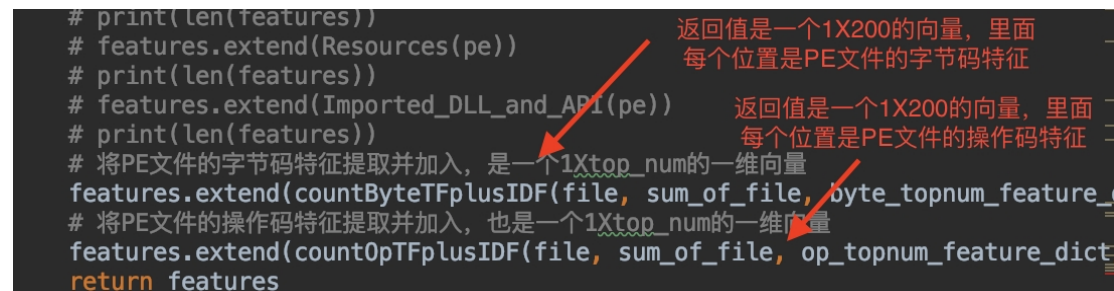


特征提取入口：

在原有的代码中添加了两个函数的调用。这两个函数均在 **extract.extract** 函数被调用，主要利用 **countByteTFplusIDF** 和 **countOpTFplusIDF** 这两个函数，二者分别提取单个 PE 文件的字节码和操作码特征。



```
# print(len(features))
# features.extend(Resources(pe))
# print(len(features))
# features.extend(Imported_DLL_and_API(pe))
# print(len(features))
# 将PE文件的字节码特征提取并加入，是一个1Xtop_num的一维向量
features.extend(countByteTFplusIDF(file, sum_of_file, byte_topnum_feature_dict))
# 将PE文件的操作码特征提取并加入，也是一个1Xtop_num的一维向量
features.extend(countOpTFplusIDF(file, sum_of_file, op_topnum_feature_dict))
return features
```

参数共有 5 个：

file——传入的单个 PE 训练样本。

sum_of_file——所有的样本文件总数（包括正常 PE 和恶意 PE），其主要是为了计算 n-grams 词频的时候使用到。

byte(op)_topnum_feature_dict——一个保存出现频率最高的字节码（操作码特征），形如{feature1:0, feature2:0, feature3:0, feature4:0 ……}。

注：之所以每个值为 0 是因为该字典只是为了告诉我们哪些特征是需要计算特征值的，countByteTFplusIDF 和 countOpTFplusIDF 这两个函数根据字典中的键，计算相应的特征的值，再填入字典，最后取出其中的值加入到一个列表中就得到了 1X200 的向量。

byte(op)_all_feature_dict——一个用于保存所有特征出现频率的字典，形如{feature1:6, feature2:3, feature3:5, ……}。

注：在 n-grams 的提取中需要就算所有出现过特征的频率。

N——n-grams 中的 n，也就是滑动窗口的大小，这里暂定为 4。

返回值：

feature——形如[0.123, 0.125, 0.312……]这样的 1X200 的列表。

几个函数用到的函数（字节码和操作码的提取方法一致，这里以字节码举例说明）：

1.countByteDF：用于计算所有文件中字节（操作）码出现的次数。

参数共 3 个：

paths——样本路径。

N——n-grams 中的 n，也就是滑动窗口的大小，这里暂定为 4。

top_num——要选取的出现最多的 200 个特征（有些特征在全部文件中就出现一两次，所以要筛选出现次数较多的特征才有意义）。

返回值：

topnum_feature_dict——形如{feature1:0, feature2:0, feature3:0, feature4:0 ……}, 它给出了哪些特征需要被计算。

all_feature_dict——形如{feature1:6, feature2:3, feature3:5, ……}, 它给出了所有文件中特征出现的次数。

逻辑：通过该函数，我们得到了所有文件中所有特征出现的次数，同时得到了哪些特征是要被提取的。

2.countByteTF：用于计算某个字节码在某个文件中出现的频率。

参数 2 个：

file——单个样本文件。

N——n-grams 中的 n，也就是滑动窗口的大小，这里暂定为 4。

返回值：

tf_dict——某个样本文件中的所有特征出现的频率。形如{feature1:0.234, feature2:0.342, ……}

逻辑：最后我们提取特征的时候，是需要计算给定的 top200 个出现次数最多的特征中的 TF 值和 IDF 乘积的，这里我们计算出单个文件中的特征的 TF，如果该特征也出现在 topnum_feature_dict（说明在全部文件中出现频率较高）中，那么就到 tf_dict 中将该特征的 TF 值取出放入 topnum_feature_dict 对应键的值中。

3.countByteIDF：用于计算出现某个特征的文件总数与所有文件总数的比值的 log 值。

即： $\log(D/d)$ ，D 为文件总数，d 为出现某个特征的文件总数。

参数 4 个：

pattern——出现的某个特征。

sum_of_file——文件总数，计算 log 值需要。

topnum_feature_dict——形如{feature1:0, feature2:0, feature3:0, feature4:0 ……}, 它给出了哪些特征需要被计算。

all_feature_dict——形如{feature1:6, feature2:3, feature3:5, ……}, 它给出了所有文件中特征出现的次数。

返回值：

$\text{math.log}((\text{sum_of_file}/d))$ ——某个特征的 IDF 值。

逻辑：首先要看一下当前滑动窗口扫到的特征在不在 `topnum_feature_dict` 中，只有在这里面我们才有继续算 IDF 值的意义。如果在我们去 `all_feature_dict` 中取出其在所有文件中出现的次数 `d`，通过 $\text{math.log}((\text{sum_of_file}/d))$ 计算得到该特征的 IDF 值。

4. `countByteTFplusIDF` 主要用于提取特征的函数，逻辑如下：

对于给定的样本文件，将其转化成 16 进制（相当于字符串），利用滑动窗口 4 个字符 4 个字符的截取，步长为 1，得到 4-grams 的特征。

该算法需要扫描文件两遍：

第一遍：

先利用 `countByteTF` 得到该文件中每个特征出现的频率，得到该文件的 `tf_dict`。

第二遍：

根据 `countByteDF` 所计算得到的 `all_feature_dict` 以及 `topnum_feature_dict`，先判断当前扫描到的特征是否出现在 `topnum_feature_dict` 中，如果出现，从 `tf_dict` 中取出该特征的 TF 值，同时利用 `countByteIDF` 计算该特征的 IDF 值，相乘放入 `topnum_feature_dict` 对应键的值中。最后会得到形如 `{feature1:0.321, feature2:0, feature3:0.414, ……}` 这样的字典，最后取出其中的值，到的列表 `[0.321, 0, 0.414, ……]`。大小为 1X200。