\boldsymbol{W}_t}^{(n)}\right)^{\top}\right),$$

- 优化算法
  - 随机梯度下降法 (Stochastic Gradient Descent,SGD)

    随机采集一个样本， 计算这个样本损失函数的梯度并更新参数

```
if update_strategy == "stochastic":
    random_index = np.arange(len(X))
    np.random.shuffle(random_index)
    for index in list(random_index):

        #print("weight.shape",self.weight.shape)
        X_i = X[index]
        z = np.dot(X_i,self.weight).reshape(1,self.class_num)

        predict = softmax(z).flatten()
        loss -= np.log(predict[y[index]])
        grad = X_i.reshape(self.n,1).dot((y_one_hot[index]-predict).reshape(1,self.class_num))

        self.weight += grad
```

  - 批量梯度下降法 (Batch Gradient Descent, BGD)

    每次迭代时，计算所有样本损失函数的梯度并求和，再进行更新

```
if update_strategy == "batch":
    # X--(m,n), weight--(n,class_num)

    z = np.dot(X, self.weight)
    predict = softmax(z)

    # 构造同维度的梯度矩阵
    grad = np.zeros_like(self.weight)

    # 损失函数 & 梯度下降更新
    for i in range(self.m):
        loss -= np.log(predict[i][y[i]])
        grad += X[i].reshape(self.n,1).dot((y_one_hot[i] - predict[i]).reshape(1, self.class_num))

    # grad = np.dot(X.T, (y_one_hot - predict))
    self.weight += self.learning_rate * grad / self.m
```

- 进行预测并计算正确率

结果

为加快训练的速度，只用了1000个数据

模型	train_loss	val_loss
LinearRegression	0.8875	0.700
Softmax-bow	0.815	0.515
Softmax-ngram	0.9975	0.55