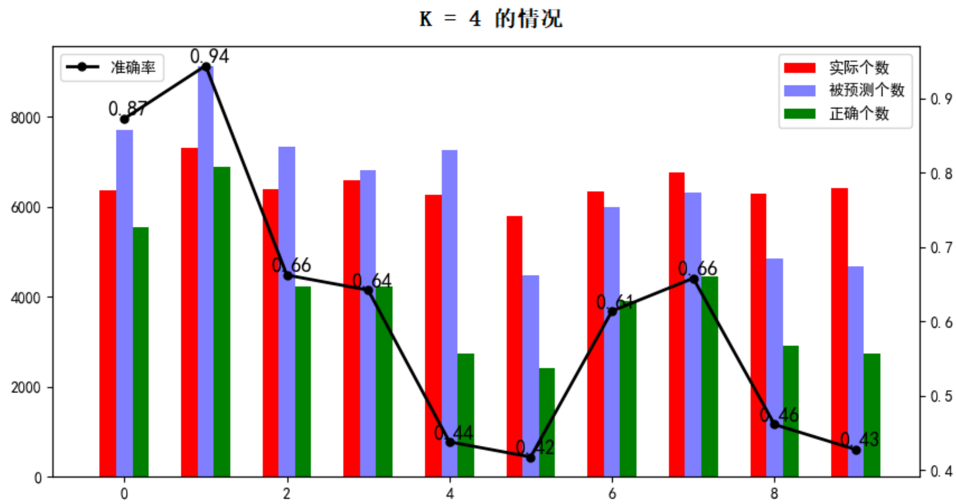
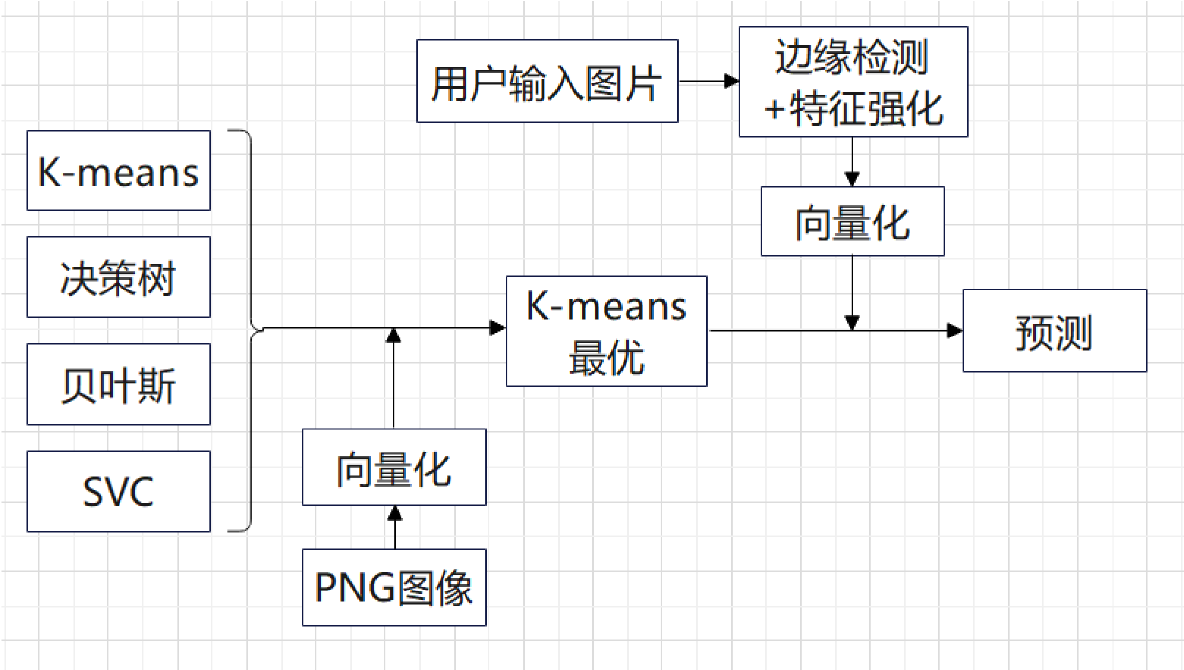
基于机器学习的手写数字识别系统

> **写在前面**
>
> 怎么说呢， K-means做识别感觉好像准确率不太，想用CNN，但是好像这是基于机器学习的大作业，用CNN好像不妥
>
> 于是就只用了K-means， 但其实我要是复习的差不多有时间的话想增加...（不太可能了）
>
> 第一次尝试前端与后端交互共同实现，也是学到了很多新东西，顺便把web作业也水了，开心😄.....
>
> 总结一下吧：**在 0~5 的情况下准确率较高， 6~9 准确率不堪入目**
>
> 
>
>
> **下面就开始正式的汇报**

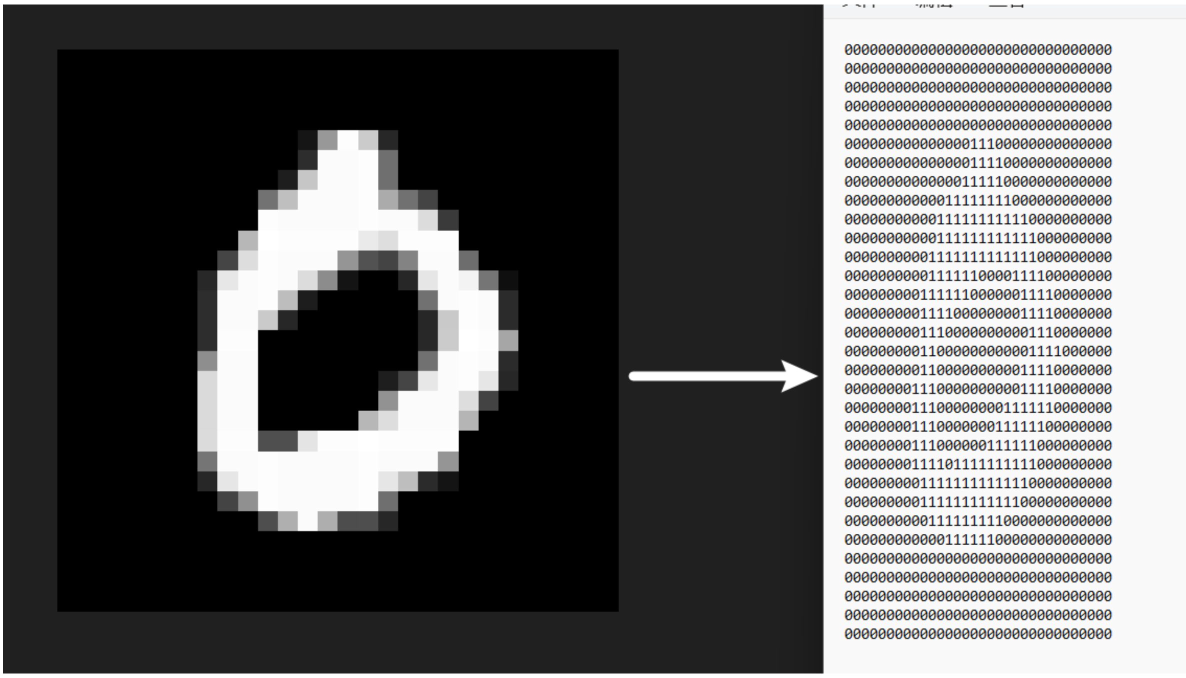
流程



训练数据处理

本次训练的数据集为MNIST数据集，训练初期目标为 32×32，数据集大小为 28×28，所以要先放大图片

接着对图片进行二值化，对于训练的预期是输入白底黑字的图片，而MINIST的图片是黑底白字，所以对于训练集，如果灰度大于127，取特征为 1， 否则取 0。



核心代码

```
def readImg(Road):
    image = cv2.imread(Road, cv2.IMREAD_GRAYSCALE)
    down_width = 32
    down_height = 32
    down_points = (down_width, down_height)
    image = cv2.resize(image, down_points, interpolation=cv2.INTER_LINEAR)
    Info = []
    for i in range(32):
```

```
        for j in range(32):
            if image[i][j] >= 127:
                Info.append(1)
            else:
                Info.append(0)
    return Info
```

模型选择

考虑四个模型， 分别是 *K-means* ， *贝叶斯*， *决策树* ， *SVC*

对同样的数据进行处理准确率分别为：

- K-means : 0.6324
- 贝叶斯： 0.4236
- 决策树： 0.5012
- SVC： 0.5326

经过第一步粗略的统计，决定采用K-means算法

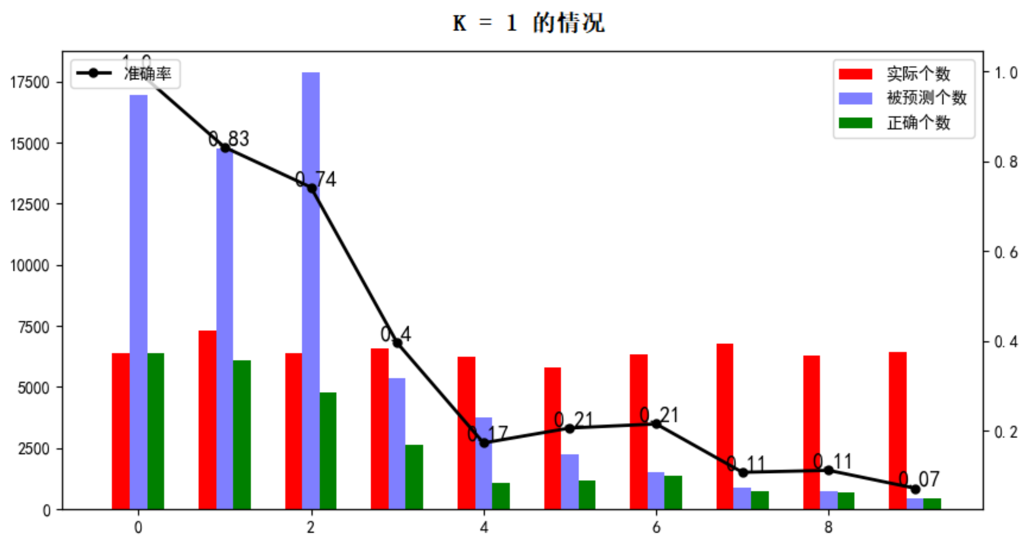
```
trainFeature, trainLabel = DataPre.getTrainFeature()
testFeature, Label2 = DataPre.getInputFeature("../data/Img", "../data/Text")
Label2 = [int(pr) for pr in Label2]
# KNN
def k_means(K):
    model = KNeighborsClassifier(n_neighbors=K)
    model.fit(trainFeature, trainLabel)
    return model
# Bayes
def Bayes():
    model = GaussianNB()
    model.fit(trainFeature, trainLabel)
    return model.predict(testFeature)
# Tree
def Tree():
    dtc = DecisionTreeClassifier()
    dtc.fit(trainFeature, trainLabel)
    return dtc.predict(testFeature)
def SVC_():
    model = SVC(kernel="linear")
    model.fit(trainFeature, trainLabel)
    return model.predict(testFeature)
```

模型优化

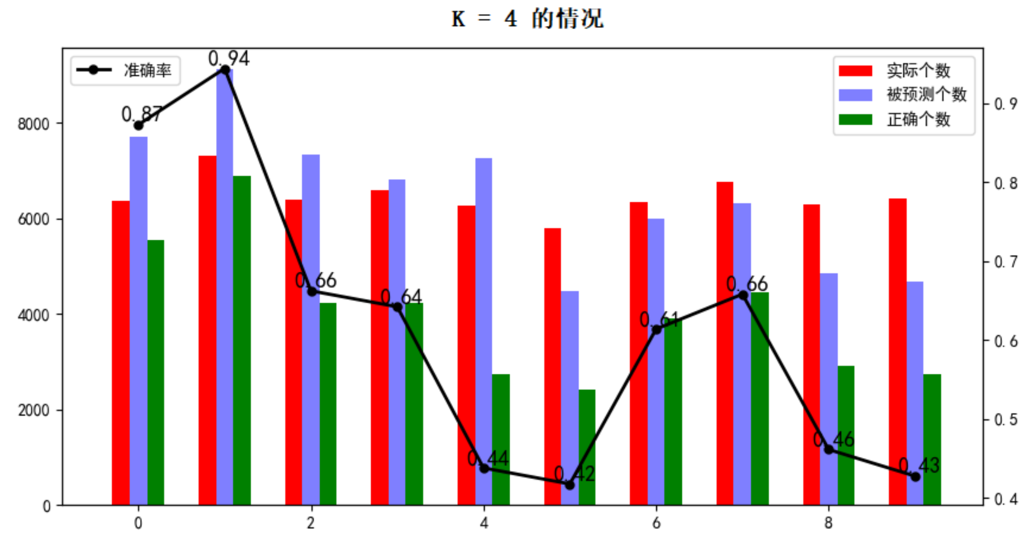
对于 K-means算法来说，最重要的莫过于K值的选择，如何判断最优情况，决定遍历选择

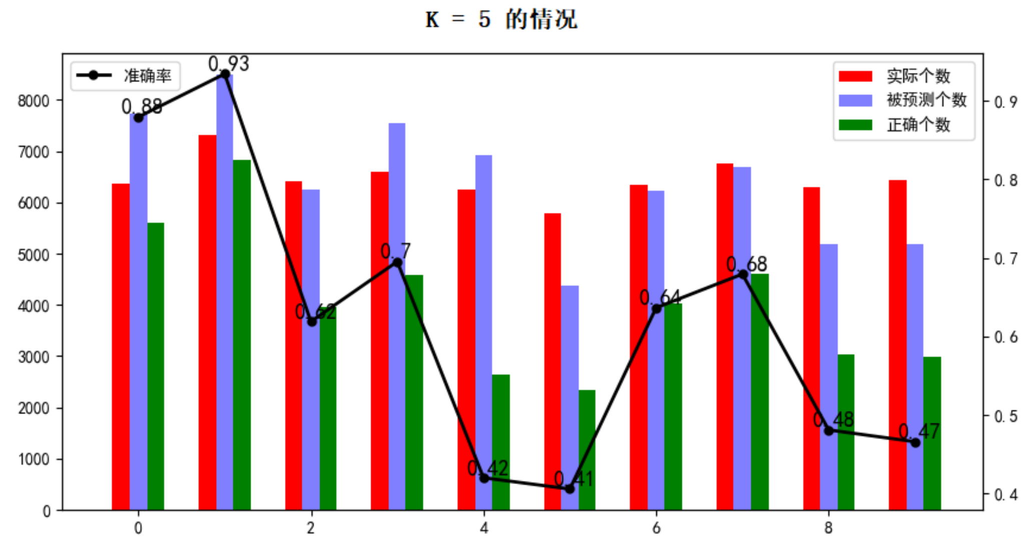
此次遍历遍历 K 从 1 ~ 9 的情况

为了简介表述以及方便观看，我们只挑选有代表性的展示



可以观察到 K = 1 的时候， 后面数字的准确度堪忧，所以我们舍弃 K = 1 的情况， K = 1， 2， 3 均为此种情况





可以观察到 K = 4， 5， 6 ~ 9 的时候较为合理，所以我们选取的 K 在这个区间段。

各种情况的运行日志如下：

> read data cost 59.17566 second
>
> the model which K is 1 cost 101.19315 second
>
> - A model with K of 1 has an accuracy of **0.39235**
>
> the model which K is 2 cost 103.17654 second
>
> - A model with K of 2 has an accuracy of **0.35766**
>
> the model which K is 3 cost 104.81046 second
>
> - A model with K of 3 has an accuracy of **0.42246**
>
> the model which K is 4 cost 113.85034 second
>
> - A model with K of 4 has an accuracy of **0.62091**
>
> the model which K is 5 cost 115.52863 second
>
> - A model with K of 5 has an accuracy of **0.62909**
>
> the model which K is 6 cost 119.29570 second
>
> - A model with K of 6 has an accuracy of **0.63032**
>
> the model which K is 7 cost 108.92201 second
>
> - A model with K of 7 has an accuracy of **0.63046**
>
> the model which K is 8 cost 105.33319 second
>
> - A model with K of 8 has an accuracy of **0.62913**
>
> the model which K is 9 cost 106.15953 second
>
> - A model with K of 9 has an accuracy of **0.63036**

虽然 K = 9 的准确度更高， 但考虑到泛化误差，依然以 K = 4 作为最后的最优选择

注意到所有 K 值的情况下在 0 , 1 上预测数量较大， 在 4 , 5 的范围内暴跌， 所以尝试控制数据集来减缓

在初始的时候，各个数据量的分布情况如下：

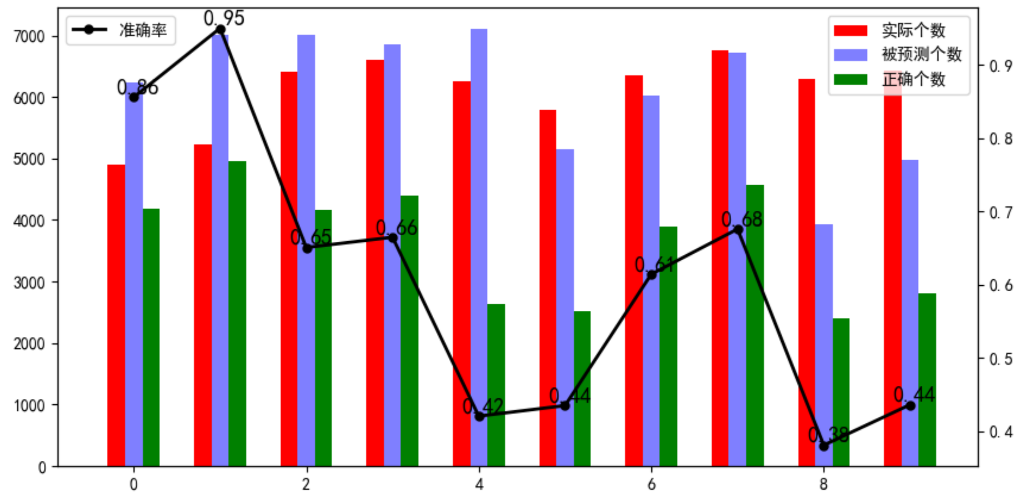
> **[6372. 7307. 6405. 6602. 6259. 5785. 6345. 6767. 6298. 6422.]**

于是我们分别减缓 0 , 1 的数据量来控制预测的个数以及质量

我们选取了两个典型分布分析

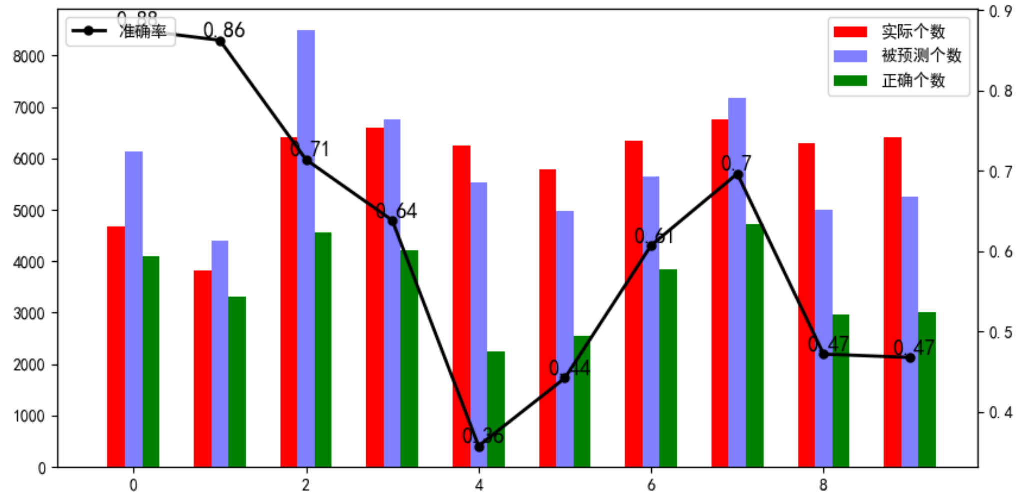
分布一：

> **[4892. 5226. 6405. 6602. 6259. 5785. 6345. 6767. 6298. 6422.]**
>
> A model with K of 4 has an accuracy of **0.59888**



分布二：

> **[4685. 3830. 6405. 6602. 6259. 5785. 6345. 6767. 6298. 6422.]**
>
> A model with K of 4 has an accuracy of **0.59802**



经过分析可知，分布二对情况四的预测情况更加下跌，而分布一在对 8 的准确率下跌，其他情况类似，于是保持原数据不动

用户输入图像优化

在最开始的测试情况下，因为使用的平板书写，所以痕迹较粗，转到拍照上传后发现出现痕迹过细无法识别情况，于是添加边缘检测以及特征优化功能

> 测试情况

> 实际情况

于是决定对边缘进行识别并优化

于是打算采取 $Canny$ 算法去进行边缘检测并加在图像上

```
Use = cv2.Canny(image, 10,10)
for i in range(32):
    for j in range(32):
        if Use[i][j] == 255:
            Info[i * 32 + j] = 1
            img[i][j] = 255
```

> 实际效果如下

观察到仍有不完全存在，于是进行轮廓识别并绘制

```
_, image = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)
_, contours, _ = cv2.findContours(image, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
cv2.drawContours(image, contours,-1, (0, 255, 0), 2)
```

> 实际效果

观察到边框为白色，进行去除

```
for i in range(32):
    for j in range(32):
        if i == 1 or i == 0 or i == 31 or i == 30:
            Info[i*32 + j] = 0
            img[i][j] = 0
            if j == 1 or j == 0 or j == 31 or j == 30:
                Info[i*32 + j] = 0
                img[i][j] = 0
```

至此，优化完成