入侵检测与数字取证

大作业报告

57119101 王晨阳

实验目的

这三题均为对恶意域名做分类处理。第一题是多分类问题，第二题是二分类问题，这两者均为有监督的。而第三题为无监督学习。

Q1

本题公约20000个数据，其中有约1000个恶意域名。恶意域名中，有约500个有标签。由于非恶意域名没有标签，且数量远比恶意域名多，故我们可以在分类时将所有无标签的样本均作为非恶意域名，首先提取出剩余的约500个恶意域名。

特征提取如下：

- access & ip
    - 访问次数
    - 连续访问次数
    - 访问 IP 数
    - 访问国家数
    - 访问城市数
    - 访问 ISP 数
    - 按小时统计访问次数
    - 按日期统计访问次数
- flint
    - 解析次数
    - 按日期统计解析次数
- whois
    - 域名创建日期

    - 域名过期日期
    - 域名更新次数
    - 域名 DNS 服务器数
    - 域名管理员邮箱数
    - 域名注册国家数
    - 域名注册邮箱数
    - 域名注册省份数
    - 域名注册邮箱数
    - 域名 DNS 服务器列表数
    - 域名 DNS 服务器数
    - 域名注册商数
- fqdn
    - 字符个数
    - 数字个数
    - 普通字符个数
    - 特殊字符个数
    - 单词字母个数
    - 段落数

在上面的特征中，统计个数的可以使用 `set()` 来实现去重与计数。值得注意的是，以上有三个特殊的特征：

- 按小时统计访问次数
- 按日期统计访问次数
- 按日期统计解析次数

这三个特征均没有做进一步处理，即，一天有 24 个小时，则按小时统计访问次数就有 24 个特征；数据共有 91 天，则按日期统计访问次数就有 91 个特征。这样做的原因是，我们使用的是 xgboost，其对于不规则数据的鲁棒性极好。如果对数据做了进一步处理，反而会丢失部分特征，降低结果准确率。

另外，数据中有部分缺失、矛盾的情况。我们根据实际数据给其置 0 或 -1。

分类时，我们每次随机选择 1500 个无标签的数据，与有标签的数据组合作为训练集。我们在实验中发现，当选择 1500 个时，可以使得最终得到的恶意域名数量在 500 个左右，更加符合题意。

利用以上方法，我们在时间可以接受的情况下，做了尽量多次的预测。这里，我们耗时 30 min 做了 500 次预测。预测得到的结果取平均值后四舍五入，作为最终结果。

具体代码如下：

```
test_num_binary = []  # 恶意域名测试集编号
test_feature_binary = []  # 恶意域名测试集特征
test_label_binary = []  # 恶意域名测试集标签

train_random_binary = [] # 随机域名训练集编号

for i in range(tot_fqdn):
    if Label_all[i] == -1: # 无标签样本
```

```
        test_num_binary.append(i)
        test_feature_binary.append(Feature_all[i])

test_label_tmp = [0] * len(test_num_binary)

tot_round = 500

for itera in range(tot_round):
    train_feature_binary = []  # 恶意域名训练集特征
    train_label_binary = []  # 恶意域名训练集标签
    train_random_binary = random.sample(test_feature_binary, 1600)
    for i in range(tot_fqdn):
        if Label_all[i] != -1:
            train_feature_binary.append(Feature_all[i])
            train_label_binary.append(1)
    for i in range(len(train_random_binary)):
        train_feature_binary.append(train_random_binary[i])
        train_label_binary.append(0)

    params = { # 玄学调参
        'booster': 'gbtree',
        'objective': 'binary:logistic',
        'gamma': 0.1,
        'max_depth': 6,
        'lambda': 2,
        'subsample': 0.8,
        'colsample_bytree': 0.8,
        'min_child_weight': 1,
        'eta': 0.05,
        'eval_metric': 'logloss'
    }

    # 构造训练集
    dtrain = xgb.DMatrix(train_feature_binary, train_label_binary)
    num_rounds = 500
    # xgboost模型训练
    model = xgb.train(params, dtrain, num_rounds)

    # 对测试集进行预测
    dtest = xgb.DMatrix(test_feature_binary)
    test_label_binary_float = model.predict(dtest)

    for i in range(len(test_label_binary_float)):
        test_label_tmp[i] += round(test_label_binary_float[i])

for i in range(len(test_label_tmp)):
    test_label_tmp[i] = test_label_tmp[i] / tot_round
```

最终，我们得到了 542 个恶意域名。

在提取出恶意域名后，我们继续使用 xgboost 进行多分类。多分类时，我们使用 10-Fold Cross-Validation 进行调参优化。然而，遗憾的是，当参数变化时，交叉验证得到的结果几乎没有变化。最终，我们计算每一类的占比，并与训练集中每一类的占比作比较，使其差值尽量小。

具体的代码如下：

```
train_num = []
train_feature = []
train_label = []

test_label = []

for i in range(len(l_fqdn)):
    if Label_all[i] != -1: # 恶意样本
        train_num.append(i)
        train_feature.append(Feature_all[i])
        train_label.append(Label_all[i])

params = {  # 玄学调参
    'booster': 'gbtree',
    'objective': 'multi:softmax',
    'num_class': 9,
    'gamma': 0.0566,
    'max_depth': 4,
    'lambda': 1,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'min_child_weight': 2.6,
    'eta': 0.092
}

# 构造训练集
dtrain = xgb.DMatrix(train_feature, train_label)
num_rounds = 650
# xgboost模型训练
model = xgb.train(params, dtrain, num_rounds)

# 对测试集进行预测
dtest = xgb.DMatrix(test_feature)
test_label = model.predict(dtest)
```

最终，我们得到结果如下：

- 训练集总数： 476
- 测试集总数： 542
- 训练集类别数量： [336, 6, 18, 17, 24, 8, 57, 6, 4]

- 测试集类别数量： [391, 7, 15, 14, 14, 12, 83, 4, 2]
- 训练集类别占比： [0.706, 0.013, 0.038, 0.036, 0.05, 0.017, 0.12, 0.013, 0.008]
- 测试集类别占比： [0.721, 0.013, 0.028, 0.026, 0.026, 0.022, 0.153, 0.007, 0.004]

Q2

本题与第一题大同小异，我们依然使用 xgboost 进行分类。提取的特征如下：

- access
  - 访问次数
  - 连续访问次数
  - 访问 IP 数
  - 按小时统计访问次数
  - 按日期统计访问次数
- flint
  - 解析次数
  - 按日期统计解析次数
- fqdn
  - 字符个数
  - 数字个数
  - 普通字符个数
  - 特殊字符个数
  - 单词字母个数
  - 段落数

这里，我们没有使用 ip 数据。这是因为，ip 数据所能提供的特征较少，且缺失过多，使用后会适得其反。

训练与预测的具体代码如下：

```
params = { # 玄学调参
    'booster': 'gbtree',
    'objective': 'binary:logistic',
    'gamma': 0.1,
    'max_depth': 6,
    'lambda': 2,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'min_child_weight': 1,
    'eta': 0.05,
    'eval_metric': 'logloss'
}

# 构造训练集
dtrain = xgb.DMatrix(TRAIN_feature, TRAIN_label)
num_rounds = 5000
```

```
# xgboost模型训练
model = xgb.train(params, dtrain, num_rounds)

# 对测试集进行预测
dtest = xgb.DMatrix(TEST_feature)
TEST_label = model.predict(dtest)
```

最终，共找到 10192 个恶意域名。

Q3

本题为无监督学习，无法继续使用 xgboost，故我们直接使用了 kmeans。提取的特征如下：

- access
  - 访问次数
  - 连续访问次数
  - 访问 IP 数
  - 访问国家数
  - 访问城市数
  - 访问 ISP 数
  - 按小时统计访问次数
  - 按日期统计访问次数
- flint
  - 解析次数
  - 按日期统计解析次数
- fqdn
  - 字符个数
  - 数字个数
  - 普通字符个数
  - 特殊字符个数
  - 单词字母个数
  - 段落数

提取过特征后，我们首先对数据进行归一化。这里，由于有负值的存在，我们使用了最大最小值方法：

```
TRAIN_feature_normalization =  MinMaxScaler().fit_transform(TRAIN_feature)
```

然后，使用 kmeans 进行聚类：

```
label_pred = KMeans(n_clusters=5).fit_predict(TRAIN_feature_PCA)
```

最终得到的结果如下：

- 类别 0：8355 个
- 类别 1：4682 个
- 类别 2：3061 个
- 类别 3：1157 个
- 类别 4：213 个

可以看到，最多的一类有 8355 个，最少的一类有 213 个。这表明我们得到的分类结果是极不平衡的。我们尝试对其改进。

之前我们提到，如下三个特征

- 按小时统计访问次数
- 按日期统计访问次数
- 按日期统计解析次数

是没有做任何处理的。这在 xgboost 可以行得通，但在 kmeans 中，其对于距离计算的贡献过大，会严重影响结果。因此，我们分别对这三个特征计算了

- 最大值
- 最大值对应的下标
- 最小值
- 最小值对应的下标
- 中位数
- 标准差

我们再次做了训练，得到结果如下：

- 类别 0：5933 个
- 类别 1：5752 个
- 类别 2：2394 个
- 类别 3：2165 个
- 类别 4：1224 个

可以看到，每类样本的数量变得更加平衡。然而，最终的得分却从 11.46 降低到了 9.29。

实验结果

- Q1：84.90
- Q2：95.70
- Q3：11.46

实验心得

本次实验由于我单人成组，工作量较大，故力争使用最简单的方法得到最好的效果，并尽量在三题中实现代码复用。通过本次实验，我们进一步熟悉了机器学习的有关内容，并增加了特征提取的经验。