# 1.题目

**音乐年份预测**

本课题采用的数据集来自于“百万歌曲数据集（Million Song Dataset, MSD）”，百万歌曲数据集包括了 1,000,000 首当代流行音乐的音频特征和元数据。音频特征是对歌曲的音频的数据描述，包括响度、音色、音调等。歌曲的元数据指歌曲名称、演唱者、词曲作者、发行公司、发行年份等信息，它们不能在歌曲的音频中直接体现。每首歌用一个.h5 文件储存其特征，所有的文件在一个目录树中分布。.h5 文件使用 HDF5 格式储存数据，是一种层次化的数据储存格式。本数据集提供了 hdf5_getters.py 模组来读取这种格式的文件（已作为附件提供，也可在数据集官网中下载）。

任务：根据一首音乐的音频特征，推测这首音乐的发行年份。

问题的输入：音乐的特征

问题的输出：发行年份（是一个实数）

**数据集下载：**

```
wget http://millionsongdataset.com/sites/default/files/AdditionalFiles/TRAXLZU12903D05F94.h5
```

# 2.数据预处理

1. **读取h5文件，并转化为txt文件**

   参考数据集提供代码：https://github.com/tbertinmahieux/MSongsDB/blob/master/PythonSrc/hdf5_getters.py

2. **将txt文件转化为csv文件**

   a. 首先将txt文件转化为csv文件，使用python的csv库

```
import csv

raw_csv_path = "./datasets/raw_data.csv"
raw_txt_path = "./datasets/YearPredictionMSD.txt"

csvFile = open(raw_csv_path,'w',newline='',encoding='utf-8')
writer = csv.writer(csvFile)
csvRow = []

f = open(raw_txt_path,'r',encoding='GB2312')
for line in f:
    csvRow = line.split(',')
    writer.writerow(csvRow)

f.close()
csvFile.close()
```

   b. 为了后续算法和数据可视化，添加标题行

```
import csv

raw_csv_path = "./datasets/raw_data.csv"
addtitle_csv_path = "./datasets/addtitle.csv"

# set title names
header_list = ['Year']
for i in range(1,91):
    header_list.append('attr' + str(i))

# read raw csv and insert title
with open(raw_csv_path, "r") as infile:
    reader = list(csv.reader(infile))
    reader.insert(0, header_list)

with open(addtitle_csv_path, "w") as outfile:
    writer = csv.writer(outfile)
    for line in reader:
        writer.writerow(line)
```

c. 根据题目提示，保留前12个特征，以减少计算量

```
import csv
import os

addtitle_csv_path = "./datasets/addtitle.csv"
delcol_csv_path = "./datasets/data.csv"

# set the index of colums
cols_to_remove = []
for i in range(13,91):
    cols_to_remove.append(i) # Column indexes to be removed (starts at 0)

cols_to_remove = sorted(cols_to_remove, reverse=True) # Reverse so we remove from the end first
row_count = 0 # Current amount of rows processed

with open(addtitle_csv_path, "r") as source:
    reader = csv.reader(source)
    with open(delcol_csv_path, "w", newline='') as result:
        writer = csv.writer(result)
        for row in reader:
            row_count += 1
            print('\r{0}'.format(row_count), end='') # Print rows processed
            for col_index in cols_to_remove:
                del row[col_index]
            writer.writerow(row)

# remove addtitle csv
cmd = "rm -rf " + addtitle_csv_path
os.system(cmd)
```

最终效果展示（仅前10行）：

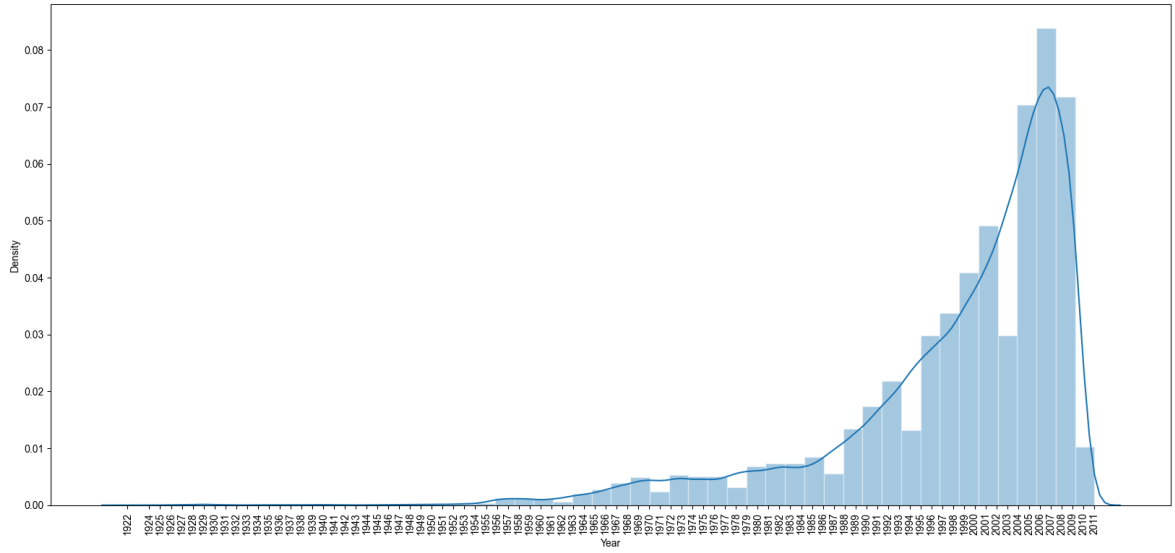
| | Year | attr1 | attr2 | attr3 | attr4 | attr5 | attr6 | attr7 | attr8 | attr9 | attr10 | attr11 | attr12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2001 | 49.94357 | 21.47114 | 73.07750 | 8.74861 | -17.40628 | -13.09905 | -25.01202 | -12.23257 | 7.83089 | -2.46783 | 3.32136 | -2.31521 |
| 1 | 2001 | 48.73215 | 18.42930 | 70.32679 | 12.94636 | -10.32437 | -24.83777 | 8.76630 | -0.92019 | 18.76548 | 4.59210 | 2.21920 | 0.34006 |
| 2 | 2001 | 50.95714 | 31.85602 | 55.81851 | 13.41693 | -6.57898 | -18.54940 | -3.27872 | -2.35035 | 16.07017 | 1.39518 | 2.73553 | 0.82804 |
| 3 | 2001 | 48.24750 | -1.89837 | 36.29772 | 2.58776 | 0.97170 | -26.21683 | 5.05097 | -10.34124 | 3.55005 | -6.36304 | 6.63016 | -3.35142 |
| 4 | 2001 | 50.97020 | 42.20998 | 67.09964 | 8.46791 | -15.85279 | -16.81409 | -12.48207 | -9.37636 | 12.63699 | 0.93609 | 1.60923 | 2.19223 |
| 5 | 2001 | 50.54767 | 0.31568 | 92.35066 | 22.38696 | -25.51870 | -19.04928 | 20.67345 | -5.19943 | 3.63566 | -4.69088 | 2.49578 | -3.02468 |
| 6 | 2001 | 50.57546 | 33.17843 | 50.53517 | 11.55217 | -27.24764 | -8.78206 | -12.04282 | -9.53930 | 28.61811 | 8.25435 | -0.43743 | 5.66265 |
| 7 | 2001 | 48.26892 | 8.97526 | 75.23158 | 24.04945 | -16.02105 | -14.09491 | 8.11871 | -1.87566 | 7.46701 | 1.18189 | 1.46625 | -6.34226 |
| 8 | 2001 | 49.75468 | 33.99581 | 56.73846 | 2.89581 | -2.92429 | -26.44413 | 1.71392 | -0.55644 | 22.08594 | 7.43847 | -0.03578 | 1.66534 |
| 9 | 2007 | 45.17809 | 46.34234 | -40.65357 | -2.47909 | 1.21253 | -0.65302 | -6.95536 | -12.20040 | 17.02512 | 2.00002 | -1.87785 | 9.85499 |

# 3.数据特征展示

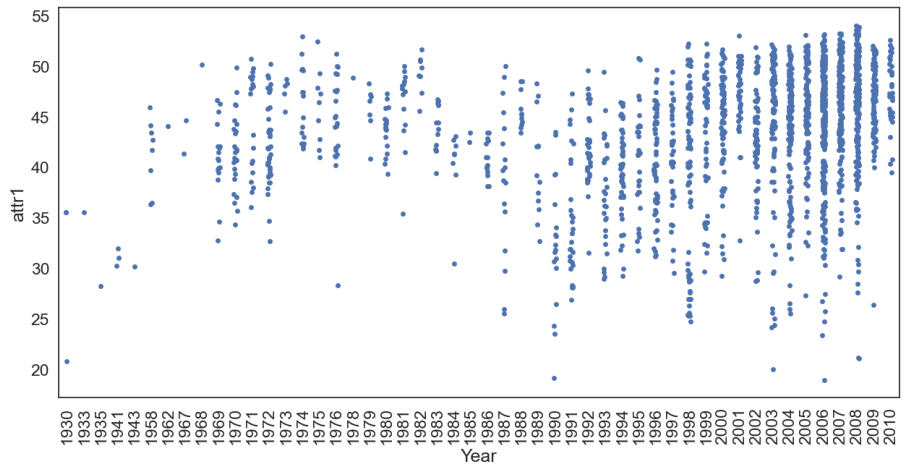
## 1.数据统计信息

| | Year | attr1 | attr2 | attr3 | attr4 | attr5 | attr6 | attr7 | attr8 | attr9 | attr10 | attr11 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 515345.000000 | 515345.000000 | 515345.000000 | 515345.000000 | 515345.000000 | 515345.000000 | 515345.000000 | 515345.000000 | 515345.000000 | 515345.000000 | 515345.000000 | 515345.000000 | 515 |
| mean | 1998.397082 | 43.387126 | 1.289554 | 8.658347 | 1.164124 | -6.553601 | -9.521975 | -2.391089 | -1.793236 | 3.727876 | 1.882385 | -0.146527 | |
| std | 10.931046 | 6.067558 | 51.580351 | 35.268585 | 16.322790 | 22.860785 | 12.857751 | 14.571873 | 7.963827 | 10.582861 | 6.530232 | 4.370848 | |
| min | 1922.000000 | 1.749000 | -337.092500 | -301.005060 | -154.183580 | -181.953370 | -81.794290 | -188.214000 | -72.503850 | -126.479040 | -41.631660 | -69.680870 | |
| 25% | 1994.000000 | 39.954690 | -26.059520 | -11.462710 | -8.487500 | -20.666450 | -18.440990 | -10.780600 | -6.468420 | -2.293660 | -2.444850 | -2.652090 | |
| 50% | 2002.000000 | 44.258500 | 8.417850 | 10.476320 | -0.652840 | -6.007770 | -11.188390 | -2.046670 | -1.736450 | 3.822310 | 1.783520 | -0.097950 | |
| 75% | 2006.000000 | 47.833890 | 36.124010 | 29.764820 | 8.787540 | 7.741870 | -2.388960 | 6.508580 | 2.913450 | 9.961820 | 6.147220 | 2.435660 | |
| max | 2011.000000 | 61.970140 | 384.065730 | 322.851430 | 335.771820 | 262.068870 | 166.236890 | 172.402680 | 126.741270 | 146.297950 | 60.345350 | 88.020820 | |

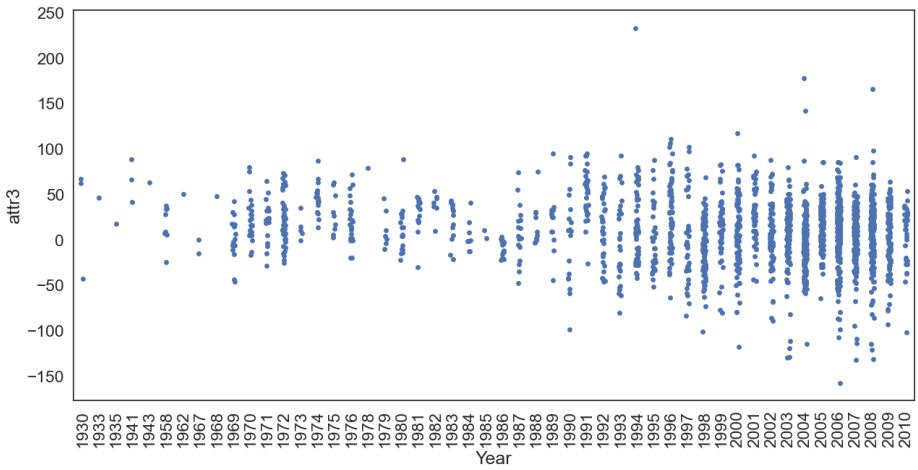
## 2.年份数据量分布图




### 3.属性的分类散点图

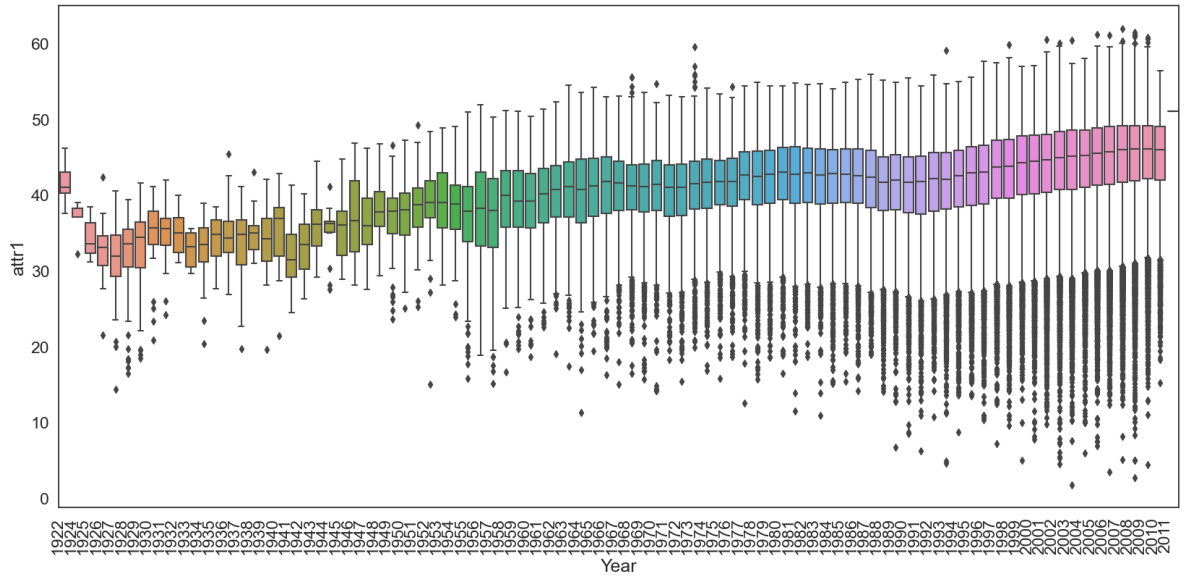



*attr1的分类散点图*

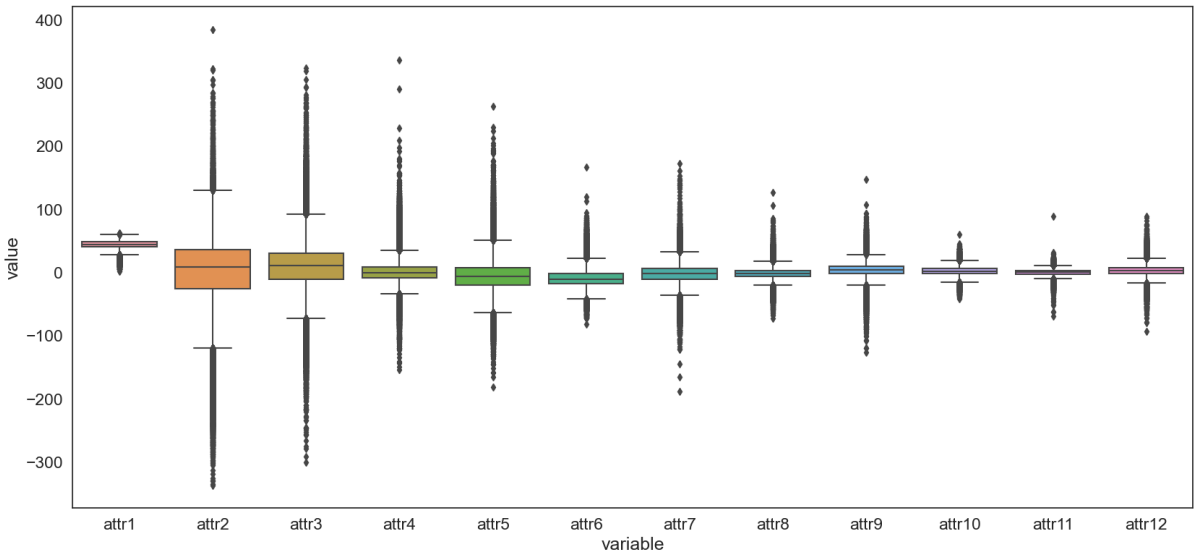



*attr3的分类散点图*

### 4.属性的箱式图

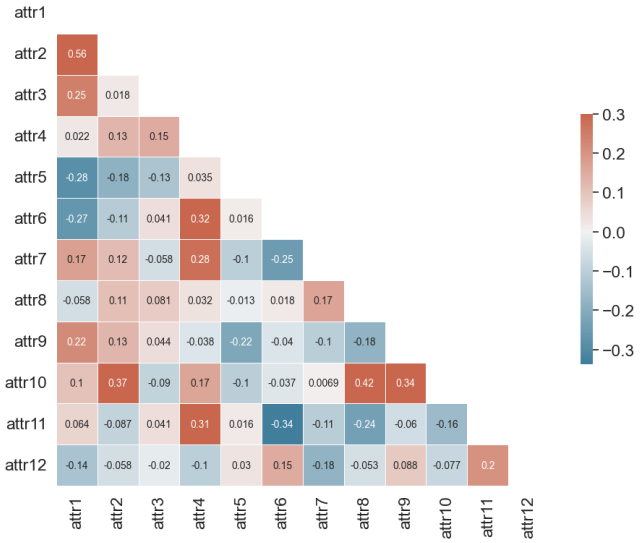



*attr1的随年份分布的箱式图*

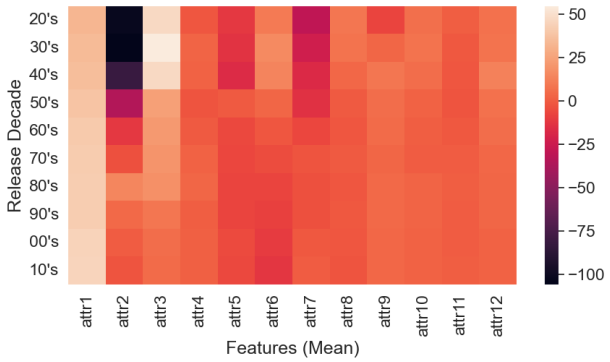



*所有属性的箱式图*

## 5.变量之间的相关性图



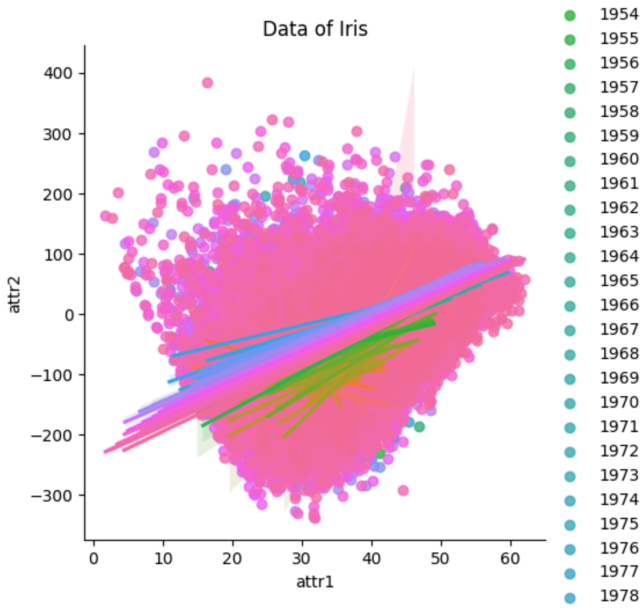
### 6.按照decade分布的热力图




## 4.分类预测

### 1.KNN法：

可视化KNN方法的参数一和参数二

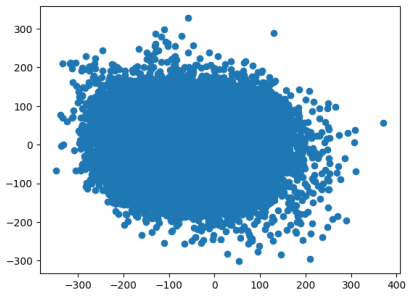



发现可视化结果的分类效果并不好，输出准确值：

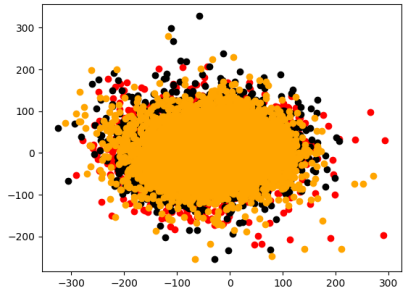
```
预测值：  [1999 2007 1996 ... 2005 2005 2008]
预测值是否等于真实值 [False False False ...  True False  True]
accuracy：  0.17546764637396187
```

### 2.PCA法：

可视化PCA降维效果（降至3维）



划分出不同年份




发现降维效果并不理想，存在很多重叠部分，对于分类任务帮助不大

### 3.SVC法：

SVC的运行时间较长，通过查看SVC的官方文档，原因是因为

> **"The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples."**

算法的时间复杂度大约是样本数量的平方，本实验中样本数量较多，将会导致较长的运行时间

### 4.随机森林：

使用留出法分割训练集和测试集，输出前100个预测值：

```
array([2001, 2010, 2003, 2007, 2005, 2007, 2009, 2009, 2006, 2008, 1994,
       2005, 2007, 1992, 1994, 1993, 1994, 2007, 1978, 2007, 2008, 2006,
       1998, 2003, 1994, 1994, 2003, 1984, 2006, 1996, 2005, 2006, 1982,
       1992, 2001, 2001, 1995, 2002, 2005, 2004, 2002, 2006, 2008, 2005,
       2007, 2005, 2003, 2009, 1998, 2002, 2002, 2006, 1994, 2004, 2006,
       2008, 1989, 2003, 2002, 2003, 1996, 2005, 2006, 1998, 2006, 2009,
       2006, 2007, 2003, 2001, 1999, 1967, 1997, 2000, 2008, 1997, 1969,
       1977, 2005, 1997, 1983, 1996, 2006, 1999, 2002, 2002, 2008, 1982,
       2003, 1986, 2002, 1982, 2001, 2000, 1982, 1966, 1990, 1999, 1995,
       2007])
```

测试集标签：

```
array([2001, 2010, 2003, 2007, 2005, 2007, 2009, 2009, 2006, 2008, 1994,
       2005, 2007, 1992, 1994, 1993, 1994, 2007, 1978, 2007, 2008, 2006,
       1998, 2003, 1994, 1994, 2003, 1984, 2006, 1996, 2005, 2006, 1982,
       1992, 2001, 2001, 1995, 2002, 2005, 2004, 2002, 2006, 2008, 2005,
       2007, 2005, 2003, 2009, 1998, 2002, 2002, 2006, 1994, 2004, 2006,
       2008, 1989, 2003, 2002, 2003, 1996, 2005, 2006, 1998, 2006, 2009,
       2006, 2007, 2003, 2001, 1999, 1967, 1997, 2000, 2008, 1997, 1969,
       1977, 2005, 1997, 1983, 1996, 2006, 1999, 2002, 2002, 2008, 1982,
       2003, 1986, 2002, 1982, 2001, 2000, 1982, 1966, 1990, 1999, 1995,
       2007])
```

随机森林各年份分类概率：

```
array([[0.  , 0.  , 0.  , ..., 0.  , 0.  , 0.  ],
       [0.  , 0.  , 0.  , ..., 0.  , 0.68, 0.  ],
       [0.  , 0.  , 0.  , ..., 0.  , 0.  , 0.  ],
       ...,
       [0.  , 0.  , 0.  , ..., 0.  , 0.  , 0.  ],
       [0.  , 0.  , 0.  , ..., 0.04, 0.  , 0.  ],
       [0.  , 0.  , 0.  , ..., 0.  , 0.  , 0.  ]])
```

预测准确率：

```
rfc.score(Xtest,Ytest)
```

```
1.0
```

可见随机森林法分类准确率达到了100%，可能存在一定程度的过拟合

> 使用留出法将会有多少样本未被投入训练集：

$$\lim_{m\to\infty}(1-\frac{1}{m})^m=\frac{1}{e}\approx 0.368$$

## 5.深度学习：

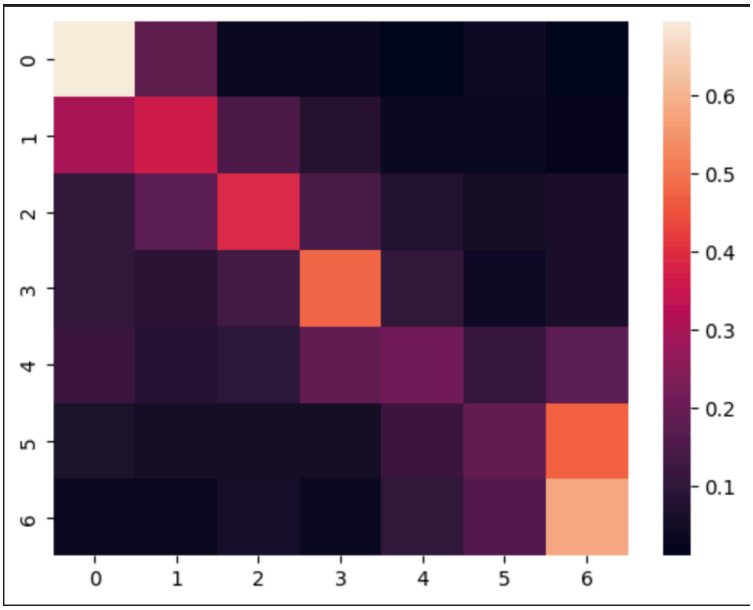
使用两层全连接层，一层softmax层作为网络结构，进行分类任务：

```
def get_network():
    model = kr.models.Sequential()
    model.add(Dense(20, input_shape=(train_features.shape[1],), activation="relu"))
    model.add(Dense(20, activation="relu"))
    model.add(Dense(label_count, activation="softmax"))
    opt = "adam"
    model.compile(loss= "categorical_crossentropy", optimizer=opt, metrics=["accuracy"], )
    return model
```

训练过程：

```
Epoch 1/10
396/396 [==============================] - 1s 1ms/step - loss: 1.8112 - accuracy: 0.2688
Epoch 2/10
396/396 [==============================] - 0s 1ms/step - loss: 1.5892 - accuracy: 0.3676
Epoch 3/10
396/396 [==============================] - 0s 992us/step - loss: 1.5016 - accuracy: 0.4120
Epoch 4/10
396/396 [==============================] - 0s 1ms/step - loss: 1.4608 - accuracy: 0.4230
Epoch 5/10
396/396 [==============================] - 2s 4ms/step - loss: 1.4369 - accuracy: 0.4354
Epoch 6/10
396/396 [==============================] - 2s 4ms/step - loss: 1.4199 - accuracy: 0.4424
Epoch 7/10
396/396 [==============================] - 1s 3ms/step - loss: 1.4060 - accuracy: 0.4470
Epoch 8/10
396/396 [==============================] - 1s 3ms/step - loss: 1.3956 - accuracy: 0.4532
Epoch 9/10
396/396 [==============================] - 1s 4ms/step - loss: 1.3860 - accuracy: 0.4564
Epoch 10/10
396/396 [==============================] - 1s 3ms/step - loss: 1.3763 - accuracy: 0.4610
99/99 [==============================] - 1s 2ms/step - loss: 1.4560 - accuracy: 0.4318
Epoch 1/10
396/396 [==============================] - 1s 1ms/step - loss: 1.8229 - accuracy: 0.2674
Epoch 2/10
396/396 [==============================] - 1s 2ms/step - loss: 1.5875 - accuracy: 0.3685
...
Epoch 10/10
396/396 [==============================] - 1s 1ms/step - loss: 1.3711 - accuracy: 0.4662
99/99 [==============================] - 0s 1ms/step - loss: 1.4661 - accuracy: 0.4188
[0.43178374 0.43380496 0.42824659 0.44239515 0.41875157]
```

训练结果通过混淆矩阵热力图展示：




混淆矩阵：

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | 0.694545 | 0.174545 | 0.036364 | 0.029091 | 0.010909 | 0.043636 | 0.010909 |
| 1 | 0.304460 | 0.367925 | 0.151801 | 0.079760 | 0.036021 | 0.036021 | 0.024014 |
| 2 | 0.099750 | 0.183639 | 0.387312 | 0.142738 | 0.073038 | 0.050501 | 0.063022 |
| 3 | 0.096882 | 0.090455 | 0.130921 | 0.473221 | 0.102595 | 0.043085 | 0.062842 |
| 4 | 0.118998 | 0.081320 | 0.094674 | 0.192051 | 0.218680 | 0.117011 | 0.177266 |
| 5 | 0.066789 | 0.051196 | 0.054241 | 0.049557 | 0.118956 | 0.190162 | 0.469098 |
| 6 | 0.029042 | 0.034850 | 0.057115 | 0.036786 | 0.101646 | 0.160697 | 0.579864 |

可见对角线上分类的概率最大，说明分类正确，分类效果较好

## 5.实验总结

想法验证，对于结果有轻微的提升：

1.划分训练集和测试集的时候不整体随机划分，而是按照年份随机划分，这样做的好处是数据分布不均匀，有的年份样本量太小，可能该年份没有加入训练集直接加入测试集造成模型未学习到；

2.进行归一化，虽然数据值的大小没有差很多个数量级，但是仍然会对权重产生轻微的影响；

3.尝试了一下深度学习的算法，用全连接网络

4.可以根据相关性图对原始的所有数据进行降维，前12个属性中部分属性对于分类效果不明显，根据原始数据降至12维可能会对最后的结果带来一定的帮助

5.按照decades表示，会减少计算量，更便于验证想法

6.根据数据分布特征，可以尝试核函数

7.许多sklearn的算法不支持GPU加速，因为sklearn在设计时就有意识的设置成CPU，方便跨设备使用，可以手写cuda，加速训练

8.设置验证集，部分算法极有可能过拟合