仿冒 APP 识别工具的设计与实现

程潇 2018111027 *
垢宇晴 2018111037 †
王皓 2018110990 ‡

January 9, 2019

1 研究背景与意义

第1章的主要内容是研究的背景与意义。

1.1 移动终端的飞速发展

目前，移动设备的使用频率已经超过了 PC 端，并且移动设备中储存了更多没有备份过的个人信息、甚至企业数据，一旦泄露，后果将无法弥补

1.2 仿冒 App 层出不穷

App 开发者由于缺少审核机制，恶意软件开发商可以轻易地发布一些仿冒的产品，因此山寨的应用层出不穷

1.3 课题目标

设计并实现一个检测假冒应用程序的工具

2 关键技术和实践难点

第2章的主要内容是关键技术和实践难点。

2.1 metadata 数据对比——短文本相似度

2.1.1 提取数据

选取 *metadata* 中的 *description_html* 字段。

2.1.2 文本预处理

1. 去掉所有带符号的词，如邮箱后缀、*hyphen* 连词、缩写等；
2. 去掉非英文的词汇；
3. 小写化；
4. 去长度小于 3 的单词，去掉数字和包含符号的单词；
5. 去除'the'、'about' 等停用词；
6. 进行词性标记, 标记每个词的词性；
7. 进行词形还原, 去掉单词的词缀，提取单词的主干部分；
8. 计算各个 *token* 的 $TFIDF$ 值, 即" 词频-逆文本频率"

2.1.3 训练模型

1. 基于预处理的文本，建立词向量，采用 $Skip-Gram$ 训练神经网络，优化收敛词向量；
2. 将训练好的模型和文本序列化至本地。

2.2 metadata 其他数据对比

2.2.1 字符串相似度

1. 选取 *metadata* 中的 *title*、*package_name*、*developer_email* 等字段；
2. 采用字符串的编辑距离度量相似度。

2.2.2 特征向量相似度

1. 选取 *metadata* 中的 *app_category*、*app_type*、*permission* 等字段；
2. 建立词汇表，取词汇表下标并进行归一化，将最后所得数值作为特征向量；
3. 计算特征向量间的余弦相似度。

2.3 *apk_icon* 对比——感知哈希算法

对每张图片生成一个" 指纹"（fingerprint）字符串，然后比较不同图片的指纹。结果越接近，就说明图片越相似。步骤如下：

*组长
†组员
‡组员

1. 第一步，缩小尺寸。将图片缩小到 8x8 的尺寸，总共 64 个像素。这一步的作用是去除图片的细节，只保留结构、明暗等基本信息，摒弃不同尺寸、比例带来的图片差异。

2. 第二步，简化色彩。将缩小后的图片，转为 64 级灰度。也就是说，所有像素点总共只有 64 种颜色。

3. 第三步，计算平均值。计算所有 64 个像素的灰度平均值。

4. 第四步，比较像素的灰度。将每个像素的灰度，与平均值进行比较。大于或等于平均值，记为 1；小于平均值，记为 0。

5. 第五步，计算哈希值。将上一步的比较结果，组合在一起，就构成了一个 64 位的整数，这就是这张图片的指纹。组合的次序并不重要，只要保证所有图片都采用同样次序就行了。

得到指纹以后，就可以对比不同的图片，看看 64 位中有多少位是不一样的。理论上，这等同于计算汉明距离（*Hammingdistance*）。如果不相同的数据位不超过 5，就说明两张图片很相似；如果大于 10，就说明这是两张不同的图片。

3 成果展示

3.1 metadata 数据对比

选取 *metadata* 中的 *description_html* 字段，对比如图。

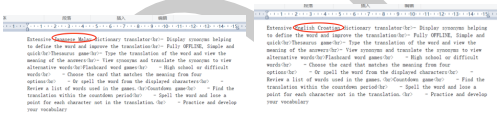

Figure 1: 相似度为 0.999999963 的两段文本对比

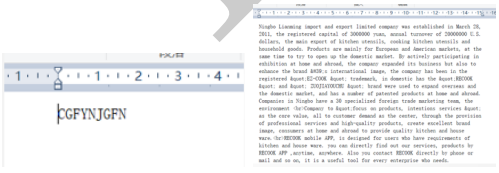

Figure 2: 相似度为 0.173356448 的两段文本对比