实验七：特征选择算法

姓名：孙铭

学号：1711377

专业：计算机科学与技术

日期：2020年6月20日

目录

摘要

本次作业主题是特征选择，实验涉及到的基本、中级、高级三类要求已全部实现。在此声明本次作业代码、实验结果及实验报告均由个人独立完成，本次实验实现的具体功能参下。

> **实验基本要求：**
>
> 1. 实现了ReliefF算法。
> 2. 基于ReliefF算法，选择排序前1/6，2/6，3/6，4/6，5/6，1的特征，比较在KNN，NB，SVM，Random Forests不同分类器的分类精度，AUC值，并用图像展示。
>
> **实验中级要求：**
>
> 3. 实现最大相关最小冗余MRMR算法。
> 4. 基于MRMR算法，选择排序前1/6，2/6，3/6，4/6，5/6，1的特征，比较在KNN，NB，SVM，Random Forests不同分类器的分类精度，AUC值，并用图像展示。
>
> **实验提高要求：**
>
> 5. 给数据集加入50，100，150，200个噪声特征。
> 6. 同样对于不同的噪声特征集，选择排序前1/6，2/6，3/6，4/6，5/6，1的特征，比较在KNN，NB，SVM，Random Forests不同分类器的分类精度，AUC值，并用图像展示。
> 7. 比较relief-F和MRMR的抗噪能力。

以上是本次实验摘要部分内容，接下来按阶段展开讲述。

环境配置

本次作业使用语言版本为`Python 3.6.5 64bit`，处理器为`Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz 1.80GHz`，项目工程文件如下。

```
| data                  # 数据集保存目录
| pic                   # 实验结果图保存文件
| result                # 实验准确率、auc结果保存目录
files.py                # 文件读写脚本
load_data.py            # 数据预处理脚本
ReliefF.py              # 本次作业算法代码主体
```

实验过程中用到的包如下。

```python
import pickle
import codecs
import json
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn import svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import label_binarize
from sklearn import metrics
import matplotlib.pyplot as plt
```

以上是本次实验环境配置部分。

数据处理

本次实验选用的数据集为`urban.mat`，`.mat`数据格式是Matlab的数据存储的标准格式。在python中可以使用`scipy.io`中的函数`loadmat()`读取mat文件，使用`savemat()`函数保存mat文件。从mat文件读取到的数据格式为字典，同样，在将数据保存成mat文件时，传入数据的格式也需为字典。

```python
import scripy.io as scio

scio.loadmat(filename)
scio.savemat(filename, data)
```

从mat文件中读取到数据并筛选出样本特征及标签后，输出样本的shape，样本总量为675，包含147个特征列和1个标签列。其中，标签分类数值为1-9，将样本共划分为9类。参考作业要求中的数据说明："利用高分辨率航空影像对城市土地类型进行分类，土地类型有：树木、草地、土壤、混凝土、沥青、建筑物、汽车、水池、阴影共九类。"

此外，九类数据中，每类数据的标签分类值及样本数量对应关系如下。

```
{'1': 116, '2': 61, '3': 106,
 '4': 59, '5': 122, '6': 112,
 '7': 29, '8': 36, '9': 34}
```

为了方便后续处理，我将样本特征和标签整合在一个二维数组中并保存成`urban.pkl`文件，便于后续处理时节约代码运行时间。

以上是本次实验数据处理部分。

实验初级要求

1. Relief算法

Relief算法是一种特征权重算法，根据各个特征和类别的相关性赋予特征不同的权重，权重小于某个阈值的特征将被移除。Relief算法中特征和类别的相关性是基于特征对近距离样本的区分能力。算法从训练集D中随机选择一个样本R，然后从和R同类的样本中寻找最近邻样本H(Near Hit)，从和R不同类的样本中寻找最近邻样本M(Near Miss)。然后根据以下规则更新每个特征的权重：

如果R和H在某个特征上的距离小于R和M在该特征上的距离，则说明该特征区分同类和不同类的最近邻是有益的，则增加该特征的权重；反之，如果R和H在某个特征上的距离大于R和M在该特征上的距离，则说明该特征对区分同类和不同类的最近邻起负面作用，则降低该特征的权重。上述过程循环m次，最后得到各特征的平均权重。特征的权重越大，表示该特征的分类能力越强，反之，表示该特征的分类能力越弱。Relief算法的运行时间随着样本抽样次数m和原始特征个数N的增加线性增加，因而运行效率非常高。

假设一个样例X有p个特征，S为样本量为n的训练样本，F即$f_1, f_2, \ldots, f_p$为特征集，一个样例X由p维向量$(x_1, x_2, \ldots, x_p)$构成，其中，x_j为X的第j个特征的值。

Relief算法可以解决名义变量和数值变量，两个样例X和Y的特征的值的差可以由下面函数来定义：

当x_k和y_k为名义变量时：

$$diff(x_k, y_k) = \begin{cases} 1 & \text{如果} x_k \text{和} y_k \text{不相同} \\ 0 & \text{如果} x_k \text{和} y_k \text{相同} \end{cases}$$

当x_k和y_k为数值变量时：

$$diff(x_k, y_k) = \frac{(x_k - y_k)}{v_k}$$

其中，v_k为归一化单位，把diff值归一到$[0, 1]$区间，可以在之前先把数值变量进行归一化。

Relief算法伪代码如下。

> **输入：样本集S，抽样次数m，特征权重阈值τ**
>
> **输出：选择后的特征集**
>
> 把S分成$S^+ = \{\text{正例}\}$和$S^- = \{\text{负例}\}$，权重$W = (0, 0, \ldots, 0)$
>
> For i = 1 to m
>
> 随机选择一个样例$X \in S$
>
> 随机选择一个距离X最近的一个正例$Z^+ \in S^+$
>
> 随机选择一个距离X最近的一个负例$Z^- \in S^-$
>
> If X是一个正例
>
> then $H = Z^+, M = Z^-$
>
> else $H = Z^-, M = Z^+$
>
> For i = 1 to p
>
> $W_i = W_i - diff(x_i, H_i)^2 + diff(x_i, M_i)^2$
>
> $relevance = \frac{1}{m}W$
>
> For i = 1 to p
>
> If $relevance_i \geq \tau$

> then f_i是一个相关的特征
>
> else f_i是一个无关的特征

---

2. ReliefF算法

尽管Relief算法相对简单，且运行效率高，结果也比较令人满意，应用较为广泛，但其局限性在于只能处理两类别数据。而Relief算法可以处理多类别问题，该算法用于处理目标属性为连续值的回归问题。Relief算法在处理多类别问题时，每次从训练样本集中随机取出一个样本R，然后从和R同类的样本集中找出R的k个近邻样本H(Near Hits)，从每个R的不同类的样本集中均找出k个近邻样本M(Near Misses)，然后更新每个特征权重，如下式所示。

$$W(A) = W(A) - \sum_{j=1}^{k} diff(A, R, H_j)/(mk) +$$

$$\sum_{C \notin class(R)} [\frac{p(C)}{1 - p(class(R))} \sum_{j=1}^{k} diff(A, R, M_j(C))]/(mk)$$

上式中，$diff(A, R_1, R_2)$表示在样本R_1和R_2在特征A上的差，$M_j(C)$表示类$C \notin class(R)$中第j个最近邻样本，如下式：

$$diff(A, R_1, R_2) = \begin{cases} \frac{|R_1[A] - R_2[A]|}{max(A) - min(A)} & If\ A\ is\ continuous \\ 0 & If\ A\ is\ discrete\ and\ R_1[A] = R_2[A] \\ 1 & If\ A\ is\ discrete\ and\ R_1[A] \neq R_2[A] \end{cases}$$

算法伪代码如下。

> **输入：训练集D，抽样次数m，特征权重阈值δ，最近邻样本个数k**
>
> **输出：各个特征的特征权重T**
>
> 置所有特征权重为0，T为空集
>
> For i = 1 to m
>
> 从训练集D中随机选择一个样本R
>
> 从R的同类样本集中找到R的k个最近邻$H_j(j = 1, 2, \ldots, k)$，从每一个不同类样本集中找到k个最近邻$M_j(C)$
>
> For A = 1 to N(all features)
>
> $W(A) = W(A) - \sum_{j=1}^{k} diff(A, R, H_j)/(mk) +$
>
> $\sum_{C \notin class(R)} [\frac{p(C)}{1 - p(class(R))} \sum_{j=1}^{k} diff(A, R, M_j(C))]/(mk)$
>
> End

于是，基于上述分析，实现ReliefF算法如下。

```
# data->训练集 sampling_times->抽样次数 features_num->需求特征数 k->最近邻样本个数
def ReliefF(data, sampling_times, features_num, k):
    data_X, data_Y = normalize(data[:, :-1]), data[:, -1]
```

```
    data_classify = [data_X[np.argwhere(data_Y == i).reshape(-1)] for i in range(1, 10)]
    features_count = data_X.shape[1]  # 特征数
    weight = np.zeros(features_count)  # 初始化权重
    max_min_A = np.max(data_X, axis=0) - np.min(data_X, axis=0) # 样本特征的最大值-最小值

    for i in range(sampling_times):
        # 从训练集中随机抽取样本R
        label = np.random.randint(1, 10, size=1)[0]     # 样本R的标签
        sampleR_index = np.random.randint(0, len(data_classify[label - 1]), size=1)[0]
        sampleR = data_classify[label - 1][sampleR_index]   # 随机抽取的样本R

        # 从同类及不同类样本中找到R的k个最近邻
        near = {}
        for j in range(1, 10):
            sample_j = data_classify[j - 1]
            dist = euclidean_dist(sampleR, sample_j)
            near[str(j)] = sample_j[np.argsort(dist)][:k]

        # 更新权重
        for A in range(features_count):
            near_hits_diff, near_misses_diff = 0, 0
            for h in range(len(data_classify)):
                if h != label - 1:
                    near_misses_diff += np.sum(diff(A, sampleR, near[str(h + 1)], max_min_A)) / 9
                elif h == label - 1:
                    near_hits_diff += np.sum(diff(A, sampleR, near[str(h + 1)], max_min_A))
            weight[A] = weight[A] - (near_hits_diff - near_misses_diff) / (sampling_times * k)

    return np.argsort(-weight)[:features_num]
```

代码返回值为已选择的特征的索引。

3. KNN、NB、SVM、RF分类器实现

对于初级要求中给出的四种分类器KNN、NB、SVM、RF，我选择利用sklearn中的包进行实现。

对于初级要求中要求计算的分解结果准确率及AUC的值，我分别利用`model.score()`和`sklearn.metrics.auc()`进行计算。需要指出的一点是，在利用`sklearn.metrics.auc()`计算auc值前，由于sklearn给出的计算auc的方式只支持二分类问题，因此需要先将特征值转化成one hot向量，之后再进行标签二值化进行auc的计算。

考虑到代码的复用，这里我将四种分类器封装成类进行实现。

KNN分类器实现代码如下。

```
def knn(self):
    sknn = KNeighborsClassifier(n_neighbors=3)
    predict_prob_y = sknn.fit(self.train_X,
self.train_Y).predict_proba(self.test_X) # 训练并返回分类结果概率
    score = sknn.score(self.test_X, self.test_Y)
    fpr, tpr, thresholds = metrics.roc_curve(self.y_one_hot.ravel(),
predict_prob_y.ravel())
    auc = metrics.auc(fpr, tpr)
    return self._print('KNN', score, auc)
```

朴素贝叶斯分类器实现如下。

```
def NaiveBayes(self):
    snb = GaussianNB()
    predict_prob_y = snb.fit(self.train_X,
self.train_Y).predict_proba(self.test_X)
    score = snb.score(self.test_X, self.test_Y)
    fpr, tpr, thresholds = metrics.roc_curve(self.y_one_hot.ravel(),
predict_prob_y.ravel())
    auc = metrics.auc(fpr, tpr)
    return self._print('NB', score, auc)
```

支持向量机分类器实现如下。

```
def SVM(self):
    clf = svm.SVC(gamma='scale', probability=True)
    predict_prob_y = clf.fit(self.train_X,
self.train_Y).predict_proba(self.test_X)
    score = clf.score(self.test_X, self.test_Y)
    fpr, tpr, thresholds = metrics.roc_curve(self.y_one_hot.ravel(),
predict_prob_y.ravel())
    auc = metrics.auc(fpr, tpr)
    return self._print('SVM', score, auc)
```

随机森林分类器实现如下。

```
def RandomForest(self):
    srf = RandomForestClassifier(n_estimators=25)
    predict_prob_y = srf.fit(self.train_X,
self.train_Y).predict_proba(self.test_X)
    score = srf.score(self.test_X, self.test_Y)
    fpr, tpr, thresholds = metrics.roc_curve(self.y_one_hot.ravel(),
predict_prob_y.ravel())
    auc = metrics.auc(fpr, tpr)
    return self._print('RF', score, auc)
```

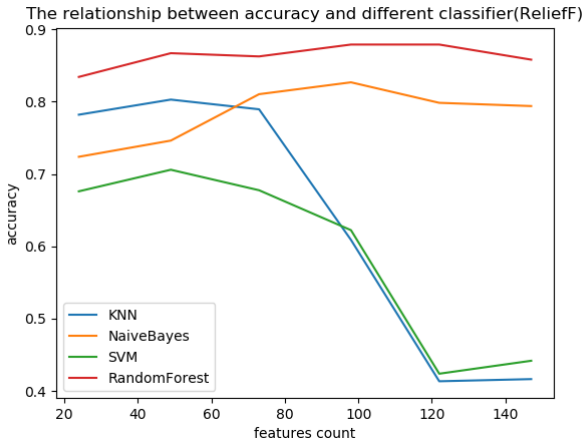
上述函数方法返回结果以字典格式保存，字典中三个键值分别为分类器种类、准确度、auc值。

4. 实验结果

结合上述内容，根据实验要求，接下来需要选择排序前1/6，2/6，3/6，4/6，5/6，1的特征，比较在KNN，NB，SVM，Random Forests不同分类器的分类精度，AUC值，并用图像展示。

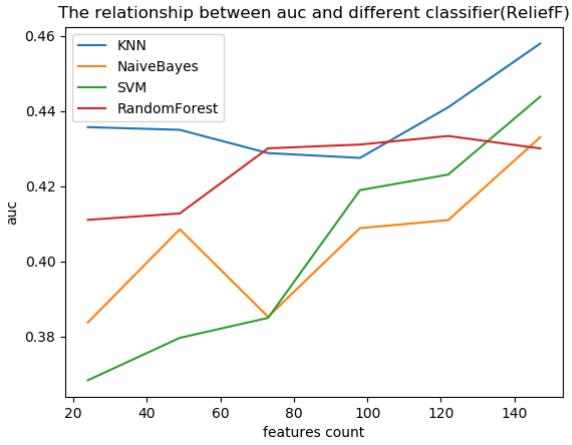
这里我采用十折交叉验证的方法进行实现，代码请见`run(data,fea_num,method,noise=None)`函数。这里不再赘述，直接给出实验结果图如下。

下图是ReliefF算法计算出的四种分类器分类结果的准确率示意图。



可以看出，随机森林分类算法的整体精度表现是最优的，最高接近90%，朴素贝叶斯分类器次之。而KNN分类器和SVM分类器随特征数量增加，分类精度越来越低。可以看出，随机森林和朴素贝叶斯算法受样本特征影响较小，分类结果比较稳定，而KNN和SVM受样本特征影响很大，当样本中无关或关联度较低特征增加时，分类器分类结果越来越差。

下图是ReliefF算法计算出的四种分类器分类结果的AUC示意图。



从上述结果图中可以看出，随特征数量增加，四种分类算法的AUC值整体递增，但随机森林和朴素贝叶斯的AUC值变化相对平稳，而KNN和支持向量机变动较大，这也能够从侧面印证支持向量机和KNN算法对数据集要求较高，对特征变化非常敏感的特性。

以上是本次实验初级要求部分。

实验中级要求

1. MRMR算法

彭汉川老师提出了特征选择中的mRMR（Max-Relevance and Min-Redundancy）算法，其原理是在原始特征集合中找到与最终输出结果相关性最大（Max-Relevance），但特征彼此之间相关性最小的一组特征（Min-Redundancy）。

算法基于对互信息的计算实现。所谓互信息，即一个随机变量中包含的关于另一个随机变量的信息量。给定两个随机变量x和y，它们的概率密度函数（对于连续变量）为$p(x)$、$p(y)$、$p(x,y)$，互信息计算公式如下。

$$I(x;y)=\int\int p(x,y)log\frac{p(x,y)}{p(x)(y)}dxdy$$

而在实际计算中，由于数据并非是连续变量，而是离散变量，因此对离散变量而言，互信息计算方式如下。

$$I(x;y)=\sum_{x\in X}\sum_{y\in Y}p(x,y)log\frac{p(x,y)}{p(x)(y)}$$

特征集S与类c的相关性由各个特征fi和类c之间的所有互信息值的平均值定义，公式如下。

$$D(S,c)=\frac{1}{|S|}\sum_{f_i\in S}I(f_i;c)$$

集合S中所有特征的冗余是特征fi和特征fj之间的所有互信息值的平均值。

$$R(S)=\frac{1}{|S|^2}\sum_{f_i,f_j\in S}I(f_i;f_j)$$

mRMR标准是上面给出的两种措施的组合

$$mRMR=\max_S[\frac{1}{|S|}\sum_{f_i\in S}I(f_i;c)-\frac{1}{|S|^2}\sum_{f_i,f_j\in S}I(f_i;f_j)]$$

而在实际应用中，采用增量搜索的方式可以得到近似最优解。公式如下。

$$\max_{x_j\in X-S_{m-1}}[I(x_j;c)-\frac{1}{m-1}\sum_{x_i\in S_{m-1}}I(x_j;x_i)]$$

即在已选择特征的基础上，在剩下的特征空间中找到能使上式最大的那个特征。相当于对剩下的每个特征进行计算然后排序。于是，代码实现如下。

```python
# data -> 训练集 || features_num -> 需求特征数
def MRMR(data, features_num):
    data_X, data_Y = normalize(data[:, :-1]), data[:, -1]
    fea_count = data_X.shape[1]    # 特征总量
    mrmr_fea = []  # 已选择特征集
    remain_fea = np.arange(fea_count)    # 剩余特征集

    # 计算初始最大互信息
    _info = np.array([metrics.mutual_info_score(data_X[:, i], data_Y) for i in range(fea_count)])
    _info_index = np.argsort(-_info)[0]
    mrmr_fea.append(_info_index)
    remain_fea = np.delete(remain_fea, _info_index)

    # 使用增量搜索实现mRMR
    for i in range(features_num - 1):
        res = []
        for j in range(len(remain_fea)):
            x_j = data_X[:, remain_fea[j]]
            info_remain = metrics.mutual_info_score(x_j, data_Y)
```

```
            info_mrmr = sum([metrics.mutual_info_score(x_j, data_X[:, t])
for t in mrmr_fea])
            # 计算一轮迭代后的mrmr
            mrmr_res = info_remain - (1 / (features_num - 1)) * info_mrmr
            res.append(mrmr_res)
        _index = np.argmax(np.array(res))
        # 更新已选特征和剩余特征
        mrmr_fea.append(remain_fea[_index])
        remain_fea = np.delete(remain_fea, _index)
    return mrmr_fea
```

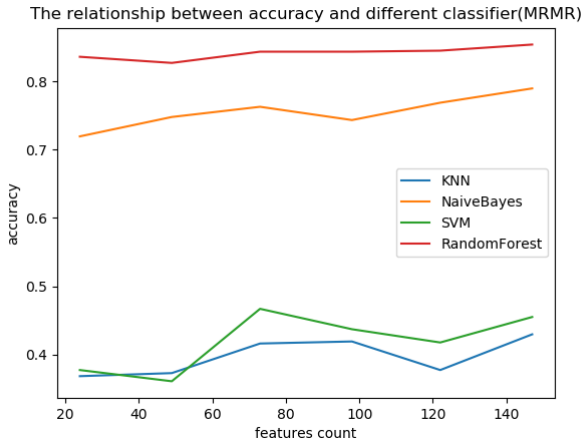
代码返回值为已选择特征值的索引。

2. 实验结果

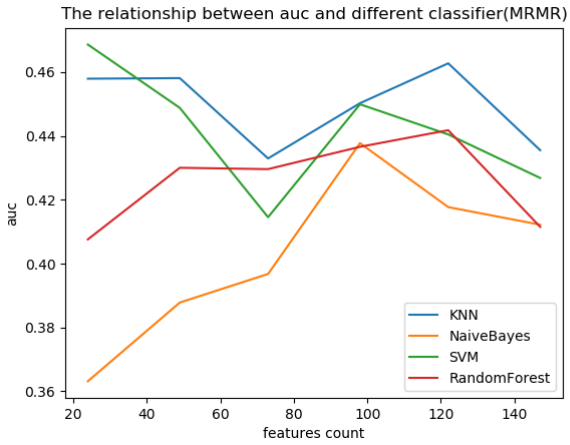
依照前文ReliefF的实验，我同样选择了排序前1/6，2/6，3/6，4/6，5/6，1的特征，比较在KNN，NB，SVM，Random Forests不同分类器的分类精度，AUC值，并用图像展示。

同上实验，分类时我也采用十折交叉验证的方法划分测试集和训练集。代码见`run(data,fea_num,method,noise=None)`。

下图是MRMR算法计算出的四种分类器分类结果的准确率示意图。



下图是MRMR算法计算出的四种分类器分类结果的AUC示意图。



从上述结果可以更明显地看出KNN和支持向量机对于样本特征的敏感性。对比上述ReliefF算法的准确度计算结果，个人认为ReliefF特征选择算法在特征选择上效果更好。

以上是本次实验中级要求部分。

实验高级要求

1. 加入高斯噪声

根据实验高级要求，需要向样本中加入50，100，150，200个噪声特征，这里我选择加入高斯噪声。

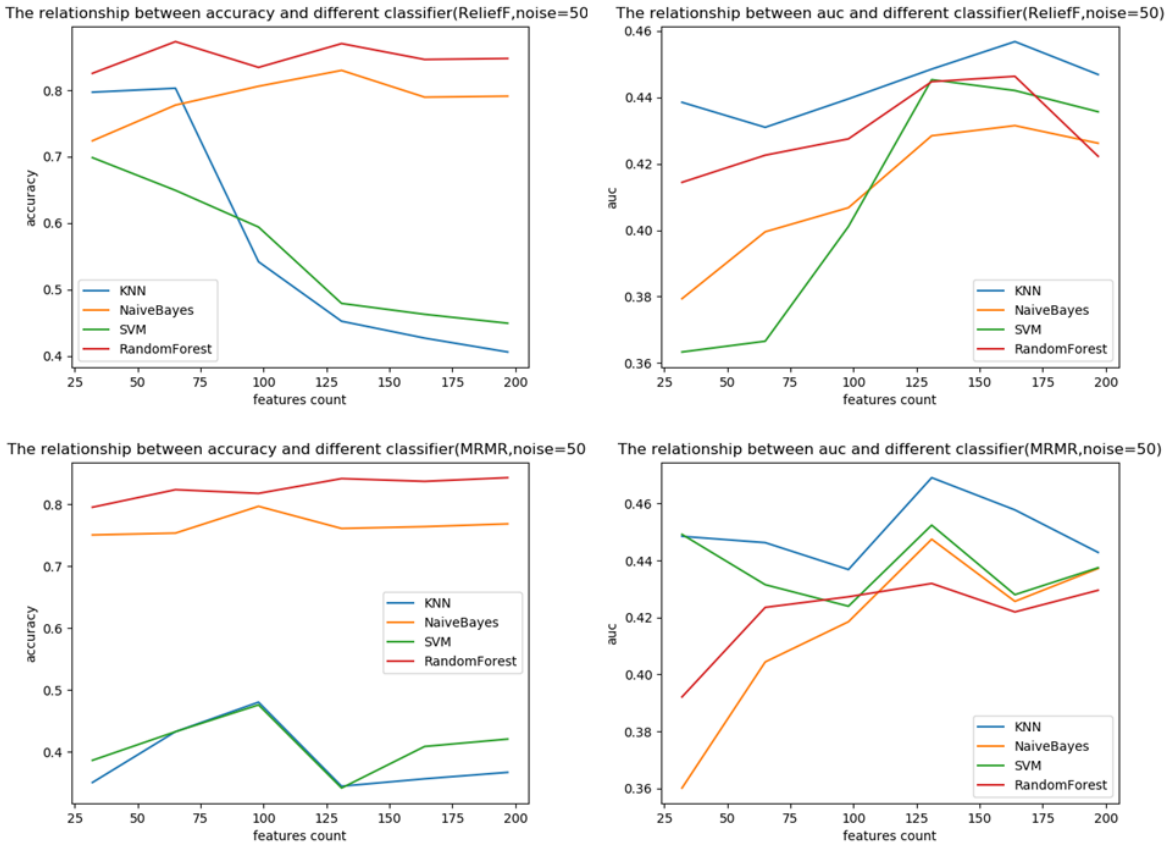
实现代码如下。

```
data = f.readpkl("data/urban.pkl")  # 训练集
noise = np.random.normal(loc=0, scale=5, size=(data.shape[0], i))
data = np.concatenate((data[:, :-1], noise, data[:, -1].reshape(-1, 1)),
axis=1)
```

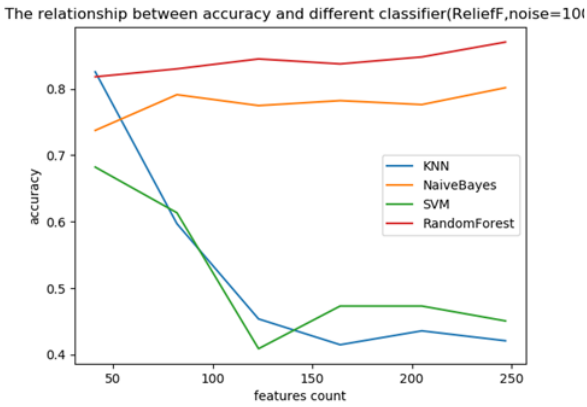
生成高斯噪声的主要方式是利用`np.random.normal()`方法，方法各参数中loc代表均值，scale代表样本方差，size代表噪声数据集的大小。在生成高斯噪声之后，需要利用`np.concatenate()`方法进行数据集的整合，整合之后即可计算实验结果。

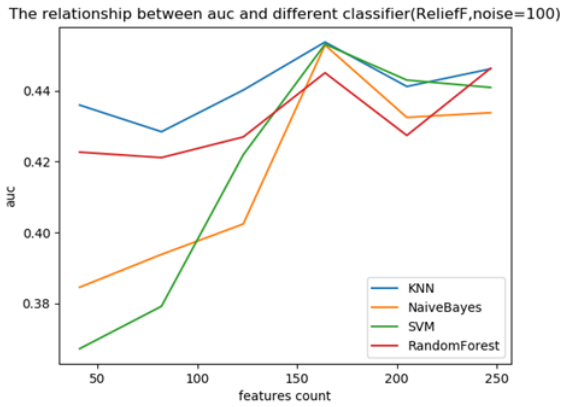
2. 实验结果

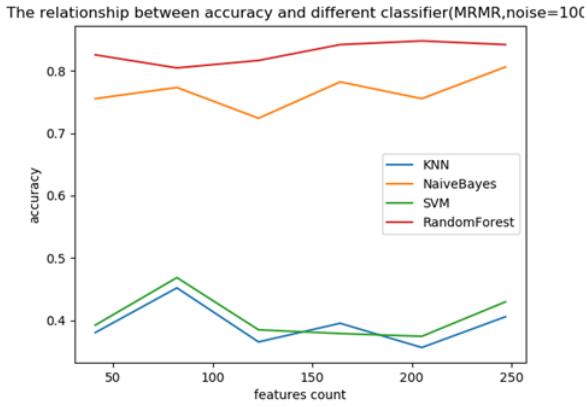
下图是数据集噪声为50时，基于ReliefF、MRMR特征选择算法在KNN、NB、SVM、RF四种分类器上的分类结果准确率和auc值随选取特征总量的变化的实验结果图。

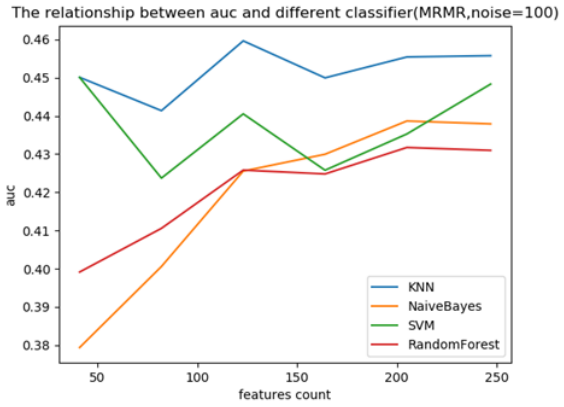


下图是数据集噪声为100时，基于ReliefF、MRMR特征选择算法在KNN、NB、SVM、RF四种分类器上的分类结果准确率和auc值随选取特征总量的变化的实验结果图。

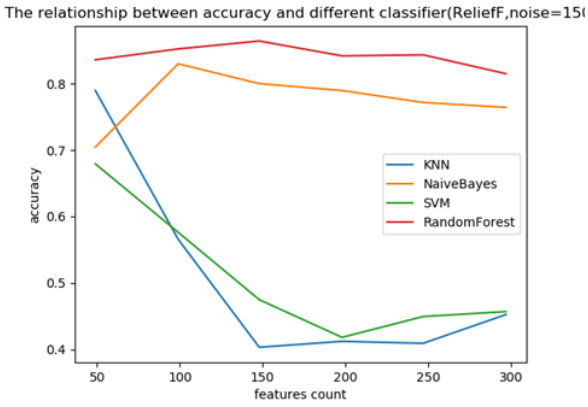


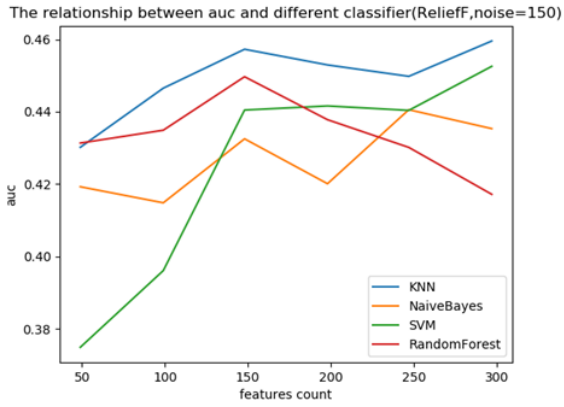


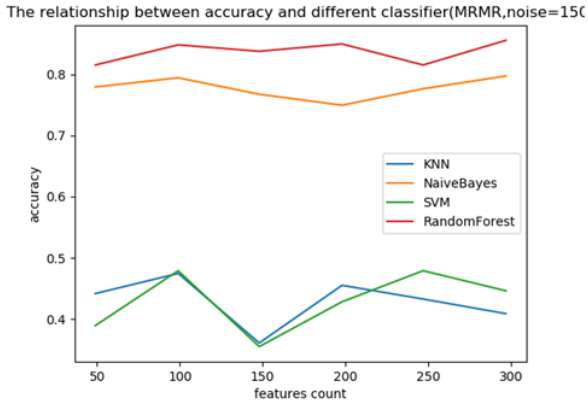


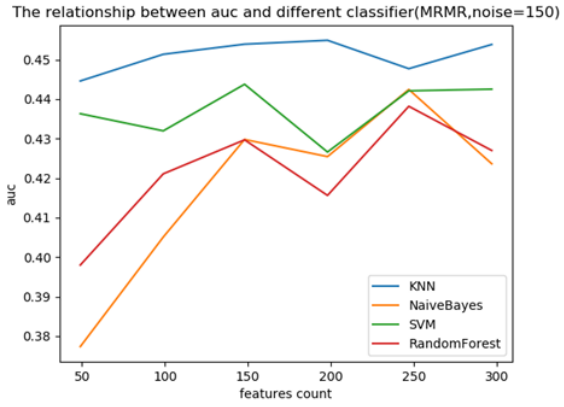


下图是数据集噪声为150时，基于ReliefF、MRMR特征选择算法在KNN、NB、SVM、RF四种分类器上的分类结果准确率和auc值随选取特征总量的变化的实验结果图。

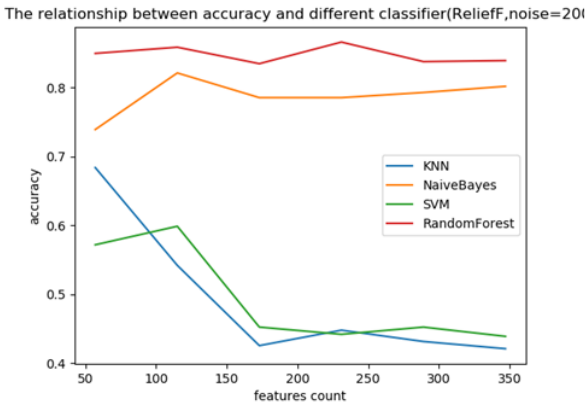


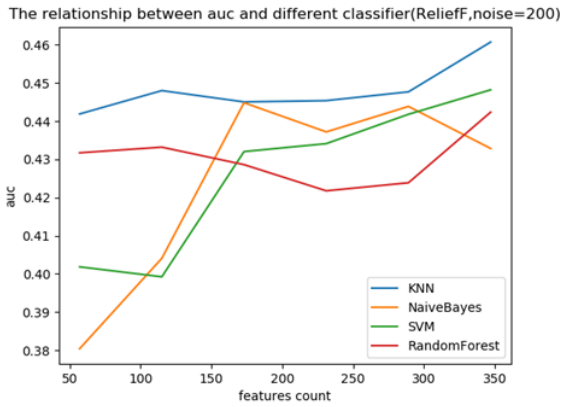


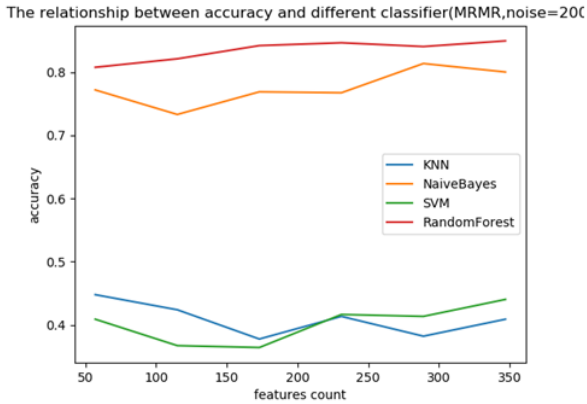


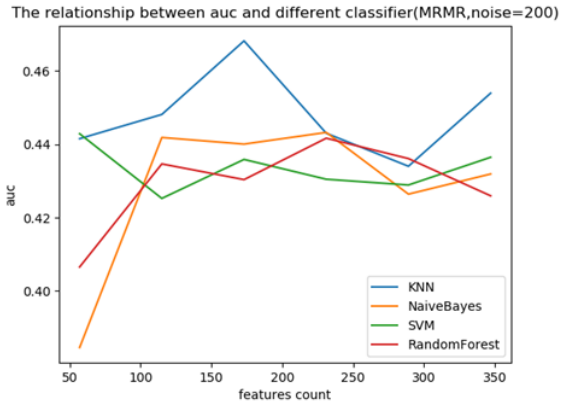


下图是数据集噪声为200时，基于ReliefF、MRMR特征选择算法在KNN、NB、SVM、RF四种分类器上的分类结果准确率和auc值随选取特征总量的变化的实验结果图。









从上述实验结果中可以看出，随着样本特征噪声数量的上升，MRMR选择出的特征在四种分类算法上整体表现自始自终较为平稳；而对于ReliefF算法，仅在随机森林分类算法和朴素贝叶斯分类算法上的分类精度和AUC值表现较为平稳，而对于KNN和SVM而言，随噪声数量增大，分类效果也随之变差。

基于上述结论，我有一个简单的猜想，即ReliefF特征选择算法是抽取出样本中最能够体现样本整体特征、整个样本中最有辨识度、最易于分类的特征集，因此可以看到在上述图像中，当ReliefF抽取的样本特征数较少时，KNN和SVM都具有不错的分类精度，因为此时筛选出的特征具有鲜明的辨识度，对于对样本比较敏感的KNN和SVM而言是易于分类的，而随样本特征数量增加，这种“辨识度”越来越模糊，导致其分类精度下降。而RMRM特征选择算法则是使得抽取出的特征能够尽可能地还原原始样本中的整体特征，想当与是原始样本集的“微缩版”，相比于“个性鲜明”的ReliefF算法，RMRM算法抽取的特征能够最大程度地保留原始样本的真实情况，因此可以看出，随着抽取特征的增加，KNN和SVM的分类精度一直很低，且变动幅度非常小，这是因为保留了原始样本特征的特征集对这两种分类算法而言是非常“模糊”的，而不像ReliefF抽取出的特征那样“个性鲜明”，因此不容易进行分类。

因此，综上，ReliefF适用于抽取出样本中最具“辨识度”的特征集，而RMRM适用于抽取出能够保留原真实样本特征的特征集，根据需求及分类算法等因素的不同，二者各有优劣，不好断言谁好谁坏。但值得指出的一点是，ReliefF算法在时间复杂度上远远优于MRMR算法，使用前者执行效率更高，这在我实际实验过程中也深有感触。

以上是本次实验高级部分要求。

以上是本次实验报告内容，感谢批阅！

同时也感谢一学期以来助教老师们的辛苦付出，这门课真的让我收获很大，更加坚定了我之后从事机器学习相关领域研究与工作的志向。传道授业之恩定当铭记于心！
