实验三：分类算法实验

一、实验名称

实验三：分类算法实验

二、实验目的

1. 掌握分类算法的算法思想：朴素贝叶斯算法，决策树算法，人工神经网络，支持向量机；
2. 编写朴素贝叶斯算法进行分类操作。

三、实验内容及步骤

实训内容：机器学习 --- 朴素贝叶斯分类器

实验步骤：

第1关 条件概率；

第2关 贝叶斯公式；

第3关 朴素贝叶斯分类算法流程；

第4关 拉普拉斯平滑；

第5关 新闻文本主题分类

四、开发环境

- OS: Ubuntu 20.04
- Language: Python
- IDE: PyCharm

五、算法原理及步骤

朴素贝叶斯

算法原理

朴素贝叶斯分类是一种十分简单的分类算法，叫它朴素贝叶斯分类是因为这种方法的思想真的很朴素，朴素贝叶斯的思想基础是这样的：对于给出的待分类项，求解在此项出现的条件下各个类别出现的概率，哪个最大，就认为此待分类项属于哪个类别。我们会选择条件概率最大的类别，这就是朴素贝叶斯的思想基础。

朴素贝叶斯分类的正式定义如下：

1、设$x = \{a_1, a_2, ..., a_m\}$为一个待分类项，而每个a为x的一个特征属性。

2、有类别集合$C = \{y_1, y_2, ..., y_n\}$。

3、计算$P(y_1|x), P(y_2|x), ..., P(y_n|x)$。

4、如果$P(y_k|x) = max\{P(y_1|x), P(y_2|x), ..., P(y_n|x)\}$，则$x \in y_k$。

那么现在的关键就是如何计算第3步中的各个条件概率。我们可以这么做：

1、找到一个已知分类的待分类项集合，这个集合叫做训练样本集。

2、统计得到在各类别下各个特征属性的条件概率估计。即
$P(a_1|y_1), P(a_2|y_1), ..., P(a_m|y_1); P(a_1|y_2), P(a_2|y_2), ..., P(a_m|y_2); ...; P(a_1|y_n), P(a_2|y_n), ..., P(a_m|y_n)$。

3、如果各个特征属性是条件独立的，则根据贝叶斯定理有如下推导：

$$P(y_i|x) = \frac{P(x|y_i)P(y_i)}{P(x)}$$

因为分母对于所有类别为常数，因为我们只要将分子最大化皆可。又因为各特征属性是条件独立的，所以有：

$$P(x|y_i)P(y_i) = P(a_1|y_i)P(a_2|y_i)...P(a_m|y_i)P(y_i) = P(y_i)\prod_{j=1}^{m} P(a_j|y_i)$$

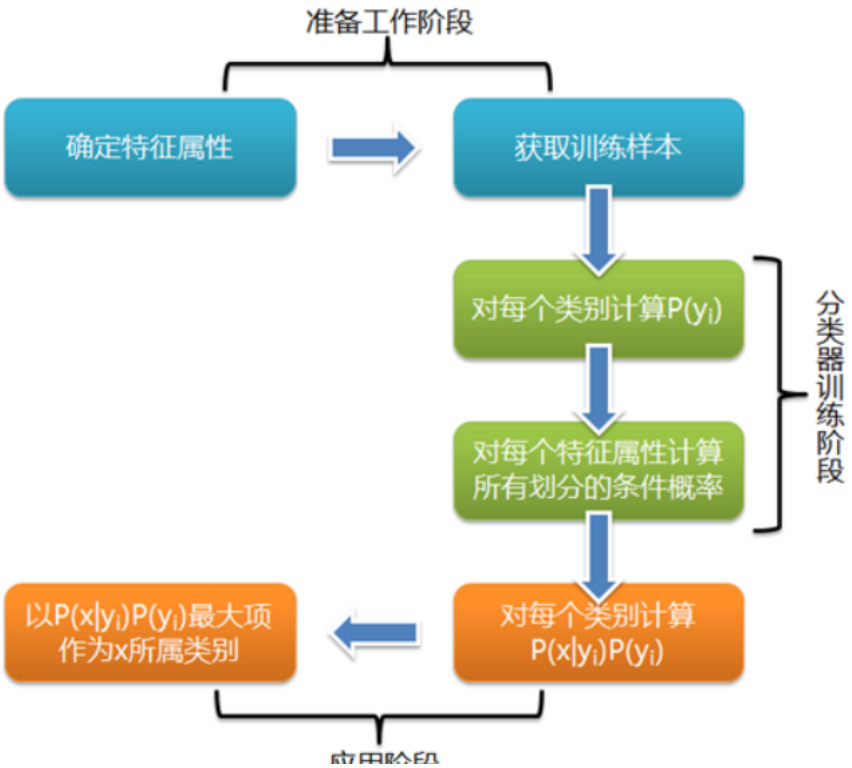
算法步骤

整个朴素贝叶斯分类分为三个阶段：

第一阶段——准备工作阶段，这个阶段的任务是为朴素贝叶斯分类做必要的准备，主要工作是根据具体情况确定特征属性，并对每个特征属性进行适当划分，然后由人工对一部分待分类项进行分类，形成训练样本集合。这一阶段的输入是所有待分类数据，输出是特征属性和训练样本。这一阶段是整个朴素贝叶斯分类中唯一需要人工完成的阶段，其质量对整个过程将有重要影响，分类器的质量很大程度上由特征属性、特征属性划分及训练样本质量决定。

第二阶段——分类器训练阶段，这个阶段的任务就是生成分类器，主要工作是计算每个类别在训练样本中的出现频率及每个特征属性划分对每个类别的条件概率估计，并将结果记录。其输入是特征属性和训练样本，输出是分类器。这一阶段是机械性阶段，根据前面讨论的公式可以由程序自动计算完成。

第三阶段——应用阶段，这个阶段的任务是使用分类器对待分类项进行分类，其输入是分类器和待分类项，输出是待分类项与类别的映射关系。这一阶段也是机械性阶段，由程序完成。



决策树

算法原理

决策树（decision tree）是一个树结构（可以是二叉树或非二叉树）。其每个非叶节点表示一个特征属性上的测试，每个分支代表这个特征属性在某个值域上的输出，而每个叶节点存放一个类别。使用决策树进行决策的过程就是从根节点开始，测试待分类项中相应的特征属性，并按照其值选择输出分支，直到到达叶子节点，将叶子节点存放的类别作为决策结果。

不同于贝叶斯算法，决策树的构造过程不依赖领域知识，它使用属性选择度量来选择将元组最好地划分成不同的类的属性。所谓决策树的构造就是进行属性选择度量确定各个特征属性之间的拓扑结构。

构造决策树的关键步骤是分裂属性。所谓分裂属性就是在某个节点处按照某一特征属性的不同划分构造不同的分支，其目标是让各个分裂子集尽可能地“纯”。尽可能“纯”就是尽量让一个分裂子集中待分类项属于同一类别。

分裂属性分为三种不同的情况：
1、属性是离散值且不要求生成二叉决策树。此时用属性的每一个划分作为一个分支。
2、属性是离散值且要求生成二叉决策树。此时使用属性划分的一个子集进行测试，按照“属于此子集”和“不属于此子集”分成两个分支。
3、属性是连续值。此时确定一个值作为分裂点split_point，按照>split_point 和<=split_point 生成两个分支。

构造决策树的关键性内容是进行属性选择度量，属性选择度量是一种选择分裂准则，是将给定的类标记的训练集合的数据划分D“最好”地分成个体类的启发式方法，它决定了拓扑结构及分裂点split_point 的选择。

算法步骤

生成决策树的基本流程：

输入： 训练集 $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_m, y_m)\}$;
　　属性集 $A = \{a_1, a_2, \ldots, a_d\}$.
过程： 函数 TreeGenerate(D, A)
1: 生成结点 node;
2: **if** D 中样本全属于同一类别 C **then**
3:　将 node 标记为 C 类叶结点; **return**
4: **end if**
5: **if** $A = \varnothing$ **OR** D 中样本在 A 上取值相同 **then**
6:　将 node 标记为叶结点, 其类别标记为 D 中样本数最多的类; **return**
7: **end if**
8: 从 A 中选择最优划分属性 a_*;
9: **for** a_* 的每一个值 a_*^v **do**
10:　为 node 生成一个分支; 令 D_v 表示 D 中在 a_* 上取值为 a_*^v 的样本子集;
11:　**if** D_v 为空 **then**
12:　　将分支结点标记为叶结点, 其类别标记为 D 中样本最多的类; **return**
13:　**else**
14:　　以 TreeGenerate(D_v, $A \setminus \{a_*\}$)为分支结点
15:　**end if**
16: **end for**
输出： 以 node 为根结点的一棵决策树

决策树的生成是一个递归过程，有三种情形会导致递归返回：
（1）当前结点包含的样本全属于同一类别，无需划分；
（2）当前属性为空或者是所有样本在所有属性上取值相同，无需划分，在这种情况下，我们把当前结点标记为叶结点，并将其类别设定为该结点所含样本最多的类别；
（3）当前结点包含的样本集为空，不能划分，同样把当前结点标记为叶结点。

人工神经网络

算法原理

利用输出后的误差来估计输出层前一层的误差，再用这层误差来估计更前一层误差，如此获取所有各层误差估计。这里的误差估计可以理解为某种偏导数，我们就是根据这种偏导数来调整各层的连接权值，再用调整后的连接权值重新计算输出误差。直到输出的误差达到要求或者迭代次数溢出设定值。

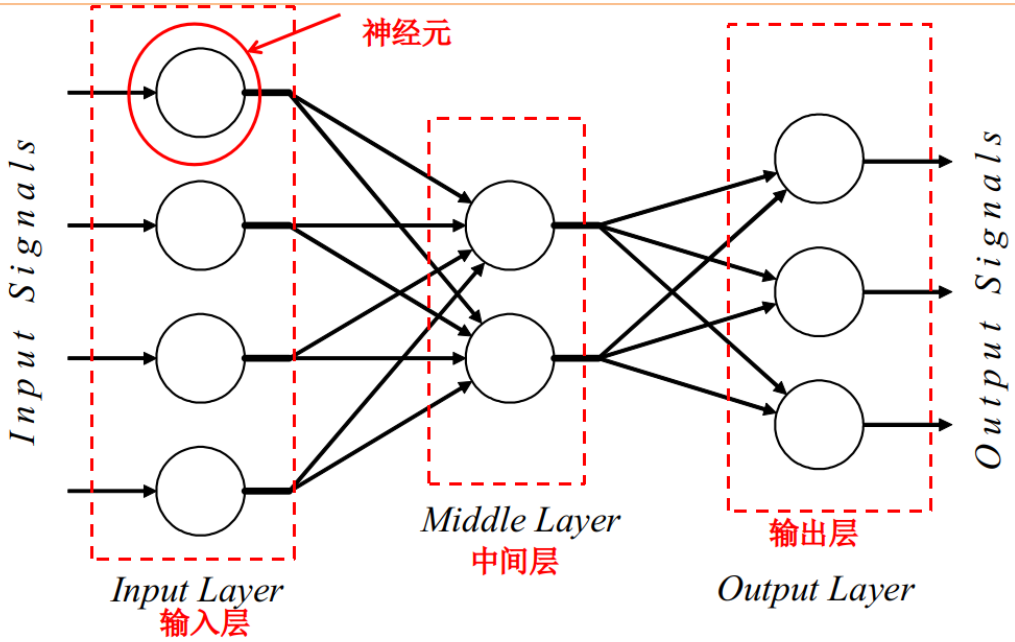
说来说去，“误差”这个词说的很多嘛，说明这个算法是不是跟误差有很大的关系？

没错，BP的传播对象就是“误差”，传播目的就是得到所有层的估计误差。

它的学习规则是：使用最速下降法，通过反向传播（就是一层一层往前传）不断调整网络的权值和阈值，最后使全局误差系数最小。

它的学习本质就是：对各连接权值的动态调整。

- 下面是一种典型的神经网络：



算法步骤

第一步：初始化

设定初始的权值w1,w2,...,wn和阈值θ为如下的一致分布中的随机数

$$\left(-\frac{2.4}{F_i}, +\frac{2.4}{F_i}\right)$$

Fi是输入神经元数量总和

第二步：计算激活函数值

根据输入x1§, x2§,..., xn§ 和权值w1,w2,...,wn计算输出y1§, y2§,..., yn§

（a）计算隐藏层神经元的输出

$$y_j(p) = sigmoid\left[\sum_{i=1}^{n} x_i(p) \cdot w_{ij}(p) - \theta_j\right]$$

n是第j个隐藏层神经元的输入数量，sigmoid是激活函数

（b）计算输出层神经元的输出

$$y_k(p) = sigmoid\left[\sum_{j=1}^{m} x_{jk}(p) \cdot w_{jk}(p) - \theta_k\right]$$

m是第k个输出神经元的输入数量

第三步：权值更新(从后往前)

（a）计算输出层的错误梯度

$$e_k(p) = y_{d,k}(p) - y_k(p)$$

计算权值纠正值

$$\Delta w_{jk}(p) = \alpha \cdot y_j(p) \cdot \delta_k(p)$$

更新输出层的权值

（b）计算隐藏层的错误梯度

$$\delta_j(p) = y_j(p) \cdot [1 - y_j(p)] \cdot \sum_{k=1}^{l} \delta_k(p) w_{jk}(p)$$

计算权值纠正值

$$\Delta w_{ij}(p) = \alpha \cdot x_i(p) \cdot \delta_j(p)$$

更新隐藏层的权值

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

第四步：迭代循环

增加p值，不断重复步骤二和步骤三直到输出的误差达到停止条件。

支持向量机

算法原理

支持向量机（Support Vector Machine ,SVM）的主要思想是：建立一个最优决策超平面，使得该平面两侧距离该平面最近的两类样本之间的距离最大化，从而对分类问题提供良好的泛化能力。对于一个多维的样本集，系统随机产生一个超平面并不断移动，对样本进行分类，直到训练样本中属于不同类别的样本点正好位于该超平面的两侧，满足该条件的超平面可能有很多个，SVM正式在保证分类精度的同时，寻找到这样一个超平面，使得超平面两侧的空白区域最大化，从而实现对线性可分样本的最优分类。

支持向量机中的支持向量（Support Vector）是指训练样本集中的某些训练点，这些点最靠近分类决策面，是最难分类的数据点。SVM 中最优分类标准就是这些点距离分类超平面的距离达到最大值；“机”（Machine）是机器学习领域对一些算法的统称，常把算法看做一个机器，或者学习函数。SVM 是一种有监督的学习方法，主要针对小样本数据进行学习、分类和预测，类似的根据样本进行学习的方法还有决策树归纳算法等。

算法步骤

首先是待优化问题：

$$max_{\alpha}W(\alpha)=\sum_{i=1}^{m}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{m}y^{(i)}y^{(j)}\alpha_i\alpha_j<x^{(i)}x^{(j)}>$$

$$st.0\leq\alpha_i\leq C,i=1,2,.....,m$$

$$\sum_{i=1}^{m}\alpha_i y^{(i)}=0$$

然后是SMO算法的大体流程：

Repeat till convergence

{

1、从α1,α2,.....,αmα1,α2,.....,αm中选取出一对αjαj和αkαk进行优化；

2、将除了上一步中挑选的αjαj和αkαk以外的ααα看做常数，对选出的两个ααα进行更新使得W(α)W(α)取得更大的值；

}

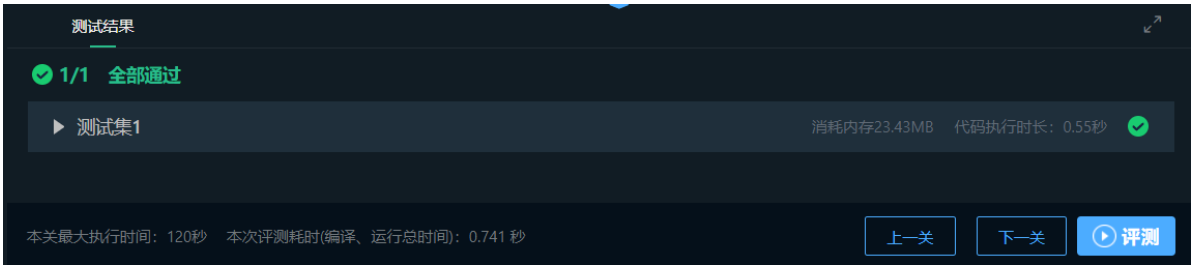
六、实验代码

第3关 朴素贝叶斯分类算法流程

```
def fit(self, feature, label):
        '''
        对模型进行训练，需要将各种概率分别保存在self.label_prob和self.condition_prob中
        :param feature: 训练数据集所有特征组成的ndarray
        :param label:训练数据集中所有标签组成的ndarray
        :return: 无返回
        '''


        #********** Begin *********#
        row_num = len(feature)
        col_num = len(feature[0])
        for c in label:
            if c in self.label_prob:
                self.label_prob[c] += 1
```

```
            else:
                self.label_prob[c] = 1

        for key in self.label_prob.keys():
            # 计算每种类别在数据集中出现的概率
            self.label_prob[key] /= row_num
            # 构建self.condition_prob中的key
            self.condition_prob[key] = {}
            for i in range(col_num):
                self.condition_prob[key][i] = {}
                for k in np.unique(feature[:, i], axis=0):
                    self.condition_prob[key][i][k] = 0

        for i in range(len(feature)):
            for j in range(len(feature[i])):
                if feature[i][j] in self.condition_prob[label[i]]:
                    self.condition_prob[label[i]][j][feature[i][j]] += 1
                else:
                    self.condition_prob[label[i]][j][feature[i][j]] = 1

        for label_key in self.condition_prob.keys():
            for k in self.condition_prob[label_key].keys():
                total = 0
                for v in self.condition_prob[label_key][k].values():
                    total += v
                for kk in self.condition_prob[label_key][k].keys():
                    #计算每种类别确定的条件下各个特征出现的概率
                    self.condition_prob[label_key][k][kk] /= total
        #********** End **********#


    def predict(self, feature):
        '''
        对数据进行预测，返回预测结果
        :param feature:测试数据集所有特征组成的ndarray
        :return:
        '''
        # ********** Begin **********#
        result = []
        #对每条测试数据都进行预测
        for i, f in enumerate(feature):
            #可能的类别的概率
            prob = np.zeros(len(self.label_prob.keys()))
            ii = 0
            for label, label_prob in self.label_prob.items():
                #计算概率
                prob[ii] = label_prob
                for j in range(len(feature[0])):
                    prob[ii] *= self.condition_prob[label][j][f[j]]
                ii += 1
            #取概率最大的类别作为结果
            result.append(list(self.label_prob.keys())[np.argmax(prob)])
        return np.array(result)
        #********** End **********#
```

运行结果

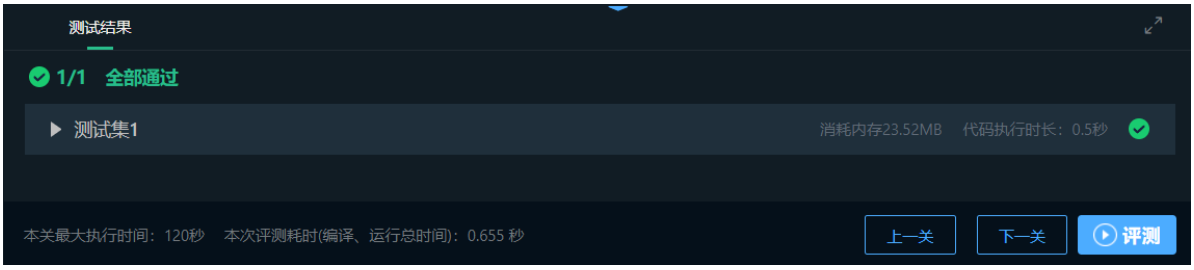


第4关 拉普拉斯平滑

```
def fit(self, feature, label):
        '''
        对模型进行训练，需要将各种概率分别保存在self.label_prob和self.condition_prob中
        :param feature: 训练数据集所有特征组成的ndarray
        :param label:训练数据集中所有标签组成的ndarray
        :return: 无返回
        '''

        #********* Begin *********#
        row_num = len(feature)
        col_num = len(feature[0])
        unique_label_count = len(set(label))

        for c in label:
            if c in self.label_prob:
                self.label_prob[c] += 1
            else:
                self.label_prob[c] = 1

        for key in self.label_prob.keys():
            # 计算每种类别在数据集中出现的概率，拉普拉斯平滑
            self.label_prob[key] += 1
            self.label_prob[key] /= (unique_label_count+row_num)

            # 构建self.condition_prob中的key
            self.condition_prob[key] = {}
            for i in range(col_num):
                self.condition_prob[key][i] = {}
                for k in np.unique(feature[:, i], axis=0):
                    self.condition_prob[key][i][k] = 1


        for i in range(len(feature)):
            for j in range(len(feature[i])):
                if feature[i][j] in self.condition_prob[label[i]]:
                    self.condition_prob[label[i]][j][feature[i][j]] += 1

        for label_key in self.condition_prob.keys():
            for k in self.condition_prob[label_key].keys():
                #拉普拉斯平滑
                total = len(self.condition_prob[label_key].keys())
                for v in self.condition_prob[label_key][k].values():
                    total += v
```

```
                for kk in self.condition_prob[label_key][k].keys():
                    # 计算每种类别确定的条件下各个特征出现的概率
                    self.condition_prob[label_key][k][kk] /= total
        #********** End **********#
```

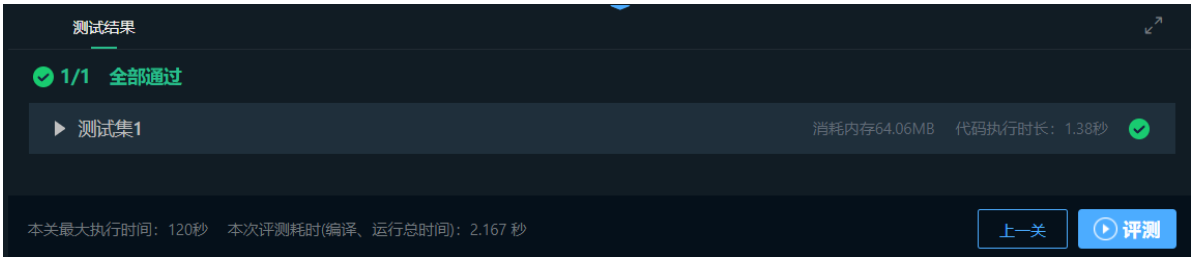
运行结果



第5关 新闻文本主题分类

```
def news_predict(train_sample, train_label, test_sample):
    '''
    训练模型并进行预测，返回预测结果
    :param train_sample:原始训练集中的新闻文本，类型为ndarray
    :param train_label:训练集中新闻文本对应的主题标签，类型为ndarray
    :test_sample:原始测试集中的新闻文本，类型为ndarray
    '''

    # ********** Begin **********#
    vec = CountVectorizer()
    train_sample = vec.fit_transform(train_sample)
    test_sample = vec.transform(test_sample)

    tfidf = TfidfTransformer()

    train_sample = tfidf.fit_transform(train_sample)
    test_sample = tfidf.transform(test_sample)

    mnb = MultinomialNB(alpha=0.01)  # 使用默认配置初始化朴素贝叶斯
    mnb.fit(train_sample, train_label)  # 利用训练数据对模型参数进行估计
    predict = mnb.predict(test_sample)  # 对参数进行预测
    return predict
    # ********** End **********#
```

运行结果



七、实验结果与分析

源码地址

https://gitee.com/gunshi3/artificial-intelligence/tree/master/lab3_code

在对算法的测试过程中，我记录了算法的运行时间，训练准确率，测试准确率，下面我将对他们一一列举：

算法	训练准确率	测试准确率	时间开销	备注
NaiveBayes	88.296%	68.783%	36.251ms	朴素贝叶斯
DecisionTree ID3	100%	70.3703%	19.308ms	ID3决策树
DecisionTree C4.5	100%	70.3703%	14.732ms	C4.5决策树
DecisionTree CART	100%	64.28%	14.773ms	CART决策树
NeuralNetwork	96.3%	72.75%	373.4158s	BP人工神经网络
SVM 线性核	86.89%	72.75%	17ms	SVM 线性核
SVM 高斯核	97.18%	87.26%	20ms.	SVM 高斯核

准确率比较

观察上表可知，准确率最高的算法是SVM高斯核，BP人工神经网络次之；DecisionTree的三个算法中ID3决策树和 C4.5决策树准确率一致，都达到 70.3703%；朴素贝叶斯测试准确率为 68.783%，最后是CART决策树，测试准确率为64.28%。后来我使用sklearn库实现神经网络，对比我手工实现的BP准确率提高到了89.655%，时间开销缩短到了1min之内。

时间开销比较

观察上表可知，时间开销最低的是决策树算法和SVM，算法的时间都小于20ms；其次是朴素贝叶斯算法，时间开销是36.251ms；运行时间最长的是BP人工神经网络，我设置了迭代次数为5000, 2个中间隐藏层，学习率设置为0.25，实际运行时间达到了8min左右。后来我使用sklearn库实现神经网络，对比我手工实现的BP准确率提高到了0.89655，时间开销缩短到了1min之内。

时间复杂度比较

(1) 时间复杂度最高的是 BP人工神经网络，以一个简单的三层BP神经网络为例，假设每层神经元数量分别为n1，n2，n3。拿一个样本（n1 * 1）进行前馈计算，那么就要进行两次矩阵运算，两次矩阵乘法（实际上是向量和矩阵相乘）分别要进行n1 * n2 和 n2 * n3次计算，由于输入层和最终输出层结点数量(n1和n3)是确定的，所以可以视为常量，中间的隐藏层n2可以由自己设定。对一个样本的前馈计算时间复杂度应该是O(n1 * n2 + n2 * n3) = O(n2)。反向传播时时间复杂度和前馈计算相同，假设总共有m个训练样本，每个样本只训练一次，那么训练一个神经网络的时间复杂度应该是O(m*n2).后来我去网上找了下，正确的时间复杂度为

$$\mathbf{Time} \sim O\left(\sum_{l=1}^{D} M_l^2 \cdot K_l^2 \cdot C_{l-1} \cdot C_l\right)$$

(2) 朴素贝叶斯由于要计算每一个属性的概率，其时间复杂度为O(C*n2)

(3) 决策树算法构建一颗分类树，分类树的节点数量和数据特征有关，假设有m个特征，那么决策树算法的时间复杂度为$O(n*2^m)$

(4) SVM算法，原始问题的时间复杂度为$O(d^3+n*d^2)$，对偶的时间解法时间复杂度为$O(n_sv^3+n*n_sv^2)$

空间复杂度比较

(1) 训练一个神经网络的空间复杂度为

$$\mathbf{Space} \sim O\left(\sum_{l=1}^{D} K_l^2 \cdot C_{l-1} \cdot C_l\right)$$

(2) 朴素贝叶斯不用过多的辅助空间，只需一个长度为m*n的数组来记录条件概率即可，S(n) = O(m*n)

(3) 决策树算法构建一颗分类树，分类树的节点数量和数据特征有关，假设有m个特征，那么决策树算法的空间复杂度为O(n*2^m)

(4) SVM算法的空间复杂度：假定分类数量为N_S，也是支持向量的个数，输入向量维度为d_L，相当于在维度为d_L的高维空间上，存储Ns个高维超平面，用于对每个d_L维度的样本点进行分类。线性SVM需要存储的参数就是这N_S个高维超平面的参数，共$d_L N_S$个数字需要存储。因此空间复杂度为S(n) = O($d_L N_S$)

八、思考题

如何在参数学习或者其他方面提高算法的分类性能?

以人工神经网络算法为例，调整学习相关参数，可以观察到如下结果。

- 设置学习时间为300的情况，可以看到均方根误差为0.0096，训练时间2.41秒。
- 设置学习时间为1300的情况，可以看到均方根误差为0.0044，训练时间10.32秒。

PS:均方根误差是观测值与真值偏差的平方与观测次数n比值的平方根，在实际测量中，观测次数n总是有限的，真值只能用最可信赖（最佳）值来代替。对一组测量中的特大或特小误差反映非常敏感，所以，标准误差能够很好地反映出测量的精密度。因此这里我们使用均方差观察学习的准确性。

通过上述对比可以发现，**训练时间越长，训练结果越准确，误差越低。**这是因为随着训练时间的加长，迭代次数会越来越多，因此更容易接近实际准确情况。

下面，我们研究学习率对于均方根误差的影响：

- 设置学习率为0.3的情况，训练时间设置为500，可以看到均方根误差为0.0073，实际训练时间为3.99秒。
- 设置学习率为0.6的情况，发现均方根误差降低到了0.0051，同时实际训练时间略有提升，达到了4.26秒，但是，此时我们人为所设定的学习时间并没有变化，此时我认为这里的时间提升，大致可以归因为一个可以被允许的误差。

而一味提升学习率就一定能够提升算法准确率吗？答案是否定的。因为学习率，又被称为“步长”。“步长”越大，每一次修改分类算法的关键参数的时候，这些参数的摆动幅度也就越大，因此，误差有可能随着学习率的提升而提升，而更为糟糕的是，学习率（步长）若过大，会使得算法在最优解附近不断来回摆动，因为步长太大的原因，算法参数无法收敛到全局最优，能否实际提升学习率是不确定的，全靠运气。

综上所述，我认为提高算法性能主要有两个方面：

- 如果只需要提高运行速度，可以降低迭代次数，但这样会导致模型的误差增大；
- 如果需要提高模型的精确度，就需要适当更改学习率，且一定要增加迭代次数，但是如果是通过增加迭代次数的方法提升精确度，那么相应的算法时间开销就会不可避免地增大。

因此，从参数的角度来考虑分类器的学习建模，我们需要权衡精确度和效率之间的关系。

九、实验总结

实验收获

通过这次实验，我了解到了四种分类器算法的基本思想，更是从实验中学习到了基本的调参方法，通过本次实验，我的收获比较多。
本次实验侧重于各个算法的实际应用，对于内在原理，要在日后的学习中逐渐理解掌握。

难点重点讨论

关于提高算法的分类性能，对于参数的调整，支持向量机可以尝试不同的核，或者减少其软间隔系数，以使得分类效果更好，但是会降低其泛化性能；决策树则可以尝试不同的剪枝策略来提高其泛化能力；朴素贝叶斯则可以尝试进行不同的平滑策略，除了拉普拉斯平滑还可以选择lidstone平滑；而人工神经网络，则可以尝试增加训练轮数或者增大网络的深度来得到更好的分类结果，但是带来的代价则是时间开销的增大，此外还可以调整学习率来得到更好的训练结果。

十、参考地址

- 带你理解朴素贝叶斯分类算法
- 决策树(Decision Tree)：通俗易懂之介绍
- 【机器学习】支持向量机 SVM（非常详细）