In [1]:
```
import numpy as np
import pandas as pd
import os
from matplotlib import pyplot as plt
import seaborn as sns
```

由于CSV文件内数据超过4千万笔，Pandas DataFrame无法一次读取这么大的数据量，因此将CSV文件内的数据每100万笔读取一次，然后从这100万笔中随机抽取5%的数据，形成一个新的训练集

In [2]:
```
chunksize = 10 ** 6
num_of_chunk = 0
train = pd.DataFrame()
for chunk in pd.read_csv('train.csv', chunksize=chunksize):
    num_of_chunk += 1
    train = pd.concat([train, chunk.sample(frac=.05, replace=False, random_state=123)], axis=0)
    print('Processing Chunk No. ' + str(num_of_chunk))
train.reset_index(inplace=True)
# 备份原始训练集的数据长度，方便后续重新分割索引
train_len = len(train)
train_len
```

```
Processing Chunk No. 1
Processing Chunk No. 2
Processing Chunk No. 3
Processing Chunk No. 4
Processing Chunk No. 5
Processing Chunk No. 6
Processing Chunk No. 7
Processing Chunk No. 8
Processing Chunk No. 9
Processing Chunk No. 10
Processing Chunk No. 11
Processing Chunk No. 12
Processing Chunk No. 13
Processing Chunk No. 14
Processing Chunk No. 15
Processing Chunk No. 16
Processing Chunk No. 17
Processing Chunk No. 18
Processing Chunk No. 19
Processing Chunk No. 20
Processing Chunk No. 21
Processing Chunk No. 22
Processing Chunk No. 23
Processing Chunk No. 24
Processing Chunk No. 25
Processing Chunk No. 26
Processing Chunk No. 27
Processing Chunk No. 28
Processing Chunk No. 29
Processing Chunk No. 30
Processing Chunk No. 31
Processing Chunk No. 32
Processing Chunk No. 33
Processing Chunk No. 34
Processing Chunk No. 35
Processing Chunk No. 36
Processing Chunk No. 37
Processing Chunk No. 38
Processing Chunk No. 39
Processing Chunk No. 40
Processing Chunk No. 41
```

Out[2]: 2021448

将test测试集的资料读出后，把train训练集和 test测试集合并成新的数据集df，方便同时对资料集进行数据预处理。

In [3]:
```
test_file = 'test.gz'
df = pd.concat([train, pd.read_csv(test_file, compression='gzip')]).drop(['index', 'id'], axis=1)
```

本次预测的目的为用10天的点击情况作为训练模型，用来预测第11天的点击情况，因此年月日期没有参考价值，但是weekday有参考价值。所以把原始数据中hour特征中的日期，转化为weekday。 每天的小时段根据每人不同的生活习惯，有参考价值。为避免24小时的纬度导致数据崩溃，采用十二地支计时法将小时的区间分为12个时辰，每两小時一个时辰。

In [4]:
```python
# 定义一个将hour数据转换为日期格式的函数
def get_date(hour):
    y = '20'+str(hour)[:2]
    m = str(hour)[2:4]
    d = str(hour)[4:6]
    return y+'-'+m+'-'+d

# 创建weekday列，将hour数据转换后填入
df['weekday'] = pd.to_datetime(df.hour.apply(get_date)).dt.dayofweek.astype(str)

# 定义一个将hour数据转换为12时辰的函数
def tran_hour(x):
    x = x % 100
    while x in [23,0]:
        return '23-01'
    while x in [1,2]:
        return '01-03'
    while x in [3,4]:
        return '03-05'
    while x in [5,6]:
        return '05-07'
    while x in [7,8]:
        return '07-09'
    while x in [9,10]:
        return '09-11'
    while x in [11,12]:
        return '11-13'
    while x in [13,14]:
        return '13-15'
    while x in [15,16]:
        return '15-17'
    while x in [17,18]:
        return '17-19'
    while x in [19,20]:
        return '19-21'
    while x in [21,22]:
        return '21-23'

# 将hour列的数据转换为时段
df['hour'] = df.hour.apply(tran_hour)
```

In [5]:
```python
# 查看数据集的的简要摘要
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6598912 entries, 0 to 4577463
Data columns (total 24 columns):
 #   Column            Dtype  
---  ------            -----  
 0   click             float64
 1   hour              object 
 2   C1                int64  
 3   banner_pos        int64  
 4   site_id           object 
 5   site_domain       object 
 6   site_category     object 
 7   app_id            object 
 8   app_domain        object 
 9   app_category      object 
 10  device_id         object 
 11  device_ip         object 
 12  device_model      object 
 13  device_type       int64  
 14  device_conn_type  int64  
 15  C14               int64  
 16  C15               int64  
 17  C16               int64  
 18  C17               int64  
 19  C18               int64  
 20  C19               int64  
 21  C20               int64  
 22  C21               int64  
 23  weekday           object 
dtypes: float64(1), int64(12), object(11)
memory usage: 1.2+ GB
```

确认每个特征的计数值，int类别的特征，最多包含4333个特征值，在这个新生成的600万笔数据的资料集，既然不是连续计数，据此判定本资料集的所有特征均为Object类型的数值。

In [6]:
```python
len_of_feature_count = []
for i in df.columns[2:23].tolist():
    print(i, ':', len(df[i].astype(str).value_counts()))
    len_of_feature_count.append(len(df[i].astype(str).value_counts()))
```

```
C1 : 7
banner_pos : 7
site_id : 3496
site_domain : 4333
site_category : 24
app_id : 5466
app_domain : 319
app_category : 31
device_id : 552856
device_ip : 1875405
device_model : 6321
device_type : 5
device_conn_type : 4
C14 : 2662
C15 : 8
C16 : 9
C17 : 470
C18 : 4
C19 : 68
C20 : 169
C21 : 62
```

In [7]:
```python
# 创建一个list，将需要转换的特征名称存入该list
need_tran_feature = df.columns[2:4].tolist() + df.columns[13:23].tolist()

# 依次将非object类型的数据转换为object类型
for i in need_tran_feature:
    df[i] = df[i].astype(str)
```

类似device_ip的计数值达到数百万，这种情况下强行进行one-hot编码，很可能造成数据崩溃。为避免出现上述情况，将每个特征的计数值以10为限，一旦超过10这个值，将进行缩减操作。

缩减操作的方式为计算某特征的所有值的点击率，依据点击频率分为very_high, higher, mid, lower, very_low这5个区间

In [8]:
```python
obj_features = []

for i in range(len(len_of_feature_count)):
    if len_of_feature_count[i] > 10:
        obj_features.append(df.columns[2:23].tolist()[i])
obj_features
```

Out[8]:
```
['site_id',
 'site_domain',
 'site_category',
 'app_id',
 'app_domain',
 'app_category',
 'device_id',
 'device_ip',
 'device_model',
 'C14',
 'C17',
 'C19',
 'C20',
 'C21']
```

In [9]:

```
df_describe = df.describe()
df_describe
```

Out[9]:

	click
count	2.021448e+06
mean	1.700731e-01
std	3.756971e-01
min	0.000000e+00
25%	0.000000e+00
50%	0.000000e+00
75%	0.000000e+00
max	1.000000e+00

In [10]:

```python
def obj_clean(X):
    # 定义一个缩减操作的函数，每次处理一个特征

    def get_click_rate(x):
        # 定义一个取得点击率的函数
        temp = train[train[X.columns[0]] == x]
        res = round((temp.click.sum() / temp.click.count()),3)
        return res

    def get_type(V, str):
        # 定义一个取得新特征值区间差距的函数
        very_high = df_describe.loc['mean','click'] + 0.04
        higher = df_describe.loc['mean','click'] + 0.02
        lower = df_describe.loc['mean','click'] - 0.02
        very_low = df_describe.loc['mean','click'] - 0.04

        vh_type = V[V[str] > very_high].index.tolist()
        hr_type = V[(V[str] > higher) & (V[str] < very_high)].index.tolist()
        vl_type = V[V[str] < very_low].index.tolist()
        lr_type = V[(V[str] < lower) & (V[str] > very_low)].index.tolist()

        return vh_type, hr_type, vl_type, lr_type

    def clean_function(x):
        # 定义一个根据区间差距转换资料值得函数
        # 判断依据：总平均点击率的正负 4% 为very_high(low), 总平均点击率的正负 2%为higher (lower)
        while x in type_[0]:
            return 'very_high'
        while x in type_[1]:
            return 'higher'
        while x in type_[2]:
            return 'very_low'
        while x in type_[3]:
            return 'lower'
        return 'mid'

    print('Run: ', X.columns[0])
    fq = X[X.columns[0]].value_counts()
    # 建立一个暂存的资料值频率列表
    # 理论上，将全部的资料值都进行分类转换可得到最佳效果；本次为了节约运算时间，将舍弃频率低于排名前1000 row以后的资料值。
    if len(fq) > 1000:
        fq = fq[:1000]

    # 将频率列表转换为dataframe, 并将索引值index填入一個新的列。
    fq = pd.DataFrame(fq)
    fq['new_column'] = fq.index

    # 使用index调用get_click_rate function, 取得每个资料值的点击率
    fq['click_rate'] = fq.new_column.apply(get_click_rate)

    # 使用 get_type function取得区间差距，并存储为一个list, 以便提供给下一个clean_function使用
    type_ = get_type(fq, 'click_rate')

    # 使用 clean_funtion funtion, 回传转换后的特征值
    return X[X.columns[0]].apply(clean_function)

# 使用for 循环将需要转换的特征输入到 obj_clean function
for i in obj_features:
    df[[i]] = obj_clean(df[[i]])

df
```

```
Run:  site_id
Run:  site_domain
Run:  site_category
Run:  app_id
Run:  app_domain
Run:  app_category
Run:  device_id
Run:  device_ip
Run:  device_model
Run:  C14

/opt/conda/lib/python3.7/site-packages/pandas/core/ops/array_ops.py:253: FutureWarning: elementwise comparison failed; re
turning scalar instead, but in the future will perform elementwise comparison
  res_values = method(rvalues)

Run:  C17
Run:  C19
Run:  C20
Run:  C21
```

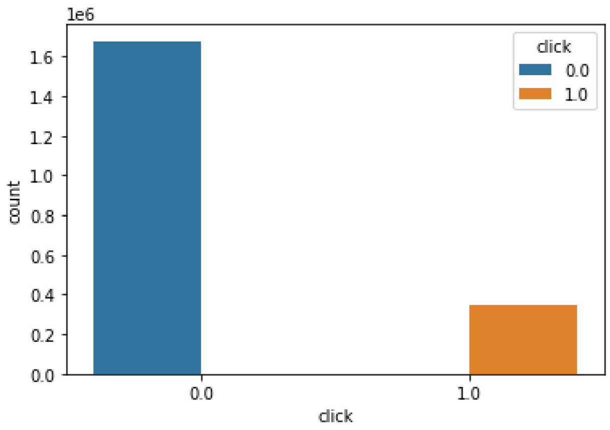
Out[10]:

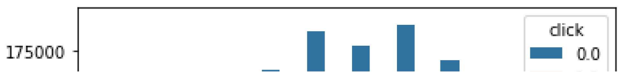
	click	hour	C1	banner_pos	site_id	site_domain	site_category	app_id	app_domain	app_category	...	device_conn
0	1.0	01-03	1005	1	very_high	very_high	mid	higher	higher	higher	...	
1	0.0	01-03	1005	0	higher	higher	higher	higher	higher	higher	...	
2	0.0	01-03	1005	0	higher	higher	higher	higher	higher	higher	...	
3	0.0	03-05	1005	1	very_low	very_low	mid	higher	higher	higher	...	
4	0.0	01-03	1005	0	higher	higher	higher	higher	higher	higher	...	
...	...	...	...	...	...	...	...	...	...	...	...	
4577459	NaN	23-01	1005	0	very_high	very_high	very_high	higher	higher	higher	...	
4577460	NaN	23-01	1005	0	mid	higher	mid	higher	higher	higher	...	
4577461	NaN	23-01	1005	0	very_high	very_high	very_high	higher	higher	higher	...	
4577462	NaN	23-01	1005	0	very_low	very_low	very_low	very_low	very_low	very_low	...	
4577463	NaN	23-01	1005	0	mid	higher	mid	higher	higher	higher	...	

6598912 rows × 24 columns

In [11]:

```
# 确认所有特征的分布情况
for i in df.columns:
    sns.countplot(x = i, hue = "click", data = df)
    plt.show()
```





根据上面显示的图表来看，['device_id', 'C14', 'C17', 'C19', 'C20', 'C21'] 这些特征仅有一个值，对预测模型没有价值，因此将这些列移除数据集。

In [12]:

```
df.drop(['device_id', 'C14', 'C17', 'C19', 'C20', 'C21'], axis=1, inplace=True)
```

In [13]:

```
# 对所有特征进行 one-hot 编码
df = pd.get_dummies(df)

# 根据处理过的df资料集，重新將train训练集, test测试集分割出來
train = df[:train_len]
test = df[train_len:]
```

In [16]:

```
del df
```

由于资料集的数据量非常的庞大，同时正向label的占比仅占全部资料的17%左右，比例明显失衡，需要用强化加权功能的演算法，因此决定使用xgboost演算法，解决强化权重问题，同时运用GPU节省运算时间

为节省调参时间，在开始运行预测模型之前，优先建立100株决策树，用grid search寻找最佳参数与重要特征，最后用xgboost算法简历模型。为了缩短建立决策时的时间，将从负向label的资料集中抽取样本，与所有正向label的资料构成一个各占50%的资料集，来平衡权重问题，因为正反label比例平衡，用ROC_AUC的分数进行调参

In [17]:

```
# 从train训练集中，标签为0的资料中，随机抽取与标签为1一样多的数量，并将其结合成正反标签各占50%的资料集
pre_X = train[train['click'] == 0].sample(n=len(train[train['click'] == 1]), random_state=111)
pre_X = pd.concat([pre_X, train[train['click'] == 1]]).sample(frac=1)
pre_y = pre_X[['click']]
pre_X.drop(['click'], axis=1, inplace=True)
test.drop(['click'], axis=1, inplace=True)
```

In [18]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import roc_auc_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split

# 将新的资料集分割为训练集和验证集
pre_X_train, pre_X_test, pre_y_train, pre_y_test = train_test_split(pre_X, pre_y, test_size=0.20, stratify=pre_y, random_s
```

In [19]:

```
# 进行Grid Search調參，建立100棵决策树來取得最佳参数
params = {"criterion":["gini", "entropy"], "max_depth":range(1,20)}
grid_search = GridSearchCV(DecisionTreeClassifier(), param_grid=params, scoring='roc_auc', cv=100, verbose=1, n_jobs=-1)
grid_search.fit(pre_X_train, pre_y_train)
grid_search.best_score_, grid_search.best_estimator_, grid_search.best_params_
```

```
Fitting 100 folds for each of 38 candidates, totalling 3800 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:   29.9s
[Parallel(n_jobs=-1)]: Done 196 tasks      | elapsed:  2.7min
[Parallel(n_jobs=-1)]: Done 446 tasks      | elapsed:  9.8min
[Parallel(n_jobs=-1)]: Done 1246 tasks      | elapsed: 55.5min
[Parallel(n_jobs=-1)]: Done 1796 tasks      | elapsed: 99.4min
[Parallel(n_jobs=-1)]: Done 2446 tasks      | elapsed: 122.5min
[Parallel(n_jobs=-1)]: Done 3196 tasks      | elapsed: 169.4min
[Parallel(n_jobs=-1)]: Done 3800 out of 3800 | elapsed: 220.0min finished
```

Out[19]:

```
(0.7315752436075236,
 DecisionTreeClassifier(criterion='entropy', max_depth=11),
 {'criterion': 'entropy', 'max_depth': 11})
```

In [20]:

```
# 根据Grid Search的结果建立一个决策树模型
tree = grid_search.best_estimator_
tree.fit(pre_X,pre_y)

# 输出重要特征，并依据特征的重要性排序
feature_importances = pd.DataFrame(tree.feature_importances_)
feature_importances.index = pre_X_train.columns
feature_importances = feature_importances.sort_values(0,ascending=False)
feature_importances
```

Out[20]:

	0
site_id_very_low	0.407132
app_id_very_high	0.167188
site_id_very_high	0.075666
app_id_very_low	0.063975
app_category_higher	0.063861
...	...
C16_320	0.000000
C1_1008	0.000000
C16_20	0.000000
C16_1024	0.000000
C15_1024	0.000000

103 rows × 1 columns

In [21]:

```
# 调整前置的训练集与验证集，将特征的重要性缩减为重要排名的三分之一
pre_X_train = pre_X_train[feature_importances.index[:int(len(feature_importances)/3)]]
pre_X_test = pre_X_test[feature_importances.index[:int(len(feature_importances)/3)]]
```

In [22]:

```
# 使用33%的重要特特征重新进行Grid Search调参
params = {"criterion":["gini", "entropy"], "max_depth":range(1,12)}
grid_search = GridSearchCV(DecisionTreeClassifier(), param_grid=params, scoring='roc_auc', cv=100, verbose=1, n_jobs=-1)
grid_search.fit(pre_X_train, pre_y_train)
grid_search.best_score_, grid_search.best_estimator_, grid_search.best_params_
```

```
Fitting 100 folds for each of 22 candidates, totalling 2200 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  46 tasks      | elapsed:   12.5s
[Parallel(n_jobs=-1)]: Done 196 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done 446 tasks      | elapsed:  4.0min
[Parallel(n_jobs=-1)]: Done 796 tasks      | elapsed: 10.2min
[Parallel(n_jobs=-1)]: Done 1246 tasks      | elapsed: 18.0min
[Parallel(n_jobs=-1)]: Done 1796 tasks      | elapsed: 25.3min
[Parallel(n_jobs=-1)]: Done 2200 out of 2200 | elapsed: 34.4min finished
```

Out[22]:

```
(0.7313727472187124,
 DecisionTreeClassifier(criterion='entropy', max_depth=11),
 {'criterion': 'entropy', 'max_depth': 11})
```

In [23]:
```python
# 调整前置的训练集与验证集，将特征的重要性缩减为重要排名的三分之一
pre_X = pre_X[feature_importances.index[:int(len(feature_importances)/3)]]

# 根据Grid Search的结果建立一个决策树模型
tree = grid_search.best_estimator_
tree.fit(pre_X,pre_y)

# 输出重要特征，并依据特征的重要性排序
feature_importances = pd.DataFrame(tree.feature_importances_)
feature_importances.index = pre_X_train.columns
feature_importances = feature_importances.sort_values(0,ascending=False)
feature_importances
```

Out[23]:

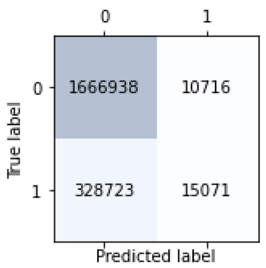
	0
site_id_very_low	0.411739
app_id_very_high	0.169080
site_id_very_high	0.076523
app_category_higher	0.065269
app_id_very_low	0.064699
C16_250	0.030797
C18_1	0.023919
device_model_very_high	0.019606
device_model_very_low	0.014941
device_model_lower	0.013434
site_domain_very_low	0.012942
site_category_higher	0.010990
device_conn_type_0	0.007462
banner_pos_0	0.007129
site_id_higher	0.007018
hour_19-21	0.005663
C18_2	0.005435
C16_50	0.005369
hour_17-19	0.004964
C18_3	0.004618
device_ip_very_low	0.003406
weekday_2	0.003306
app_domain_lower	0.003166
hour_21-23	0.003113
weekday_3	0.003100
device_ip_very_high	0.003092
device_ip_mid	0.003073
weekday_1	0.002881
app_id_lower	0.002690
banner_pos_1	0.002651
C18_0	0.002597
site_category_very_low	0.002206
app_domain_very_high	0.002108
C15_216	0.001016

In [24]:
```python
# 最终预测模型之特征，将采用特征值0.005以上的特征
feature_len = len(feature_importances[feature_importances[feature_importances.columns[0]] > 0.005])

# 调整最终完整的测试集与验证集的特征
y = train[['click']]
X = train[feature_importances[:feature_len].index]
test = test[feature_importances[:feature_len].index]
```

In [25]:

```python
from xgboost import XGBClassifier

# 使用xgboost 建模，并指定先前调参得到的节点深度使用xgboost 建模，並指定先前調參得到的節點深度限制
model = XGBClassifier(tree_method = 'gpu_hist', n_jobs=-1, n_estimators=500, max_depth=11)
model.fit(X,y.values.ravel())
y_pred = model.predict(X)
print("Roc_auc_score: ",roc_auc_score(y,y_pred)*100,"%")

# 输出混淆矩阵，查看预测结果
confmat = confusion_matrix(y_true=y, y_pred=y_pred, labels=[0, 1])

fig, ax = plt.subplots(figsize=(2.5, 2.5))
ax.matshow(confmat, cmap=plt.cm.Blues, alpha=0.3)
for i in range(confmat.shape[0]):
    for j in range(confmat.shape[1]):
        ax.text(x=j, y=i, s=confmat[i, j], va='center', ha='center')

plt.xlabel('Predicted label')
plt.ylabel('True label')

plt.tight_layout()
plt.show()

# 输出submission
submission = pd.read_csv(samplesubmision_file, compression='gzip', index_col='id')
submission[submission.columns[0]] = model.predict_proba(test)[:,1]
submission.to_csv('submission.csv')
```

```
Roc_auc_score:  51.87249034065638 %
```




```
/opt/conda/lib/python3.7/site-packages/numpy/lib/arraysetops.py:569: FutureWarning: elementwise comparison failed; return
ing scalar instead, but in the future will perform elementwise comparison
  mask |= (ar1 == a)
```