```
基于Tree-ensemble的KDD99入侵检测数据集的研究
```

0x01 简介

关于Tree-ensemble:指的是机器学习+集成学习：Tree指的是机器学习算法中的决策树。ensemble来自集成学习（Ensemble Learning），集成学习主要有两种bagging和boosting，常见的boosting方法有Adaboost、Gradient Boosting。我们所说的Tree-ensemble就是将集成学习和决策树结合构成的新算法，例如其中随机森林（RF）= bagging+决策树，梯度提升树（GBDT）=Gradient Boosting+决策树，这个系列我觉得包括决策树、随机森林、极端树、Adaboost、GBDT、XGBoost、Voting、Stacking（划分的不是很严格）

关于KDD99:KDD99是一项入侵检测领域的基准数据集，为基于AI的网络入侵检测研究奠定了基础。详情参考 http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html和http://www.kdd.org/kdd-cup/view/kdd-cup-1999。

我们所说的大数据安全=算法+数据+安全，数据驱动安全，算法赋能安全，而大数据安全应用无非是：安全场景-------->合适算法-------->安全系统-------->落地实现。而本文所说的是其中的关键一环“合适算法”，运用多种算法（tree-ensemble）对数据（kdd99）进行安全分析，评估算法性能，选取最优模型。为之后的落地实现打造核心引擎。

0x02 数据简介和分类任务

KDD99数据集分为训练集和测试集，用的是kddcup.data*10*percent.gz训练集，总共494021条记录，测试集用的是corrected.gz，总共311029条记录。KDD99数据集共有41个特征，5类大标签，分别为正常、dos、r2l，u2l、probe。我们尝试用决策树（Decision Tree）、随机森林（Random Forest）、极端树（extreme tree）、Adaboost、GBDT、XGBoost、Voting、Stacking等算法对KDD99数据集进行分类

0x03 特征工程

```
1、数据分析到合并稀疏特征
2、One-hot编码处理离散型特征
3、随机森林选择特征
4、标准化、归一化、正则化
```

具体来说：

1）数据分析到合并稀疏特征：做特征工程时，我们可能会遇到一个特征我们假设其特征列的符号值为v，其特征存在多种取值，标签label设为y，特征v如果有很多特征值对应标签y是相同的，那么这些v之间是没有意义的，我们称之为稀疏特征，这个时候可以进行合并稀疏特征，可以降低计算成本和错误分类的可能性。

尝试对特征“service”合并稀疏特征，发现ntp*u , urh*i , tftp*u , redi对应的标签都是“normal”，可以将它们四个合并，pmdump , http*2784 ,harvest , aol , http_8001对应的标签都是“satan”，可以将它们五个合并。

```
def merge_sparse_feature(df):
    df.loc[(df['service'] == 'ntp_u')
     | (df['service'] == 'urh_i')
     | (df['service'] == 'tftp_u')
    | (df['service'] == 'red_i')
    , 'service'] = 'normal_service_group'

    df.loc[(df['service'] == 'pm_dump')
     | (df['service'] == 'http_2784')
     | (df['service'] == 'harvest')
     | (df['service'] == 'aol')
     | (df['service'] == 'http_8001')
    , 'service'] = 'satan_service_group'
    return df
```

2）One-hot编码处理离散型特征：只有把离散型特征数字化，算法才可以处理，而假如某一特征有三种值“红”、“绿”、“蓝”，如果编码为“0”，“1”，“2”，那么“0”与“1”之间距离为1，“1”与“2”之间距离为1，但“0”与“2”之间距离为2，有大小的关系，但是原属性值“红绿蓝”是不区分大小的，这种编码显然是不对的。如果使用One-hot编码把“红”编码为“001”，把“绿”编码为“010”，把“蓝”编码为“100”，那么三者之间的距离都是一样的，满足原属性值“红绿蓝”不区分大小的性质。

对protocol*type , service ,flag特征进行One-hot编码，例如把特征protocol*type的三个值tcp、udp、icmp编码为。

```
def one_hot(df):
#print(df["service"])
service_one_hot = pd.get_dummies(df["service"])
df = df.drop('service', axis=1)
df = df.join(service_one_hot)
# test data has this column in service, clashes with protocol_type
# and not seen in training data, won't be learn by the model, safely delete
if 'icmp' in df.columns:
df = df.drop('icmp', axis=1)

protocol_type_one_hot = pd.get_dummies(df["protocol_type"])
df = df.drop('protocol_type', axis=1)
df = df.join(protocol_type_one_hot)

flag_type_one_hot = pd.get_dummies(df["flag"])
df = df.drop('flag', axis=1)
df = df.join(flag_type_one_hot)
return df
```

3）随机森林选择特征：在One-hot编码后，我们得到了115个特征，显然只有其中一部分特征是有显著作用的，这里我们选用随机森林来选择特征，由于随机森林的随机性，每次产生的结果不尽相同，观察大约有40‘ ’到45个显著特征，所以每次我们选取前50个特征，重复9次选取其中特征的交集，得到最终的特征列表，有44个特征。

```
selected_feat_names = set()
rfc = RandomForestClassifier(n_jobs=-1)
rfc.fit(X, y)
print("training finished")

importances = rfc.feature_importances_
indices = np.argsort(importances)[::-1]  # descending order
for f in range(X.shape[1]):
if f < 50:
selected_feat_names.add(X.columns[indices[f]])
print("%2d) %-*s %f" % (f + 1, 30, X.columns[indices[f]], importances[indices[f]]))
for i in range(9):
tmp = set()
rfc = RandomForestClassifier(n_jobs=-1)
rfc.fit(X, y)
print("training finished")
importances = rfc.feature_importances_
indices = np.argsort(importances)[::-1]  # descending order
for f in range(X.shape[1]):
if f < 50:  # need roughly more than 40 features according to experiments
tmp.add(X.columns[indices[f]])
print("%2d) %-*s %f" % (f + 1, 30, X.columns[indices[f]], importances[indices[f]]))

selected_feat_names &= tmp
print(len(selected_feat_names), "features are selected")
```

4）标准化、归一化、正则化：标准化是将数据转化为均值为0，方差为1的形式，将所有维度的变量一视同仁，归一化是把不同度量量纲化，但保留原始数据中由标准差反映的潜在权重关系，正则化是增加惩罚项，限制模型的复杂度。

这里因为用到的都是基于树的算法，所以不需要进行标准化、归一化、正则化

0x04 各种算法试验

```
1、决策树
2、随机森林
3、极端树
4、Adaboost
5、GBDT
6、XGBoost
7、Voting
8、Stacking
```

具体来说：决策树的核心代码如下：

```
df = pd.read_csv(r'data/train10pc', header=None, names=__ATTR_NAMES)
# sparse feature merge
df = processing.merge_sparse_feature(df)
# one hot encoding
df = processing.one_hot(df)
# y labels mapping
df = processing.map2major5(df)
with open(r'data/selected_feat_names.pkl', 'rb') as f:
selected_feat_names = pickle.load(f)
print("training data loaded")

y = df["attack_type"].values
X = df[selected_feat_names].values
# TODO: put the best paras learn from grid search
dt = DecisionTreeClassifier(random_state=0)
dt = dt.fit(X, y)
print("training finished")

df=pd.read_csv(r'data/corrected',header=None, names=__ATTR_NAMES)
df = processing.merge_sparse_feature(df)
# one hot encoding
df = processing.one_hot(df)
# y labels mapping
df = processing.map2major5(df)
with open(r'data/selected_feat_names.pkl', 'rb') as f:
selected_feat_names = pickle.load(f)
print("test data loaded")

X = df[selected_feat_names].values
y = df['attack_type'].values
y_rf = dt.predict(X)

print("dt results:")
cost_based_scoring.score(y, y_rf, True)
```

1）决策树实验结果：
2）随机森林实验结果：3）极端树实验结果：4）Adaboost实验结果：5）GBDT实验结果：6）XGBoost实验结果：7）Voting实验结果：
8）Stacking实验结果：

0x05 调参

关键词：GridSearch（百度）

核心代码如下：

```
# optimize params
y = df["attack_type"].values
X = df[selected_feat_names].values
rfc = RandomForestClassifier(n_jobs=-1)
parameters = {
#'n_estimators': tuple(range(10, 50, 10)),  # overfit if too large, underfit if too small
'n_estimators': [25, 30, 35],
'criterion': ("gini", "entropy")
}
scorer = cost_based_scoring.scorer(show=True)
gscv = GridSearchCV(rfc, parameters,scoring=scorer,cv=3,verbose=2,refit=False,n_jobs=1,return_train_score=False)
gscv.fit(X, y)
print("optimization params:",gscv.best_params_['n_estimators'],gscv.best_params_['criterion'])
print("grid search finished")
#35,entropy
```

0x06 评价结果数据

算法的准确率、召回率、score分数总结如图：▫单看召回率：▫单看准确率：▫其中图中黑体数字表示优于Winning对应的数据（winning是kdd99竞赛的第一名的分类结果）

结论：

对算法模型来说：

1、随机森林、极端树、GBDT、XGBoost是最好的算法，尽管和最优的有点差距;

2、随机森林和极端树的结果很相似，因为他们算法也相似;

3、Adaboost很垃圾,不知道是不是调参原因

4、GBDT和XGBoost结果很相似，因为算法很相似;

5、决策树、极端树、GBDT、XGBoost、Voting和Stacking在类2上的召回率良好，准确率也还可以;

6、GBDT、XGBoost在类0、1、2上综合表现还好;

7、Tree-ensemble之类的算法较逻辑回归等其他算法来说，通用型强，性能很好;

对数据来说：

1、在损失准确率的情况下类0召回率还好;

2、在损失召回率的情况下类1的准确率高;

3、类2处理的很好;

4、类3和类4结果不好;

参考

http://cseweb.ucsd.edu/%7Eelkan/clresults.html http://cseweb.ucsd.edu/~elkan/mp13method.html https://github.com/hengji-liu/kdd99 https://github.com/hengji-liu/kdd99/blob/master/report.pdf http://www.cnblogs.com/jasonfreak/p/5720137.html