In [1]:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
import warnings
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn import svm
from sklearn.exceptions import DataConversionWarning
warnings.filterwarnings(action='ignore', category=DataConversionWarning)
warnings.filterwarnings("ignore")
import seaborn as sns
from sklearn.metrics import roc_auc_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
import numpy as np
import warnings
warnings.simplefilter("ignore", UserWarning)
```

In [2]:

```
ls
```

```
Archive.zip                          __pycache__/
__MACOSX/                            temp_model.py
nohup.out                            Untitled.ipynb
orange_small_train_appetency.labels  working (2).ipynb
orange_small_train_churn.labels      working_apr.ipynb
orange_small_train .data             working_aprv2.ipynb
orange_small_train_upselling.labels
```

In [3]:

```
data = pd.read_table('orange_small_train .data')
churn = pd.read_table('orange_small_train_churn.labels')
appetency = pd.read_table('orange_small_train_appetency.labels')
upselling = pd.read_table('orange_small_train_upselling.labels')
```

In [4]:

```
churn = churn.replace(-1,0)
appetency = appetency.replace(-1,0)
upselling = upselling.replace(-1,0)
```

In [5]:

```
from sklearn.utils import class_weight
c_weight = class_weight.compute_class_weight('balanced',np.unique(churn['Churn']),ch
churn_dict = dict(zip(np.unique(churn['Churn']), c_weight))
a_weight = class_weight.compute_class_weight('balanced',np.unique(appetency['Appeten
appetency_dict = dict(zip(np.unique(appetency['Appetency']), a_weight))
u_weight = class_weight.compute_class_weight('balanced',np.unique(upselling['Upselli
upselling_dict = dict(zip(np.unique(upselling['Upselling']), u_weight))
```

In [6]:

```
data.describe()
```

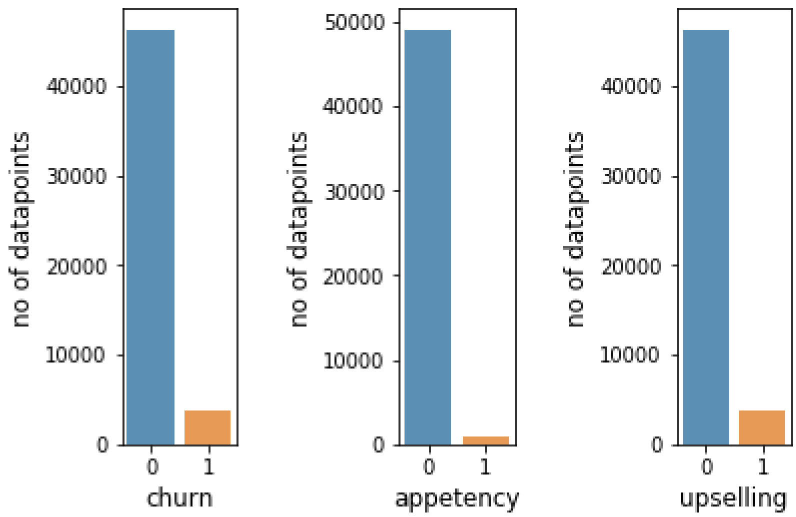
Out[6]:

| | Var1 | Var2 | Var3 | Var4 | Var5 | Var6 | |
|---|---|---|---|---|---|---|---|
| **count** | 702.000000 | 1241.000000 | 1240.000000 | 1579.000000 | 1.487000e+03 | 44471.000000 | 4446 |
| **mean** | 11.487179 | 0.004029 | 425.298387 | 0.125396 | 2.387933e+05 | 1326.437116 | |
| **std** | 40.709951 | 0.141933 | 4270.193518 | 1.275481 | 6.441259e+05 | 2685.693668 | |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000e+00 | 0.000000 | |
| **25%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000e+00 | 518.000000 | |
| **50%** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000e+00 | 861.000000 | |
| **75%** | 16.000000 | 0.000000 | 0.000000 | 0.000000 | 1.187425e+05 | 1428.000000 | |
| **max** | 680.000000 | 5.000000 | 130668.000000 | 27.000000 | 6.048550e+06 | 131761.000000 | 14 |

8 rows × 192 columns

In [7]:

```
count_churn = churn['Churn'].value_counts()
count_appetency = appetency['Appetency'].value_counts()
count_upselling = upselling['Upselling'].value_counts()

plt.subplot(151)
sns.barplot(count_churn.index, count_churn.values, alpha=0.8)
plt.ylabel('no of datapoints', fontsize=12)
plt.xlabel('churn', fontsize=12)

plt.subplot(153)
sns.barplot(count_appetency.index, count_appetency.values, alpha=0.8)
plt.ylabel('no of datapoints', fontsize=12)
plt.xlabel('appetency', fontsize=12)

plt.subplot(155)
sns.barplot(count_upselling.index, count_upselling.values, alpha=0.8)
plt.ylabel('no of datapoints', fontsize=12)
plt.xlabel('upselling', fontsize=12)
plt.show()
```




for all 3 task, data is highly imbalanced

**removing constant variables**

In [8]:

```
data = data.loc[:,data.apply(pd.Series.nunique) != 1]
```

In [9]:

```
data.shape
```

Out[9]:

```
(50000, 225)
```

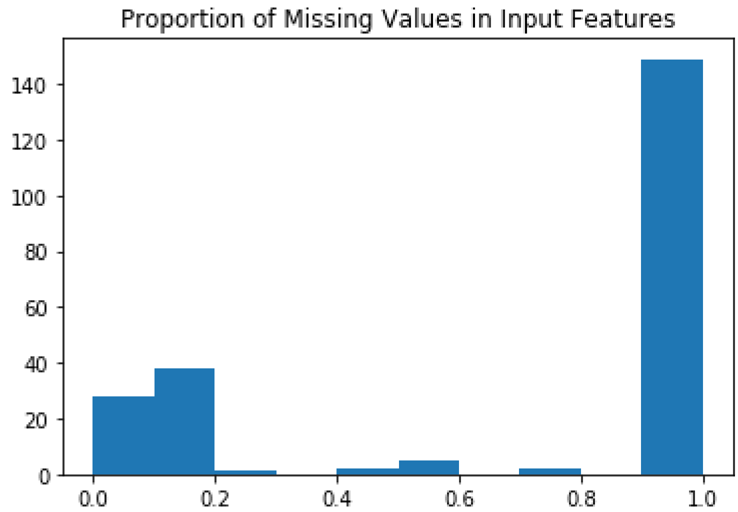
***checking missing datapoints feature wise}***

In [10]:

```
no_data_sample = len(data.index)

input_features_missing_proportions = data.isnull().sum() / no_data_sample

plt.hist(input_features_missing_proportions)
plt.title('Proportion of Missing Values in Input Features')
```

Out[10]:

```
Text(0.5, 1.0, 'Proportion of Missing Values in Input Features')
```




In [11]:

```
data = data.loc[:, data.isnull().mean() <= .8]
```

In [12]:

```
data.shape
```

Out[12]:

```
(50000, 76)
```

for all 3 task, data is highly imbalanced

***checking missing datapoints feature wise}***

In [13]:

```
numeric_col = data._get_numeric_data().columns
numeric_col
```

Out[13]:

```
Index(['Var6', 'Var7', 'Var13', 'Var21', 'Var22', 'Var24', 'Var25', 'V
ar28',
       'Var35', 'Var38', 'Var44', 'Var57', 'Var65', 'Var72', 'Var73',
'Var74',
       'Var76', 'Var78', 'Var81', 'Var83', 'Var85', 'Var94', 'Var109',
       'Var112', 'Var113', 'Var119', 'Var123', 'Var125', 'Var126', 'Va
r132',
       'Var133', 'Var134', 'Var140', 'Var143', 'Var144', 'Var149', 'Va
r153',
       'Var160', 'Var163', 'Var173', 'Var181', 'Var189'],
      dtype='object')
```

In [14]:

```
#replacing null missing value with mean of dataset
numeric_column = [i for i in numeric_col]
for i in numeric_column:
    data[i]=data[i].fillna(data[i].median())
```

In [15]:

```
data_num =data[numeric_column]
data_num.head()
```

Out[15]:

| | Var6 | Var7 | Var13 | Var21 | Var22 | Var24 | Var25 | Var28 | Var35 | Var38 | ... | Var140 | Var |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1526.0 | 7.0 | 184.0 | 464.0 | 580.0 | 14.0 | 128.0 | 166.56 | 0.0 | 3570.0 | ... | 185.0 | |
| **1** | 525.0 | 0.0 | 0.0 | 168.0 | 210.0 | 2.0 | 24.0 | 353.52 | 0.0 | 4764966.0 | ... | 0.0 | |
| **2** | 5236.0 | 7.0 | 904.0 | 1212.0 | 1515.0 | 26.0 | 816.0 | 220.08 | 0.0 | 5883894.0 | ... | 800.0 | |
| **3** | 861.0 | 0.0 | 0.0 | 144.0 | 0.0 | 2.0 | 0.0 | 22.08 | 0.0 | 0.0 | ... | 0.0 | |
| **4** | 1029.0 | 7.0 | 3216.0 | 64.0 | 80.0 | 4.0 | 64.0 | 200.00 | 0.0 | 0.0 | ... | 3255.0 | |

5 rows × 42 columns

In [16]:

```
data_num.describe()
```

Out[16]:

| | Var6 | Var7 | Var13 | Var21 | Var22 | Var24 |
|---|---|---|---|---|---|---|
| **count** | 50000.000000 | 50000.000000 | 50000.000000 | 50000.000000 | 50000.000000 | 50000.000000 |
| **mean** | 1274.969080 | 6.830600 | 1136.948880 | 224.508720 | 279.201000 | 4.145280 |
| **std** | 2537.052042 | 5.965663 | 2654.880938 | 534.128741 | 669.089558 | 9.225198 |
| **min** | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 581.000000 | 0.000000 | 4.000000 | 120.000000 | 150.000000 | 0.000000 |
| **50%** | 861.000000 | 7.000000 | 232.000000 | 144.000000 | 180.000000 | 2.000000 |
| **75%** | 1316.000000 | 7.000000 | 1264.000000 | 212.000000 | 265.000000 | 4.000000 |
| **max** | 131761.000000 | 140.000000 | 197872.000000 | 36272.000000 | 45340.000000 | 494.000000 |

8 rows × 42 columns

In [17]:

```
# doing data standardization
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(data_num)
data_num= scaler.transform(data_num)
```

In [18]:

```
data_num = pd.DataFrame(data_num)
data_num.head()
```

Out[18]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.098947 | 0.028396 | -0.358946 | 0.448382 | 0.449569 | 1.068250 | 0.176938 | -0.615269 | -0.2263 |
| **1** | -0.295609 | -1.144997 | -0.428253 | -0.105797 | -0.103427 | -0.232548 | -0.333302 | 1.385127 | -0.2263 |
| **2** | 1.561289 | 0.028396 | -0.087745 | 1.848807 | 1.847004 | 2.369048 | 3.552375 | -0.042626 | -0.2263 |
| **3** | -0.163171 | -1.144997 | -0.428253 | -0.150731 | -0.417289 | -0.232548 | -0.451050 | -2.161146 | -0.2263 |
| **4** | -0.096952 | 0.028396 | 0.783113 | -0.300509 | -0.297722 | -0.015748 | -0.137056 | -0.257474 | -0.2263 |

5 rows × 42 columns

## data cleaning categorical column

In [19]:

```
num_cat =data.select_dtypes(exclude=['int', 'float']).columns
num_cat = [i for i in num_cat]
```

In [20]:

```
imp = SimpleImputer( strategy='most_frequent')
imp.fit(data[num_cat])
```

Out[20]:

```
SimpleImputer(copy=True, fill_value=None, missing_values=nan,
       strategy='most_frequent', verbose=0)
```

In [21]:

```
data[num_cat] = imp.transform(data[num_cat])
data_cat = data[num_cat]
```

In [22]:

```
data_cat.head()
```

Out[22]:

| | Var192 | Var193 | Var194 | Var195 | Var196 | Var197 | Var198 | Var199 | Var200 |
|---|---|---|---|---|---|---|---|---|---|
| **0** | bZkvyxLkBI | RO12 | SEuy | taul | 1K8T | lK27 | ka_ns41 | nQUveAzAF7 | yP09M03 |
| **1** | CEat0G8rTN | RO12 | SEuy | taul | 1K8T | 2Ix5 | qEdASpP | y2LIM01bE1 | yP09M03 |
| **2** | eOQt0GoOh3 | AERks4l | SEuy | taul | 1K8T | ffXs | NldASpP | y4g9XoZ | vynJTq9 |
| **3** | jg69tYsGvO | RO12 | SEuy | taul | 1K8T | ssAy | _ybO0dd | 4hMlgkf58mhwh | yP09M03 |
| **4** | IXSgUHShse | RO12 | SEuy | taul | 1K8T | uNkU | EKR938I | ThrHXVS | 0v21jmy |

5 rows × 34 columns

In [23]:

```
# finding no of category in categorical columns
data_cat= data_cat.astype('category')
cat_level = data_cat.apply(lambda col: len(col.cat.categories))
cat_level
```

Out[23]:

```
Var192      361
Var193       51
Var194        3
Var195       23
Var196        4
Var197      225
Var198     4291
Var199     5073
Var200    15415
Var201        2
Var202     5713
Var203        5
Var204      100
Var205        3
Var206       21
Var207       14
Var208        2
Var210        6
Var211        2
Var212       81
Var214    15415
Var216     2016
Var217    13990
Var218        2
Var219       22
Var220     4291
Var221        7
Var222     4291
Var223        4
Var225        3
Var226       23
Var227        7
Var228       30
Var229        4
dtype: int64
```

In [24]:

```
processed_cat_col = cat_level[cat_level <= 5000].index

processed_cat_col
```

Out[24]:

```
Index(['Var192', 'Var193', 'Var194', 'Var195', 'Var196', 'Var197', 'Va
r198',
       'Var201', 'Var203', 'Var204', 'Var205', 'Var206', 'Var207', 'Va
r208',
       'Var210', 'Var211', 'Var212', 'Var216', 'Var218', 'Var219', 'Va
r220',
       'Var221', 'Var222', 'Var223', 'Var225', 'Var226', 'Var227', 'Va
r228',
       'Var229'],
      dtype='object')
```

In [25]:

```
data_cat_new = data_cat[processed_cat_col]
data_cat_new.head()
```

Out[25]:

| | Var192 | Var193 | Var194 | Var195 | Var196 | Var197 | Var198 | Var201 | Var203 | Var204 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | bZkvyxLkBI | RO12 | SEuy | taul | 1K8T | lK27 | ka_ns41 | smXZ | 9_Y1 | FbIm | ... |
| **1** | CEat0G8rTN | RO12 | SEuy | taul | 1K8T | 2Ix5 | qEdASpP | smXZ | 9_Y1 | k13i | ... |
| **2** | eOQt0GoOh3 | AERks4l | SEuy | taul | 1K8T | ffXs | NIdASpP | smXZ | 9_Y1 | MGOA | ... |
| **3** | jg69tYsGvO | RO12 | SEuy | taul | 1K8T | ssAy | _ybO0dd | smXZ | 9_Y1 | YULl | ... |
| **4** | IXSgUHShse | RO12 | SEuy | taul | 1K8T | uNkU | EKR938I | smXZ | 9_Y1 | RVjC | ... |

5 rows × 29 columns

In [26]:

```
data = pd.concat([data_num,data_cat_new],axis=1)
```

In [27]:

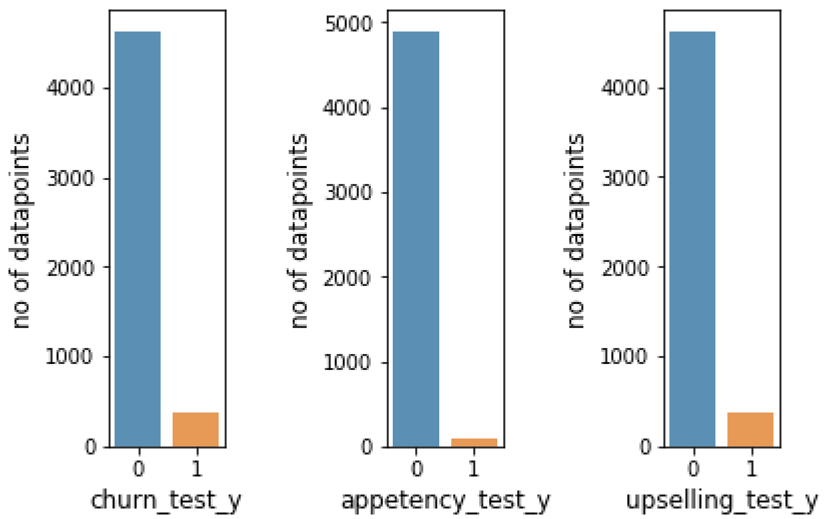
```
data_train = data.head(int(len(data)*(90/100)))

appetency_train_y = appetency.head(int(len(data)*(90/100)))
churn_train_y = churn.head(int(len(data)*(90/100)))
upselling_train_y = upselling.head(int(len(data)*(90/100)))
```

In [28]:

```
data_test = data.tail(int(len(data)*(10/100)))

appetency_test_y = appetency.tail(int(len(data)*(10/100)))
churn_test_y = churn.tail(int(len(data)*(10/100)))
upselling_test_y = upselling.tail(int(len(data)*(10/100)))
```

In [29]:

```
count_churn_test_y = churn_test_y['Churn'].value_counts()
count_appetency_test_y = appetency_test_y['Appetency'].value_counts()
count_upselling_test_y = upselling_test_y['Upselling'].value_counts()

plt.subplot(151)
sns.barplot(count_churn_test_y.index, count_churn_test_y.values, alpha=0.8)
plt.ylabel('no of datapoints', fontsize=12)
plt.xlabel('churn_test_y', fontsize=12)

plt.subplot(153)
sns.barplot(count_appetency_test_y.index, count_appetency_test_y.values, alpha=0.8)
plt.ylabel('no of datapoints', fontsize=12)
plt.xlabel('appetency_test_y', fontsize=12)

plt.subplot(155)
sns.barplot(count_upselling_test_y.index, count_upselling_test_y.values, alpha=0.8)
plt.ylabel('no of datapoints', fontsize=12)
plt.xlabel('upselling_test_y', fontsize=12)
plt.show()
```



In [48]:

```
data_train.head()
```

Out[48]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.098947 | 0.028396 | -0.358946 | 0.448382 | 0.449569 | 1.068250 | 0.176938 | -0.615269 | -0.2263 |
| **1** | -0.295609 | -1.144997 | -0.428253 | -0.105797 | -0.103427 | -0.232548 | -0.333302 | 1.385127 | -0.2263 |
| **2** | 1.561289 | 0.028396 | -0.087745 | 1.848807 | 1.847004 | 2.369048 | 3.552375 | -0.042626 | -0.2263 |
| **3** | -0.163171 | -1.144997 | -0.428253 | -0.150731 | -0.417289 | -0.232548 | -0.451050 | -2.161146 | -0.2263 |
| **4** | -0.096952 | 0.028396 | 0.783113 | -0.300509 | -0.297722 | -0.015748 | -0.137056 | -0.257474 | -0.2263 |

5 rows × 71 columns

**train**

In [41]:

```
data_train_cat_col =data_train.select_dtypes(exclude=['int', 'float']).columns
```

In [44]:

```
numeric_col_train = data_train._get_numeric_data().columns
numeric_col_train = [i for i in numeric_col_train]
data_train_num = data_train[numeric_col_train]
```

In [45]:

```
cat_one_hot_train = data_train[data_train_cat_col].apply(LabelEncoder().fit_transfor
```

In [50]:

```
#cat_one_hot_train = pd.get_dummies(data_train[data_train_cat_col])
```

In [46]:

```
df_train = pd.concat([data_train_num,cat_one_hot_train],axis=1)
```

In [49]:

```
df_train.head()
```

Out[49]:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.098947 | 0.028396 | -0.358946 | 0.448382 | 0.449569 | 1.068250 | 0.176938 | -0.615269 | -0.2263 |
| **1** | -0.295609 | -1.144997 | -0.428253 | -0.105797 | -0.103427 | -0.232548 | -0.333302 | 1.385127 | -0.2263 |
| **2** | 1.561289 | 0.028396 | -0.087745 | 1.848807 | 1.847004 | 2.369048 | 3.552375 | -0.042626 | -0.2263 |
| **3** | -0.163171 | -1.144997 | -0.428253 | -0.150731 | -0.417289 | -0.232548 | -0.451050 | -2.161146 | -0.2263 |
| **4** | -0.096952 | 0.028396 | 0.783113 | -0.300509 | -0.297722 | -0.015748 | -0.137056 | -0.257474 | -0.2263 |

5 rows × 71 columns

**test**

In [50]:

```
data_test_cat_col =data_test.select_dtypes(exclude=['int', 'float']).columns
numeric_col_test = data_test._get_numeric_data().columns
numeric_col_test = [i for i in numeric_col_test]
data_test_num = data_test[numeric_col_test]
cat_one_hot_test = data_test[data_test_cat_col].apply(LabelEncoder().fit_transform)
df_test = pd.concat([data_test_num,cat_one_hot_test],axis=1)
df_test.shape
```

Out[50]:

```
(5000, 71)
```

# Machine Learning Models

## Naive Bayes

In [34]:

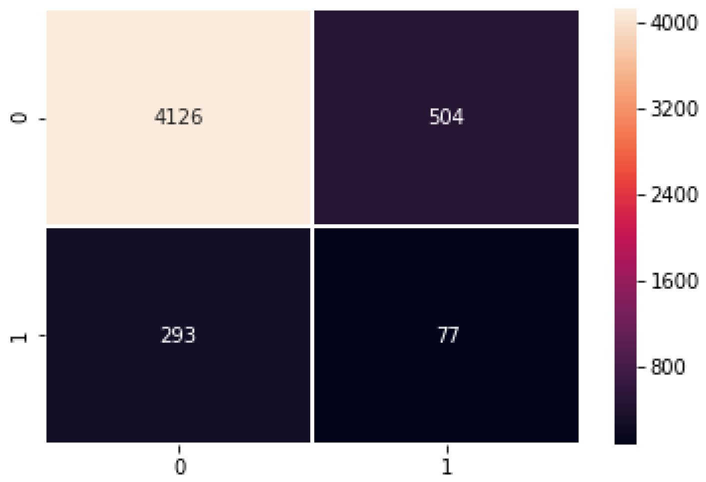
```
import sklearn
sklearn.metrics.SCORERS.keys()
```

Out[34]:

```
dict_keys(['f1_macro', 'f1_micro', 'recall', 'neg_mean_squared_log_err
or', 'roc_auc', 'brier_score_loss', 'average_precision', 'neg_median_a
bsolute_error', 'f1_weighted', 'fowlkes_mallows_score', 'completeness_
score', 'neg_mean_squared_error', 'mutual_info_score', 'r2', 'v_measur
e_score', 'precision_weighted', 'recall_macro', 'precision_micro', 'ne
g_log_loss', 'adjusted_rand_score', 'precision', 'f1', 'balanced_accur
acy', 'accuracy', 'explained_variance', 'homogeneity_score', 'precisio
n_macro', 'recall_micro', 'f1_samples', 'recall_weighted', 'neg_mean_a
bsolute_error', 'normalized_mutual_info_score', 'precision_samples',
'adjusted_mutual_info_score', 'recall_samples'])
```

In [51]:

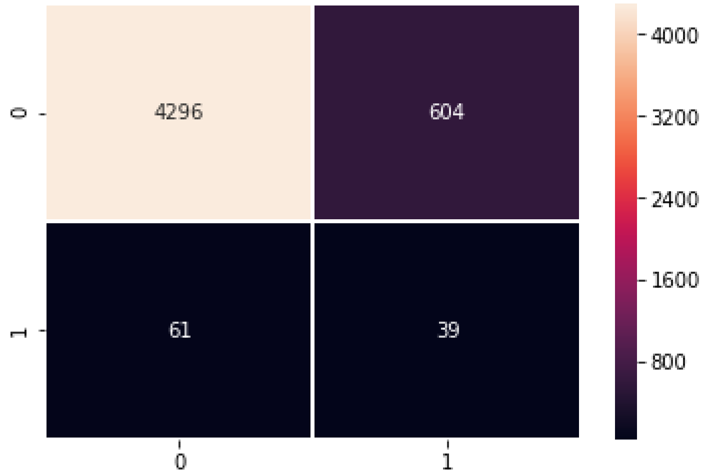
```
tuned_parameters ={'alpha':[i/100 for i in range(0,101,10)]}
model = RandomizedSearchCV(BernoulliNB(),param_distributions=tuned_parameters,scorin
model.fit(df_train, churn_train_y)
pred=model.predict(df_test)
conf_mat = confusion_matrix(churn_test_y,pred)
sns.heatmap(conf_mat,annot=True,fmt="d",linewidths=.5)
plt.show()

print("The best parameters are %s with a score of %0.2f"
      % (model.best_params_, model.best_score_))
nb_churn_result = roc_auc_score(churn_test_y, pred,)
print("roc_auc for Churn is: %0.3f"%(roc_auc_score(churn_test_y, pred)))
print("Precision for Churn is: %0.3f"%(precision_score(churn_test_y, pred)))
print("Recall for Churn is: %0.3f"%(recall_score(churn_test_y, pred)))
print("F1-Score for Churn is: %0.3f"%(f1_score(churn_test_y, pred)))
```




```
The best parameters are {'alpha': 0.2} with a score of 0.62
roc_auc for Churn is: 0.550
Precision for Churn is: 0.133
Recall for Churn is: 0.208
F1-Score for Churn is: 0.162
```

In [52]:

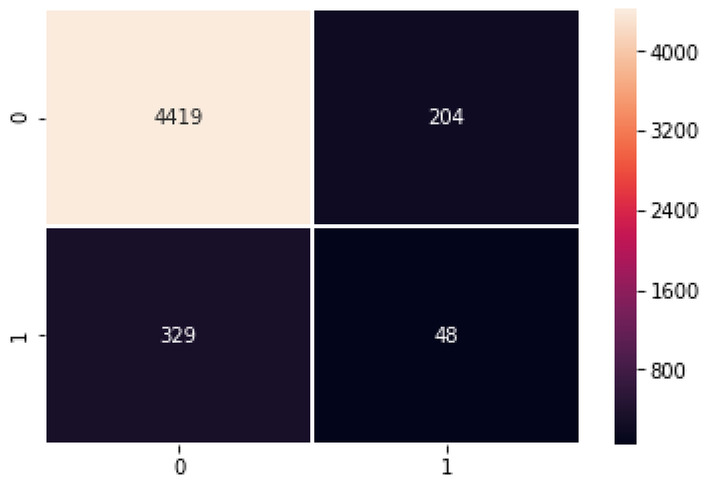
```
tuned_parameters ={'alpha':[i/100 for i in range(0,101,10)]}
model = RandomizedSearchCV(BernoulliNB(),param_distributions=tuned_parameters,scorin
model.fit(df_train, appetency_train_y)
pred=model.predict(df_test)
conf_mat = confusion_matrix(appetency_test_y,pred)
sns.heatmap(conf_mat,annot=True,fmt="d",linewidths=.5)
plt.show()

print("The best parameters are %s with a score of %0.2f"
      % (model.best_params_, model.best_score_))
nb_appetency_result = roc_auc_score(appetency_test_y, pred,)
print("roc_auc for appetency is: %0.3f"%(roc_auc_score(appetency_test_y, pred)))
print("Precision for appetency is: %0.3f"%(precision_score(appetency_test_y, pred)))
print("Recall for appetency is: %0.3f"%(recall_score(appetency_test_y, pred)))
print("F1-Score for appetency is: %0.3f"%(f1_score(appetency_test_y, pred)))
```




```
The best parameters are {'alpha': 0.9} with a score of 0.75
roc_auc for appetency is: 0.633
Precision for appetency is: 0.061
Recall for appetency is: 0.390
F1-Score for appetency is: 0.105
```

In [53]:

```
tuned_parameters ={'alpha':[i/100 for i in range(0,101,10)]}
model = RandomizedSearchCV(BernoulliNB(),param_distributions=tuned_parameters,scorin
model.fit(df_train, upselling_train_y)
pred=model.predict(df_test)
conf_mat = confusion_matrix(upselling_test_y,pred)
sns.heatmap(conf_mat,annot=True,fmt="d",linewidths=.5)
plt.show()

print("The best parameters are %s with a score of %0.2f"
      % (model.best_params_, model.best_score_))
nb_upselling_result = roc_auc_score(upselling_test_y, pred,)
print("roc_auc for upselling is: %0.3f"%(roc_auc_score(upselling_test_y, pred)))
print("Precision for upselling is: %0.3f"%(precision_score(upselling_test_y, pred)))
print("Recall for upselling is: %0.3f"%(recall_score(upselling_test_y, pred)))
print("F1-Score for upselling is: %0.3f"%(f1_score(upselling_test_y, pred)))
```

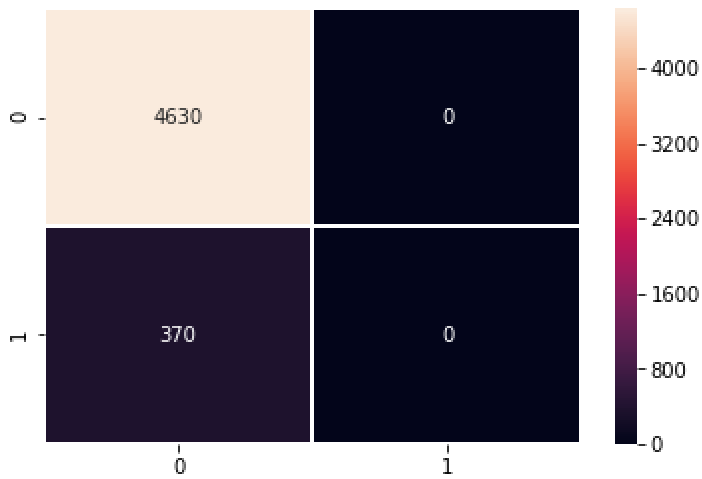



```
The best parameters are {'alpha': 0.1} with a score of 0.71
roc_auc for upselling is: 0.542
Precision for upselling is: 0.190
Recall for upselling is: 0.127
F1-Score for upselling is: 0.153
```

# Linear SVC

In [54]:

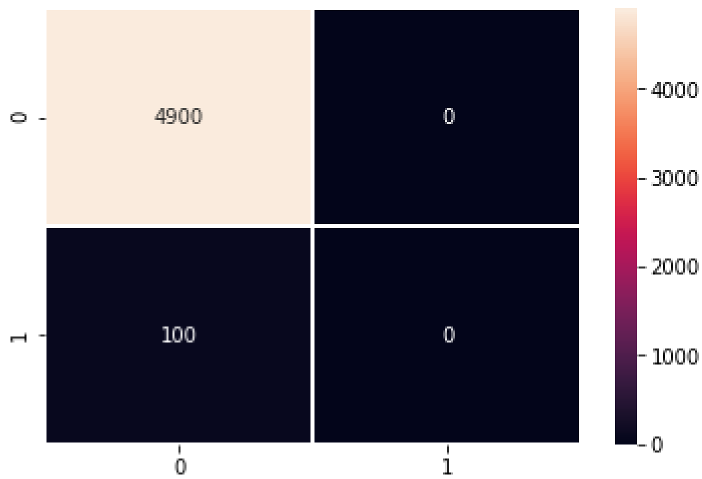
```
from scipy.stats import uniform
import scipy
tuned_parameters ={'penalty':['l1','l2','elasticnet'],
             'alpha':[500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001,

model = RandomizedSearchCV(SGDClassifier(),param_distributions=tuned_parameters,scor
model.fit(df_train, churn_train_y)
pred=model.predict(df_test)
conf_mat = confusion_matrix(churn_test_y,pred)
sns.heatmap(conf_mat,annot=True,fmt="d",linewidths=.5)
plt.show()

print("The best parameters are %s with a score of %0.2f"
      % (model.best_params_, model.best_score_))
sgd_churn_result = roc_auc_score(churn_test_y, pred)
print("roc_auc for Churn is: %0.3f"%(roc_auc_score(churn_test_y, pred)))
print("Precision for Churn is: %0.3f"%(precision_score(churn_test_y, pred)))
print("Recall for Churn is: %0.3f"%(recall_score(churn_test_y, pred)))
print("F1-Score for Churn is: %0.3f"%(f1_score(churn_test_y, pred)))
```




```
The best parameters are {'penalty': 'elasticnet', 'alpha': 0.1} with a
score of 0.55
roc_auc for Churn is: 0.500
Precision for Churn is: 0.000
Recall for Churn is: 0.000
F1-Score for Churn is: 0.000
```

In [55]:

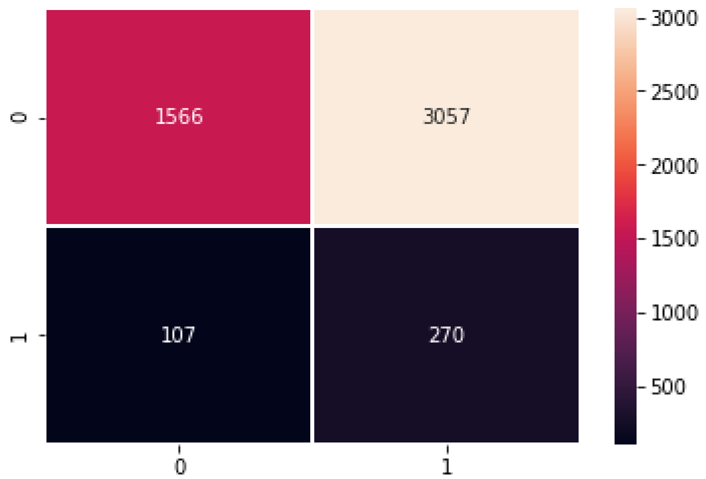
```
from scipy.stats import uniform
import scipy
tuned_parameters ={'penalty':['l1','l2','elasticnet'],
              'alpha':[500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001,

model = RandomizedSearchCV(SGDClassifier(),param_distributions=tuned_parameters,scor
model.fit(df_train, appetency_train_y)
pred=model.predict(df_test)
conf_mat = confusion_matrix(appetency_test_y,pred)
sns.heatmap(conf_mat,annot=True,fmt="d",linewidths=.5)
plt.show()

print("The best parameters are %s with a score of %0.2f"
      % (model.best_params_, model.best_score_))
sgd_appetency_result = roc_auc_score(appetency_test_y, pred)
print("roc_auc for appetency is: %0.3f"%(roc_auc_score(appetency_test_y, pred)))
print("Precision for appetency is: %0.3f"%(precision_score(appetency_test_y, pred)))
print("Recall for appetency is: %0.3f"%(recall_score(appetency_test_y, pred)))
print("F1-Score for appetency is: %0.3f"%(f1_score(appetency_test_y, pred)))
```




```
The best parameters are {'penalty': 'l1', 'alpha': 1e-05} with a score
of 0.52
roc_auc for appetency is: 0.500
Precision for appetency is: 0.000
Recall for appetency is: 0.000
F1-Score for appetency is: 0.000
```

In [56]:

```
from scipy.stats import uniform
import scipy
tuned_parameters ={'penalty':['elasticnet'],
             'alpha':[500,100,50,10,5,1,0.5,0.1,0.05,0.01,0.005,0.001,0.0005,0.0001,

model = RandomizedSearchCV(SGDClassifier(),param_distributions=tuned_parameters,scor
model.fit(df_train, upselling_train_y)
pred=model.predict(df_test)
conf_mat = confusion_matrix(upselling_test_y,pred)
sns.heatmap(conf_mat,annot=True,fmt="d",linewidths=.5)
plt.show()

print("The best parameters are %s with a score of %0.2f"
      % (model.best_params_, model.best_score_))
sgd_upselling_result = roc_auc_score(upselling_test_y, pred)
print("roc_auc for upselling is: %0.3f"%(roc_auc_score(upselling_test_y, pred)))
print("Precision for upselling is: %0.3f"%(precision_score(upselling_test_y, pred)))
print("Recall for upselling is: %0.3f"%(recall_score(upselling_test_y, pred)))
print("F1-Score for upselling is: %0.3f"%(f1_score(upselling_test_y, pred)))
```

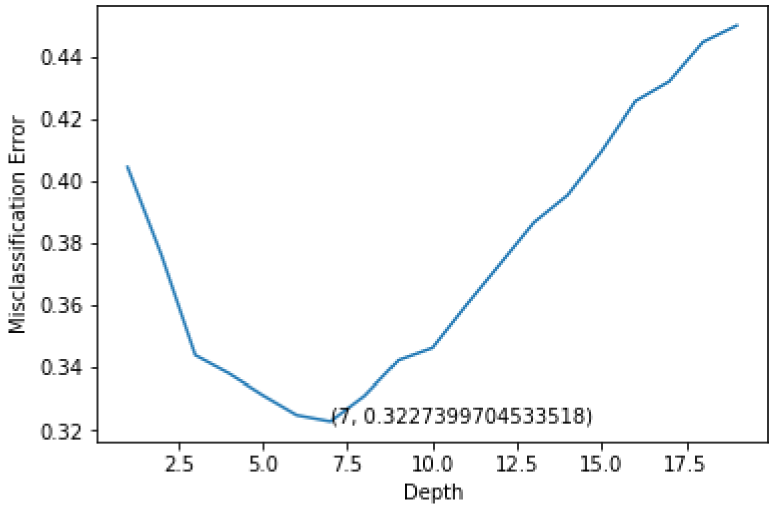


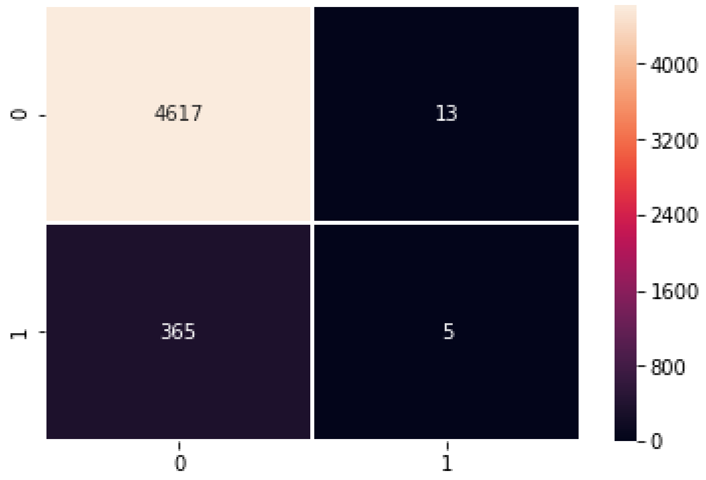

```
The best parameters are {'penalty': 'elasticnet', 'alpha': 1e-05} with
a score of 0.09
roc_auc for upselling is: 0.527
Precision for upselling is: 0.081
Recall for upselling is: 0.716
F1-Score for upselling is: 0.146
```

# Decision Tree

In [57]:

```
depth = [x for x in range(1,20,1)]
cv_scores = []
for d in depth:
    clf = DecisionTreeClassifier(max_depth = d)
    scores = cross_val_score(clf, df_train, churn_train_y, cv=3, scoring='roc_auc')
    cv_scores.append(scores.mean())

error = [1 - x for x in cv_scores]
optimal_depth = depth[error.index(min(error))]
plt.plot(depth, error)
xy = (optimal_depth, min(error))
plt.annotate('(%s, %s)' % xy, xy = xy, textcoords='data')
plt.xlabel("Depth")
plt.ylabel("Misclassification Error")
plt.show()

model = DecisionTreeClassifier(max_depth = optimal_depth) # MAx depth obtained from
model.fit(df_train, churn_train_y)
pred=model.predict(df_test)
conf_mat = confusion_matrix(churn_test_y,pred)
sns.heatmap(conf_mat,annot=True,fmt="d",linewidths=.5)
plt.show()

dt_churn_result = roc_auc_score(churn_test_y, pred)
print("roc_auc for churn is: %0.3f"%(roc_auc_score(churn_test_y, pred)))
print("Precision for churn is: %0.3f"%(precision_score(churn_test_y, pred)))
print("Recall for churn is: %0.3f"%(recall_score(churn_test_y, pred)))
print("F1-Score for churn is: %0.3f"%(f1_score(churn_test_y, pred)))
```






```
roc_auc for churn is: 0.505
Precision for churn is: 0.278
Recall for churn is: 0.014
F1-Score for churn is: 0.026
```

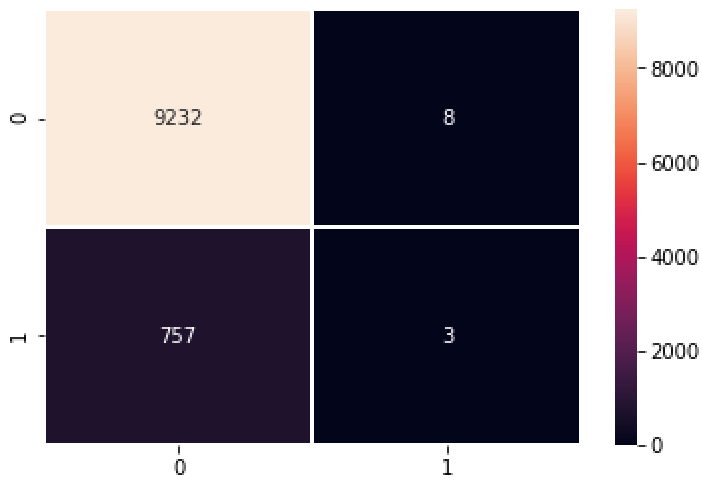
In [51]:

```
from sklearn import model_selection
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
seed = 7
kfold = model_selection.KFold(n_splits=5, random_state=seed)
cart = DecisionTreeClassifier()
num_trees = 50
model = BaggingClassifier(base_estimator=cart, n_estimators=num_trees, random_state=
results = model_selection.cross_val_score(model, df_train, churn_train_y,n_jobs=-1,s
print(results.mean())
```

```
0.6480924064111282
```

In [52]:

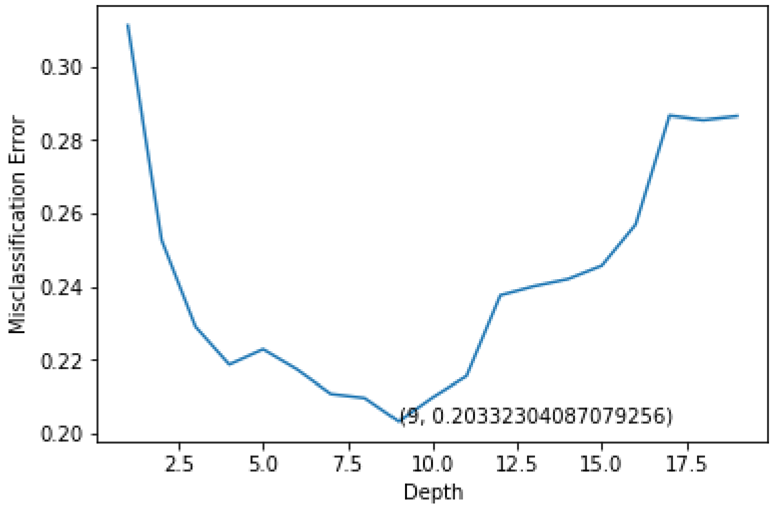
```
model.fit(df_train, churn_train_y)
pred=model.predict(df_test)
conf_mat = confusion_matrix(churn_test_y,pred)
sns.heatmap(conf_mat,annot=True,fmt="d",linewidths=.5)
plt.show()

dt_churn_result = roc_auc_score(churn_test_y, pred)
print("roc_auc for churn is: %0.3f"%(roc_auc_score(churn_test_y, pred)))
print("Precision for churn is: %0.3f"%(precision_score(churn_test_y, pred)))
print("Recall for churn is: %0.3f"%(recall_score(churn_test_y, pred)))
print("F1-Score for churn is: %0.3f"%(f1_score(churn_test_y, pred)))
```

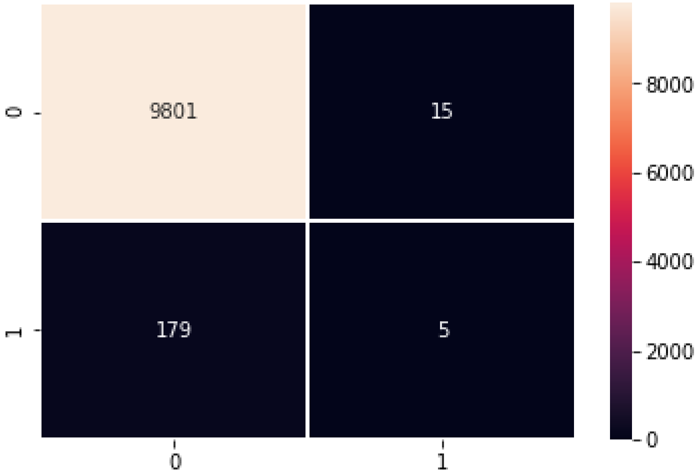



```
roc_auc for churn is: 0.502
Precision for churn is: 0.273
Recall for churn is: 0.004
F1-Score for churn is: 0.008
```

In [52]:

```
depth = [x for x in range(1,20,1)]
cv_scores = []
for d in depth:
    clf = DecisionTreeClassifier(max_depth = d)
    scores = cross_val_score(clf, df_train, appetency_train_y, cv=3, scoring='roc_au
    cv_scores.append(scores.mean())

error = [1 - x for x in cv_scores]
optimal_depth = depth[error.index(min(error))]
plt.plot(depth, error)
xy = (optimal_depth, min(error))
plt.annotate('(%s, %s)' % xy, xy = xy, textcoords='data')
plt.xlabel("Depth")
plt.ylabel("Misclassification Error")
plt.show()

model = DecisionTreeClassifier(max_depth = optimal_depth) # MAx depth obtained from
model.fit(df_train, appetency_train_y)
pred=model.predict(df_test)
conf_mat = confusion_matrix(appetency_test_y,pred)
sns.heatmap(conf_mat,annot=True,fmt="d",linewidths=.5)
plt.show()

dt_appetency_result = roc_auc_score(appetency_test_y, pred)
print("roc_auc for appetency is: %0.3f"%(roc_auc_score(appetency_test_y, pred)))
print("Precision for appetency is: %0.3f"%(precision_score(appetency_test_y, pred))
print("Recall for appetency is: %0.3f"%(recall_score(appetency_test_y, pred)))
print("F1-Score for appetency is: %0.3f"%(f1_score(appetency_test_y, pred)))
```

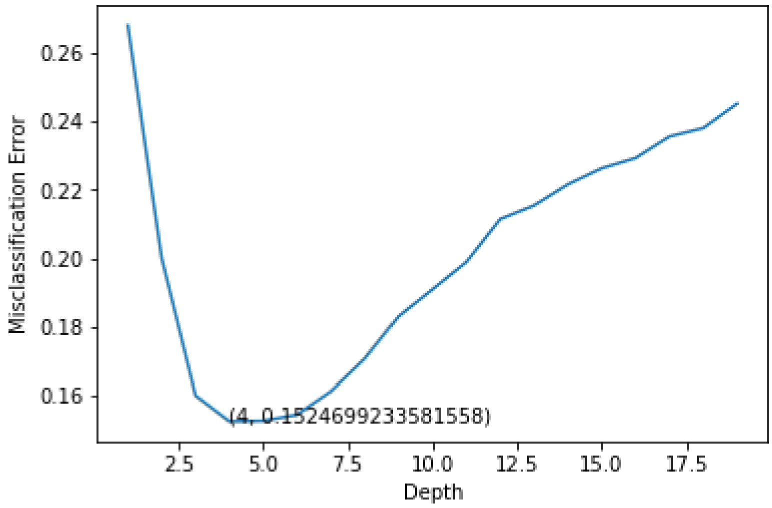


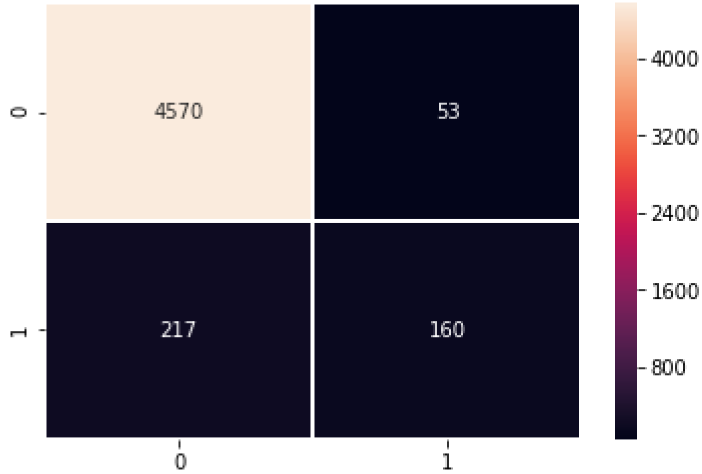



```
roc_auc for appetency is: 0.513
Precision for appetency is: 0.250
Recall for appetency is: 0.027
F1-Score for appetency is: 0.049
```

In [62]:

```
depth = [x for x in range(1,20,1)]
cv_scores = []
for d in depth:
    clf = DecisionTreeClassifier(max_depth = d)
    scores = cross_val_score(clf, df_train, upselling_train_y, cv=3, scoring='roc_au
    cv_scores.append(scores.mean())

error = [1 - x for x in cv_scores]
optimal_depth = depth[error.index(min(error))]
plt.plot(depth, error)
xy = (optimal_depth, min(error))
plt.annotate('(%s, %s)' % xy, xy = xy, textcoords='data')
plt.xlabel("Depth")
plt.ylabel("Misclassification Error")
plt.show()

model = DecisionTreeClassifier(max_depth = optimal_depth) # MAx depth obtained from
model.fit(df_train, upselling_train_y)
pred=model.predict(df_test)
conf_mat = confusion_matrix(upselling_test_y,pred)
sns.heatmap(conf_mat,annot=True,fmt="d",linewidths=.5)
plt.show()

dt_upselling_result = roc_auc_score(upselling_test_y, pred)
print("roc_auc for upselling is: %0.3f"%(roc_auc_score(upselling_test_y, pred)))
print("Precision for upselling is: %0.3f"%(precision_score(upselling_test_y, pred)))
print("Recall for upselling is: %0.3f"%(recall_score(upselling_test_y, pred)))
print("F1-Score for upselling is: %0.3f"%(f1_score(upselling_test_y, pred)))
```






```
roc_auc for upselling is: 0.706
Precision for upselling is: 0.751
Recall for upselling is: 0.424
F1-Score for upselling is: 0.542
```

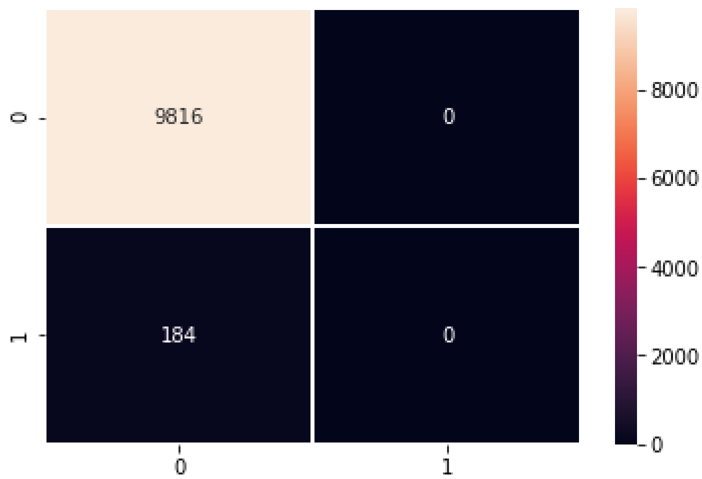
# Xgboost

In [ ]:

```
from xgboost import XGBClassifier
params = {'n_estimators':[i for i in range(100,1001,100)],
         'max_depth': [x for x in range(0,100)]}
xgb = XGBClassifier(nthread=-1)

model = RandomizedSearchCV(xgb,params,cv=None,scoring = 'roc_auc',)
model.fit(df_train, churn_train_y)
pred=model.predict(df_test)
conf_mat = confusion_matrix(df_test,pred)
sns.heatmap(conf_mat,annot=True,fmt="d",linewidths=.5)
plt.show()

xgb_churn_result = roc_auc_score(df_test, pred)
print("roc_auc for churn is: %0.3f"%(roc_auc_score(churn_test_y, pred)))
print("Precision for churn is: %0.3f"%(precision_score(churn_test_y, pred)))
print("Recall for churn is: %0.3f"%(recall_score(churn_test_y, pred)))
print("F1-Score for churn is: %0.3f"%(f1_score(churn_test_y, pred)))
```

In [67]:

```
from xgboost import XGBClassifier
params = {'n_estimators':[i for i in range(100,1001,100)],
         'subsample':[0.7, 0.8, 0.9],
         'min_child_weight':[x for x in range(1,50,3)],
         'reg_lambda':[x for x in range(100,1000,100) ],
         'max_depth': [x for x in range(0,50,3)]}
xgb = XGBClassifier(nthread=-1)

model = RandomizedSearchCV(xgb,params,cv=None,scoring = 'roc_auc')
model.fit(appetency_tr_x, appetency_tr_y)
pred=model.predict(appetency_cv_x)
conf_mat = confusion_matrix(appetency_cv_y,pred)
sns.heatmap(conf_mat,annot=True,fmt="d",linewidths=.5)
plt.show()

xgb_appetency_result = roc_auc_score(appetency_cv_y, pred)
print("roc_auc for appetency is: %0.3f"%(roc_auc_score(appetency_cv_y, pred)))
print("Precision for appetency is: %0.3f"%(precision_score(appetency_cv_y, pred)))
print("Recall for appetency is: %0.3f"%(recall_score(appetency_cv_y, pred)))
print("F1-Score for appetency is: %0.3f"%(f1_score(appetency_cv_y, pred)))
```




```
roc_auc for appetency is: 0.500
Precision for appetency is: 0.000
Recall for appetency is: 0.000
F1-Score for appetency is: 0.000
```

In [ ]:

```
from xgboost import XGBClassifier
params = {'n_estimators':[i for i in range(100,1001,100)],
        'subsample':[0.7, 0.8, 0.9],
        'min_child_weight':[x for x in range(1,50,3)],
        'reg_lambda':[x for x in range(100,1000,100) ],
        'max_depth': [x for x in range(0,50,3)]}
xgb = XGBClassifier(nthread=-1)

model = RandomizedSearchCV(xgb,params,cv=None,scoring = 'roc_auc')
model.fit(upselling_tr_x, upselling_tr_y)
pred=model.predict(upselling_cv_x)
conf_mat = confusion_matrix(upselling_cv_y,pred)
sns.heatmap(conf_mat,annot=True,fmt="d",linewidths=.5)
plt.show()

xgb_upselling_result = roc_auc_score(upselling_cv_y, pred)
print("roc_auc for upselling is: %0.3f"%(roc_auc_score(upselling_cv_y, pred)))
print("Precision for upselling is: %0.3f"%(precision_score(upselling_cv_y, pred)))
print("Recall for upselling is: %0.3f"%(recall_score(upselling_cv_y, pred)))
print("F1-Score for upselling is: %0.3f"%(f1_score(upselling_cv_y, pred)))
```

In [ ]:

```
## final scoring is arithmatic mean of all appetency , appetency and upselling

xgb_score = (xgb_appetency_result + xgb_appetency_result + xgb_upselling_result) / 
xgb_score
```

# NN

In [63]:

```
from sklearn import metrics
from keras import backend as K

def auc(y_true, y_pred):
    auc = tf.metrics.auc(y_true, y_pred)[1]
    K.get_session().run(tf.local_variables_initializer())
    return auc
```

```
Using TensorFlow backend.
```

In [70]:

```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import BatchNormalization
from keras.initializers import RandomNormal
# some model parameters

output_dim = 2
input_dim = df_train.shape[1]

batch_size = 128
nb_epoch = 10
model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(input_dim,), kernel_initialize
model.add(BatchNormalization())
model.add(Dense(256, activation='relu', kernel_initializer=RandomNormal(mean=0.0, st
model.add(BatchNormalization())
model.add(Dense(128, activation='relu', kernel_initializer=RandomNormal(mean=0.0, st
model.add(BatchNormalization())
model.add(Dense(output_dim, activation='softmax'))

print(model.summary())

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['ac

model_log = model.fit(df_train, upselling_train_y, batch_size=batch_size, epochs=nb_
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 512)               36864
_________________________________________________________________
batch_normalization_4 (Batch (None, 512)               2048
_________________________________________________________________
dense_6 (Dense)              (None, 256)               131328
_________________________________________________________________
batch_normalization_5 (Batch (None, 256)               1024
_________________________________________________________________
dense_7 (Dense)              (None, 128)               32896
_________________________________________________________________
batch_normalization_6 (Batch (None, 128)               512
_________________________________________________________________
dense_8 (Dense)              (None, 2)                 258
=================================================================
Total params: 204,930
Trainable params: 203,138
Non-trainable params: 1,792
_________________________________________________________________
None
Train on 45000 samples, validate on 5000 samples
Epoch 1/10
45000/45000 [==============================] - 5s 110us/step - loss:
0.3326 - acc: 0.9002 - val_loss: 0.2705 - val_acc: 0.9246
Epoch 2/10
45000/45000 [==============================] - 3s 64us/step - loss: 0.
2690 - acc: 0.9265 - val_loss: 0.2799 - val_acc: 0.9246
```

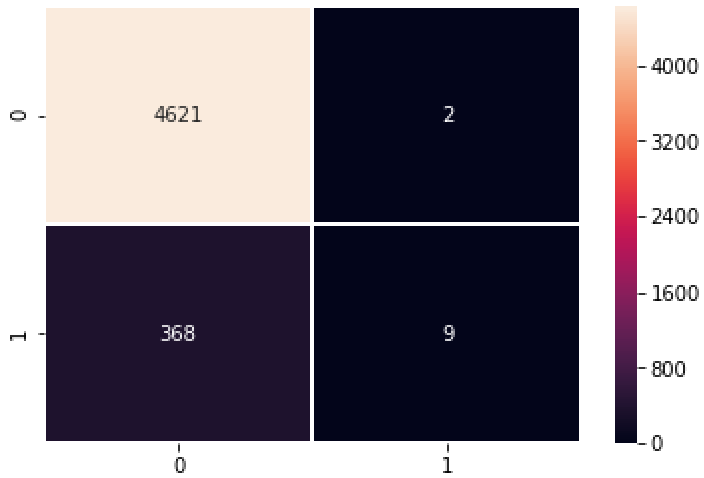
```
Epoch 3/10
45000/45000 [==============================] - 3s 63us/step - loss: 0.
2668 - acc: 0.9265 - val_loss: 0.2710 - val_acc: 0.9246
Epoch 4/10
45000/45000 [==============================] - 3s 64us/step - loss: 0.
2644 - acc: 0.9266 - val_loss: 0.2871 - val_acc: 0.9246
Epoch 5/10
45000/45000 [==============================] - 3s 64us/step - loss: 0.
2604 - acc: 0.9265 - val_loss: 0.2876 - val_acc: 0.9246
Epoch 6/10
45000/45000 [==============================] - 3s 62us/step - loss: 0.
2545 - acc: 0.9265 - val_loss: 0.3399 - val_acc: 0.9246
Epoch 7/10
45000/45000 [==============================] - 3s 62us/step - loss: 0.
2466 - acc: 0.9267 - val_loss: 0.3423 - val_acc: 0.9246
Epoch 8/10
45000/45000 [==============================] - 3s 62us/step - loss: 0.
2350 - acc: 0.9270 - val_loss: 0.2753 - val_acc: 0.9248
Epoch 9/10
45000/45000 [==============================] - 3s 62us/step - loss: 0.
2293 - acc: 0.9278 - val_loss: 0.2401 - val_acc: 0.9248
Epoch 10/10
45000/45000 [==============================] - 3s 63us/step - loss: 0.
2246 - acc: 0.9287 - val_loss: 0.2359 - val_acc: 0.9260
```

In [71]:

```
pred = model.predict_classes(df_test)
```

In [72]:

```
conf_mat = confusion_matrix(upselling_test_y,pred)
sns.heatmap(conf_mat,annot=True,fmt="d",linewidths=.5)
plt.show()

xgb_upselling_result = roc_auc_score(upselling_test_y, pred)
print("roc_auc for upselling is: %0.3f"%(roc_auc_score(upselling_test_y, pred)))
print("Precision for upselling is: %0.3f"%(precision_score(upselling_test_y, pred))
print("Recall for upselling is: %0.3f"%(recall_score(upselling_test_y, pred)))
print("F1-Score for upselling is: %0.3f"%(f1_score(upselling_test_y, pred)))
```




```
roc_auc for upselling is: 0.512
Precision for upselling is: 0.818
Recall for upselling is: 0.024
F1-Score for upselling is: 0.046
```

# Random Forest

In [34]:

```
from mlxtend.feature_selection import SequentialFeatureSelector as sfs
```

In [39]:

```
clf = RandomForestClassifier(n_estimators=20, n_jobs=-1)
sfs1 = sfs(clf,
           k_features=20,
           forward=True,
           floating=False,
           verbose=2,
           scoring='f1')
```

In [ ]:

```
sfs1 = sfs1.fit(df_train,upselling_train_y)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurren
t workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    1.9s remainin
g:    0.0s
[Parallel(n_jobs=1)]: Done 1032 out of 1032 | elapsed:  9.2min finishe
d

[2019-03-08 00:44:33] Features: 1/20 -- score: 0.5413180019887461[Para
llel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.1s remainin
g:    0.0s
[Parallel(n_jobs=1)]: Done 1031 out of 1031 | elapsed: 15.5min finishe
d

[2019-03-08 01:00:00] Features: 2/20 -- score: 0.547608163662988[Paral
lel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent work
ers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.6s remainin
g:    0.0s
[Parallel(n_jobs=1)]: Done 1030 out of 1030 | elapsed: 17.2min finishe
d

[2019-03-08 01:17:14] Features: 3/20 -- score: 0.5483378543118544[Para
llel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.1s remainin
g:    0.0s
[Parallel(n_jobs=1)]: Done 1029 out of 1029 | elapsed: 16.3min finishe
d

[2019-03-08 01:33:31] Features: 4/20 -- score: 0.549060208381184[Paral
lel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent work
ers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.3s remainin
g:    0.0s
[Parallel(n_jobs=1)]: Done 1028 out of 1028 | elapsed: 17.1min finishe
d

[2019-03-08 01:50:38] Features: 5/20 -- score: 0.5496899695969151[Para
llel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent wor
kers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    2.3s remainin
g:    0.0s
```

In [ ]:

```

```