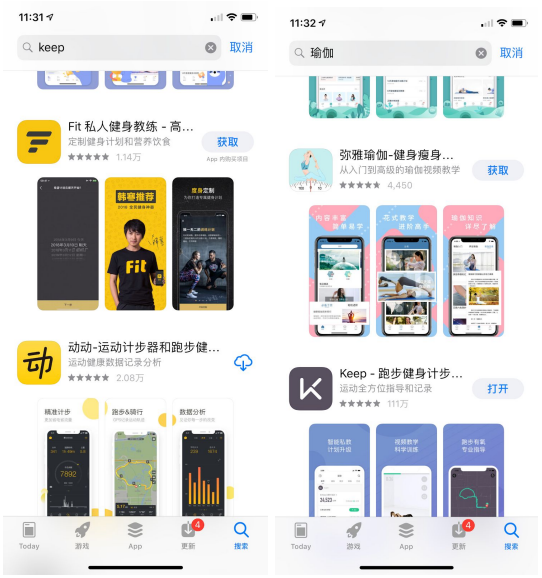
决策树

要解决的问题

SEO 相信大家都了解，是网站中常用的对搜索引擎优化的一种做法，全称 Search Engine Optimization，通过一些词，根据词频、相关度等等让搜索引擎更好的找到该网站的一种方式。ASO 是 App Store Optimization，类似SEO 也是根据在每个App 后台通过标题、关键词去设置一定相关熟悉的次来提升自己的曝光。

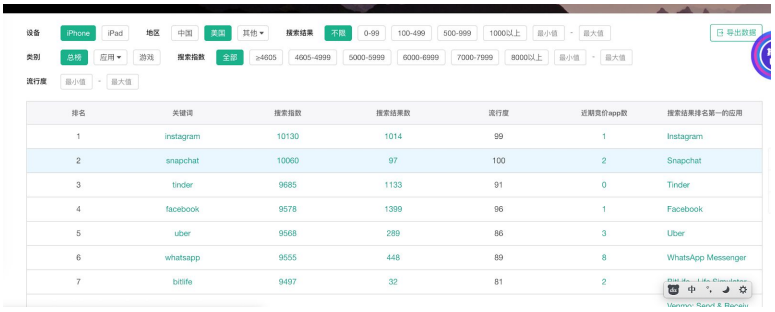


比如我们搜 Keep 这个次之外，还会出现很多跟Keep 相关的App，这些App 或多或少就蹭了Keep 的热度，当然也是符合整个ASO 用户推荐的逻辑的，相关度高的排到最前面。
另外如搜瑜伽，也能出现 Keep 的App

通过这部分来的流量称谓搜索流量，很显然这部分流量一毛钱不用花，就可以获得自然免费流量，那么如何优化App 的关键词对于一款 App 尤为重要，尤其是我们的刚刚起步的海外版。

那么前期海外版 App 改如何选词呢？

首先我们通过 ASO 第三方数据平台看看相关的次，例如七麦数据：
https://www.qimai.cn/trend/keywordRank/country/us



可以看到这里有很多关键词，同时有相关的搜索指数、结果数、流行度、排名第一的应用等等。所以对应ASO 优化这块最初级的筛选就是 搜索指数、流行度要高，结果数少， 相似度(可以根据近义词或者标题中是否包含来)强来决定。

目前的选词基本上是人工选，目前词多数据工作量大，并且不一定选出来的一定匹配。而且每次发版本都要做 ASO 优化，对于筛次人员来说不仅工作量大，工作内容重复，而且纯靠人工经验。

所以我们要解决的问题就转化为给定一组数据，根据搜索指数、流行度、结果数、相似度来判断次是否属于我们想要加的关键词。其中特征为 搜索指数、流行度、结果数、相似度，数据为判定其为大、小还是高、低，使用决策树来给出预测模型，进行数据分类

准备数据

下载七麦的数据，进行头尾的无用数据清除。由于导出数据的限制，目前只能导出 5000 条数据。

关键词	搜索指数	搜索结果数	搜索流行度	搜索结果排名第一的应用
instagram	10130	1014	99	Instagram
snapchat	10060	96	100	Snapchat
tinder	9685	1115	91	Tinder
facebook	9578	1399	96	Facebook
uber	9568	289	86	Uber
whatsapp	9555	448	89	WhatsApp Messenger
bitlife	9497	32	81	BitLife - Life Simulator
venmo	9487	32	82	Venmo: Send & Receive Money
fortnite	9283	195	89	Fortnite
youtube	9244	1305	92	YouTube: Watch, Listen, Stream
cash app	9241	1555	82	Cash App
spotify	9214	829	86	Spotify Music

我们要筛选的是和健身相关的词。所以相关性这里我们筛选排名第一中的应用中名字是否包含和健身相关的次，例如 Fitness、HIIT、Health 等, 要求搜索结果数 < 1000 搜索指数 > 4800 ， 流行度 > 40 以及搜索关键词或名称包含如上词语的结果

我们在数据处理的时候最后加上一列表示结果，如果是我们想要的为 success，不是的话，为 failed

遍历整个数据，把 搜索结果数 < 1000 搜索指数 > 4800 ， 流行度 > 40 以及搜索关键词或名称包含如上词语的结果置为满足条件，不符合条件的置为不符合的条件, 最终生成其特征所对应的值如下：

```
fix_word_data = []
if search_index > 4800:
    fix_word_data.append("valid search index")
else:
    fix_word_data.append("invalid search index")

if search_result < 1000:
    fix_word_data.append("valid search result")
else:
    fix_word_data.append("invalid search result")

if search_popular > 40:
    fix_word_data.append("popular")
else:
    fix_word_data.append("unpopular")

if "fitness" in app_name.lower() or "hiit" in app_name.lower():
    fix_word_data.append("valid words")
else:
    fix_word_data.append("invalid words")

if search_index > 4800 and search_result < 1000 and search_popular > 40 and (
        "fitness" in app_name.lower() or "hiit" in app_name.lower()):
    print ('keyword is ' + keyword)
    fix_word_data.append("success")
else:
    fix_word_data.append("failed")
```

数据拆分好之后，我们开始用 DT 进行分类和决策，根据样本进行训练和测试区分，取70%训练，30%测试

```
def create_train_test_data(all_data, label):
    size = int(len(all_data) * 0.7)
    t_train_data = all_data[:size]
    t_test_data = all_data[size:]
    test = []
    for test in t_test_data:
        t_dict = {}
        for la in range(len(label)):
            t_dict.setdefault(label[la], test[la])
        test.append(t_dict)

    return t_train_data, test
```

设计思路

决策树

首先我们划分数据集的目标是让无序的数据更加有序，划分数据的方法是多种的，比如我们可以先用搜索结果数分，也可以先用流行度分等等。但是哪一种分类更有效更加科学呢？

根据熵（entropy）和信息增益(information gain)知道，熵越大越混乱，我们尽可能的使得分类后的熵越小，也就是信息增益越大。

根据熵的公式得知，我们知道每种特征出现的概率可以求得对应的数据集的熵

$$H = -\sum_{i=1}^{n} p(x_i) log_2 p(x_i)$$

转化为代码如下

```
def cal_shannon_entropy(data_set):
    num = len(data_set)
    label_counts = {}
    for data in data_set:
        current_label = data[-1]
        if current_label not in label_counts.keys():
            label_counts[current_label] = 0
        label_counts[current_label] += 1
    shannon_entropy = 0
    for key in label_counts:
        prob = float(label_counts[key]) / num
        shannon_entropy -= prob * log(prob, 2)
    return shannon_entropy
```

接下来，我们来获取选择出最佳特征，也就是信息增益最大的特征。信息增益是原始熵和按照某种规则排之后熵的差，把所有数据和特征塞入即可，原生熵为0.03763145983067001。

接下来我们按照每一个特征去划分，并且求出其信息增益，判断信息增益最大的为有限划分的特征，选为 best feature 实现代码如下

```python
def choose_best_feature(words_data):
    feature_num = len(words_data[0]) - 1
    base_entropy = cal_shannon_entropy(words_data)
    best_info_gain = 0
    best_feature = -1
    for i in range(feature_num):
        feature_list = [feature[i] for feature in words_data]
        new_entropy = 0
        for value in set(feature_list):
            sub_data = []
            for words in words_data:
                if words[i] == value:
                    sub_vec = words[:i]
                    sub_vec.extend(words[i+1:])
                    sub_data.append(sub_vec)
            prob = len(sub_data) / float(len(words_data))
            new_entropy += prob * cal_shannon_entropy(sub_data)
        print('new_entropy is + ' + str(new_entropy))
        info_gain = base_entropy - new_entropy
        if info_gain > best_info_gain:
            best_info_gain = info_gain
            best_feature = i
    return best_feature
```

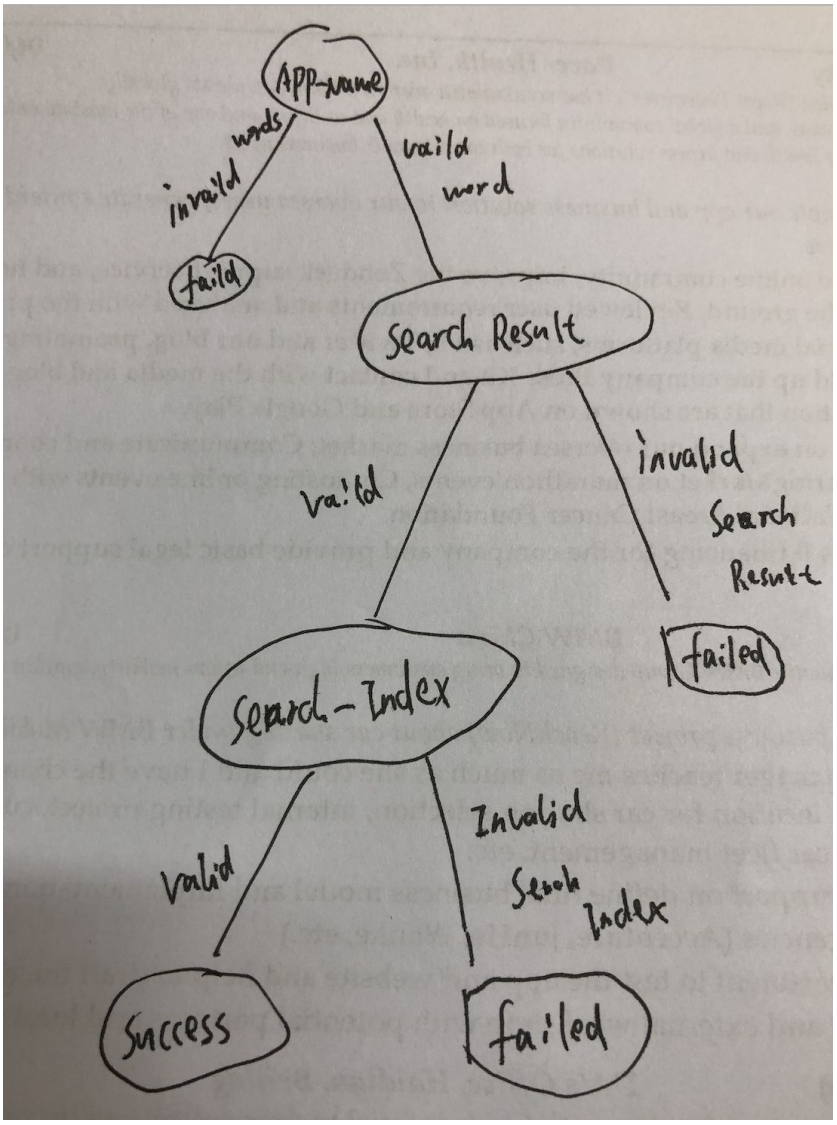
每次用完一个特征之后，删掉特征，重新选取剩下特征中的信息增益最大的，即best feature

```python
def create_tree(labels, words_data):
    result_list = [example[-1] for example in words_data]
    if result_list.count(result_list[0]) == len(result_list):
        return result_list[0]
    if len(words_data[0]) == 1:
        # vote for result
        class_count = {}
        for vote in result_list:
            if vote not in class_count.keys():
                class_count[vote] = 0
            class_count[vote] += 1
        sorted_count = sorted(class_count.items(), key=operator.itemgetter(1), reverse=True)
        return sorted_count[0][0]
    best_feature = choose_best_feature(words_data)
    best_feature_label = labels[best_feature]
    decision_tree = {best_feature_label: {}}
    del (labels[best_feature])
    feature_values = [example[best_feature] for example in words_data]
    for value in set(feature_values):
        sub_labels = labels[:]
        sub_data = []
        for words in words_data:
            if words[best_feature] == value:
                sub_vec = words[:best_feature]
                sub_vec.extend(words[best_feature + 1:])
                sub_data.append(sub_vec)
        decision_tree[best_feature_label][value] = create_tree(sub_labels, sub_data)
    return decision_tree
```

用 decision_tree 保存选择的 feature 和结果，最终生成树如下

{'App-Name': {'valid words': {'Search-Result': {'valid search result': {'Search-Index': {'invalid search index': 'failed', 'valid search index': 'success'}}, 'invalid search result': 'failed'}}, 'invalid words': 'failed'}}

如果用手工的方式画出来的话就是



先进行 App-Name 分，在通过 Search-Result，最后通过 Search-Index，同时也可以发现在目前的分类情况下，和搜索的流行度也并没有什么关系。所以如果还是人工筛选的话，提供了一条最优路径，即如上图的判断所示。

最终挑出可用的关键词如下：

keyword is my fitness pal
keyword is calorie counter
keyword is myfitnesspal
keyword is orange theory fitness
keyword is mindbody
keyword is interval timer
keyword is calorie tracker
keyword is aaptiv
keyword is 30 day fitness
keyword is sweat
keyword is ifit

keyword is hiit workouts
keyword is anytime fitness
keyword is wahoo fitness
keyword is crunch fitness
keyword is butt workout
keyword is asana rebel

除了 DT 之外，为了使得结果更好。使用随机森林和 Ada 来进行测试

随机森林

首先进行 n 个样本采样，这里我们取得是10 。

```
def cross_validation_split_for_random_forest(data_set, n):
    data_set_split = list()
    data_set_copy = list(data_set)
    fold_size = int(len(data_set) / n)
    for i in range(n):
        fold = list()
        while len(fold) < fold_size:
            index = random.randrange(len(data_set_copy))
            fold.append(data_set_copy.pop(index))
        data_set_split.append(fold)
    return data_set_split
```

其次进行属性中随机选择k个属性，选择最佳分割属性作为节点建立决策树，重复m 次之后，来进行投票表决

```
def cal_random_forest_prob(all_data, label):
    datas = cross_validation_split_for_random_forest(all_data, 10)
    for data in datas:
        l = list(label)
        train_data = list(datas)
        train_data.remove(data)
        da = []
        for d in train_data:
            da += d
        test_data = data
        td = convert_test_data(test_data, l)
        tree = create_tree(l, da)
        print(tree)
        valid_data(td, tree)
```

Ada boost

Ada 核心思想是给不同的分类器加权，也就是好的分类器权重越高，把多个弱分类器组合为强分类器。
权重和分类器错误率相关，由下面公式得到错误率越低，权重越大。根据错误率来修正权重，训练 N 次后最后找到总错误率小于一个阈值完成迭代。初始权重取总样本分之一

$$\alpha_m = \frac{1}{2}\log(\frac{1-e_m}{e_m})$$

转化为代码

```
def cal_alpha(error):
    alpha = 1 / 2 * log((1 - error) / error)
    return alpha
```

根据 alpha 和 tree 验证的错误率得到新的权重

```
def cal_new_weight(alpha, weight):
    new_weight = []
    sum_weight = 0
    for i in range(len(weight)):
        flag = 1
        weighti = weight[i] * exp(-alpha * flag)
        new_weight.append(weighti)
        sum_weight += weighti
    return new_weight
```

训练强分类器，定义训练的错误率阈值和次数

```
def train_ada_boost(label, data, error_threshold, max_iter):
    fx = {}
    weight = []
    num = len(label)
    for i in range(num):
        w = float(1 / num)
        weight.append(w)

    for i in range(max_iter):
        fx[i] = {}
        td = convert_test_data(data, label)
        prob = valid_data(td)
        error = (weight[i] * prob)
        alpha = cal_alpha(error)
        weight = cal_new_weight(alpha, weight)
        fx[i]['alpha'] = alpha / num
        if error < error_threshold:
            break
    return fx
```

最终打印每个训练器权重如下
{0: {'alpha': 0.24361662108864732}, 1: {'alpha': 0.3714040396090309}, 2: {'alpha': 0.5591424223557279}}

测试和对比

我们分别执行决策树和随机森林以及 Ada算法，打出准确率和运行时间

```
if __name__ == '__main__':
    labels, data = load_words_data()
    l = list(labels)
    random.shuffle(data)

    train_data, test_data = create_train_test_data(data, labels)
    t = time.time()
    tree = create_tree(labels, train_data)
    valid_data(test_data, tree)
    print('time gap is:' + str(time.time() - t) )
    print (tree)

    t1 = time.time()
    cal_random_forest_prob(data, l)
    print('time t1 gap is:' + str(time.time() - t1) )

    ada = train_ada_boost(l, data, 0.01, 5)
    print(ada)
```

DT：

```
base_entropy is : 0.7106768538561 23
new_entropy is + 0.710676853856123
prob is 0.712
time gap is:0.033406972885131836
{'Search-Result': {'invalid search result': 'failed', 'valid search result': {'App-Name': {'valid words'
base_entropy is : 0.5332398959774798
```

准确率 71.2%
耗时：0.033

随机森林

```
new_entropy is + 0.7256040085121113
new_entropy is + 0.7128481244254435
base_entropy is : 0.7168319999345276
new_entropy is + 0.7165149469934304
{'Search-Result': {'invalid search result': 'failed', 'valid search result': {'Search-Index': {'valid search index': {'Ap
prob is 0.875751503006012
max is 0.9018036072144289
time t1 gap is:0.578096866607666
```

准确率 90.18036%
耗时：0.578

首先随机森林的效果要明显好于决策树，随机森林在经过多个随机采样后，不做特征选择，就算不剪枝也不会出现过度拟合，最后结果由投票选出，得到最佳决策。决策树在节点分割的时候往往会出现过度拟合，决策树有时候对训练数据可以得到很低的错误率，但是运用到验证数据上时但出现较高错误率，所以需要剪枝和修正。

但是决策树的优点是简单，方便，对于小的数据量来说效果较好。从时间维度来说，由于随机森林需要多次构建树，导致时间成本比单独决策树高了很多，这算是随机森林的一个问题。另外随机森林如何选取样本分割数量和特征量比较重要。

Ada Boosting 将多个弱分类组合成一个强分类，通过加权重的方法，改善了分类器的分类正确率，同时也避免了过度拟合，通过{0: {'alpha': 0.24361662108864732}, 1: {'alpha': 0.3714040396090309}, 2: {'alpha': 0.5591424223557279}} 权重将错误改善到 0.01 以下，属于强分类器。但 Ada 对异常数据比较敏感，而且和训练的迭代次数有关，有时候并不一定能够完全满足最小错误率的阈值，需要分别调试得到，算法的可以理解性稍微复杂一些。

分类器/ 特征	过度拟合问题	执行效率	强分类器
DT	是	高	否
随机森林	很小	中	是
Ada boost	很小	中	是

思考和总结

根据上述结果和对三者分类器的理解，个人认为：

决策树的特点是它总是在沿着特征做切分，可以看出每个特征对决策的影响。如果我想快速理解一批数据特性，用决策树是个好方法。问题呢就是准确度相对低。

随机森林是一种集成算法，根据多个决策树来判断。数据准确度高，而且调参简单，大多数数据可以优先使用随机森林尝试下。

Adaboost算法通过组合弱分类器而得到强分类器，同时具有分类错误率上界随着训练增加而稳定下降，而且可以结合多种不同的分类器，算是能够在精确度上做到最好的分类器。

进一步优化

比如使用 graphviz 将结果可视化
训练更多特征的数据，对比结果