Customer Churn Prediction with XGBoost

Using Gradient Boosted Trees to Predict Mobile Customer Departure

---

Contents

Background

This notebook has been adapted from an AWS blog post (https://aws.amazon.com/blogs/ai/predicting-customer-churn-with-amazon-machine-learning/)

Losing customers is costly for any business. Identifying unhappy customers early on gives you a chance to offer them incentives to stay. This notebook describes using machine learning (ML) for the automated identification of unhappy customers, also known as customer churn prediction. ML models rarely give perfect predictions though, so this notebook is also about how to incorporate the relative costs of prediction mistakes when determining the financial outcome of using ML.

We use an example of churn that is familiar to all of us–leaving a mobile phone operator. Seems like I can always find fault with my provider du jour! And if my provider knows that I'm thinking of leaving, it can offer timely incentives–I can always use a phone upgrade or perhaps have a new feature activated–and I might just stick around. Incentives are often much more cost effective than losing and reacquiring a customer.

Setup

This notebook was created and tested on an ml.m4.xlarge notebook instance.

Let's start by specifying:

- The S3 bucket and prefix that you want to use for training and model data. This should be within the same region as the Notebook Instance, training, and hosting.
- The IAM role arn used to give training and hosting access to your data. See the documentation for how to create these. Note, if more than one role is required for notebook instances, training, and/or hosting, please replace the boto regexp with a the appropriate full IAM role arn string(s).

In [1]:
```
bucket = 'awskrug010122341234'
prefix = 'sagemaker/DEMO-xgboost-churn'

# Define IAM role
import boto3
import re
from sagemaker import get_execution_role

role = get_execution_role()
```

Next, we'll import the Python libraries we'll need for the remainder of the exercise.

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import io
import os
import sys
import time
import json
from IPython.display import display
from time import strftime, gmtime
import sagemaker
from sagemaker.predictor import csv_serializer
```

Data

Mobile operators have historical records on which customers ultimately ended up churning and which continued using the service. We can use this historical information to construct an ML model of one mobile operator’s churn using a process called training. After training the model, we can pass the profile information of an arbitrary customer (the same profile information that we used to train the model) to the model, and have the model predict whether this customer is going to churn. Of course, we expect the model to make mistakes–after all, predicting the future is tricky business! But I’ll also show how to deal with prediction errors.

The dataset we use is publicly available and was mentioned in the book Discovering Knowledge in Data (https://www.amazon.com/dp/0470908742/) by Daniel T. Larose. It is attributed by the author to the University of California Irvine Repository of Machine Learning Datasets. Let's download and read that dataset in now:

In [3]:
```
!wget http://dataminingconsultant.com/DKD2e_data_sets.zip
!unzip -o DKD2e_data_sets.zip
```

```
--2018-09-07 16:10:20--  http://dataminingconsultant.com/DKD2e_data_sets.zip
Resolving dataminingconsultant.com (dataminingconsultant.com)... 160.153.91.162
Connecting to dataminingconsultant.com (dataminingconsultant.com)|160.153.91.162|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1003616 (980K) [application/zip]
Saving to: ‘DKD2e_data_sets.zip’

DKD2e_data_sets.zip 100%[===================>] 980.09K  1.04MB/s    in 0.9s

2018-09-07 16:10:22 (1.04 MB/s) - ‘DKD2e_data_sets.zip’ saved [1003616/1003616]

Archive:  DKD2e_data_sets.zip
 extracting: Data sets/adult.zip
  inflating: Data sets/cars.txt
  inflating: Data sets/cars2.txt
  inflating: Data sets/cereals.CSV
  inflating: Data sets/churn.txt
  inflating: Data sets/ClassifyRisk
  inflating: Data sets/ClassifyRisk - Missing.txt
 extracting: Data sets/DKD2e data sets.zip
  inflating: Data sets/nn1.txt
```

In [4]:
```
churn = pd.read_csv('./Data sets/churn.txt')
pd.set_option('display.max_columns', 500)
churn
```

Out[4]:

	State	Account Length	Area Code	Phone	Int'l Plan	VMail Plan	VMail Message	Day Mins	Day Calls	Day Charge	Eve Mins	Eve Calls	Eve Charge	Night Mins	Night Calls	Night Charge	Intl Mins	Intl Calls	Intl Charge	CustServ Calls	Churn?
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	3	2.70	1	False.
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	3	3.70	1	False.
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	5	3.29	0	False.
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	7	1.78	2	False.
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	3	2.73	3	False.
5	AL	118	510	391-8027	yes	no	0	223.4	98	37.98	220.6	101	18.75	203.9	118	9.18	6.3	6	1.70	0	False.
6	MA	121	510	355-9993	no	yes	24	218.2	88	37.09	348.5	108	29.62	212.6	118	9.57	7.5	7	2.03	3	False.
7	MO	147	415	329-9001	yes	no	0	157.0	79	26.69	103.1	94	8.76	211.8	96	9.53	7.1	6	1.92	0	False.
8	LA	117	408	335-4719	no	no	0	184.5	97	31.37	351.6	80	29.89	215.8	90	9.71	8.7	4	2.35	1	False.
9	WV	141	415	330-8173	yes	yes	37	258.6	84	43.96	222.0	111	18.87	326.4	97	14.69	11.2	5	3.02	0	False.
10	IN	65	415	329-6603	no	no	0	129.1	137	21.95	228.5	83	19.42	208.8	111	9.40	12.7	6	3.43	4	True.
11	RI	74	415	344-9403	no	no	0	187.7	127	31.91	163.4	148	13.89	196.0	94	8.82	9.1	5	2.46	0	False.
12	IA	168	408	363-1107	no	no	0	128.8	96	21.90	104.9	71	8.92	141.1	128	6.35	11.2	2	3.02	1	False.
13	MT	95	510	394-8006	no	no	0	156.6	88	26.62	247.6	75	21.05	192.3	115	8.65	12.3	5	3.32	3	False.
14	IA	62	415	366-9238	no	no	0	120.7	70	20.52	307.2	76	26.11	203.0	99	9.14	13.1	6	3.54	4	False.
15	NY	161	415	351-7269	no	no	0	332.9	67	56.59	317.8	97	27.01	160.6	128	7.23	5.4	9	1.46	4	True.
16	ID	85	408	350-8884	no	yes	27	196.4	139	33.39	280.9	90	23.88	89.3	75	4.02	13.8	4	3.73	1	False.
17	VT	93	510	386-2923	no	no	0	190.7	114	32.42	218.2	111	18.55	129.6	121	5.83	8.1	3	2.19	3	False.
18	VA	76	510	356-2992	no	yes	33	189.7	66	32.25	212.8	65	18.09	165.7	108	7.46	10.0	5	2.70	1	False.
19	TX	73	415	373-2782	no	no	0	224.4	90	38.15	159.5	88	13.56	192.8	74	8.68	13.0	2	3.51	1	False.
20	FL	147	415	396-5800	no	no	0	155.1	117	26.37	239.7	93	20.37	208.8	133	9.40	10.6	4	2.86	0	False.
21	CO	77	408	393-7984	no	no	0	62.4	89	10.61	169.9	121	14.44	209.6	64	9.43	5.7	6	1.54	5	True.
22	AZ	130	415	358-1958	no	no	0	183.0	112	31.11	72.9	99	6.20	181.8	78	8.18	9.5	19	2.57	0	False.
23	SC	111	415	350-2565	no	no	0	110.4	103	18.77	137.3	102	11.67	189.6	105	8.53	7.7	6	2.08	2	False.
24	VA	132	510	343-4696	no	no	0	81.1	86	13.79	245.2	72	20.84	237.0	115	10.67	10.3	2	2.78	0	False.
25	NE	174	415	331-3698	no	no	0	124.3	76	21.13	277.1	112	23.55	250.7	115	11.28	15.5	5	4.19	3	False.
26	WY	57	408	357-3817	no	yes	39	213.0	115	36.21	191.1	112	16.24	182.7	115	8.22	9.5	3	2.57	0	False.
27	MT	54	408	418-6412	no	no	0	134.3	73	22.83	155.5	100	13.22	102.1	68	4.59	14.7	4	3.97	3	False.
28	MO	20	415	353-2630	no	no	0	190.0	109	32.30	258.2	84	21.95	181.5	102	8.17	6.3	6	1.70	0	False.
29	HI	49	510	410-7789	no	no	0	119.3	117	20.28	215.1	109	18.28	178.7	90	8.04	11.1	1	3.00	1	False.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3303	WI	114	415	373-7308	no	yes	26	137.1	88	23.31	155.7	125	13.23	247.6	94	11.14	11.5	7	3.11	2	False.

By modern standards, it’s a relatively small dataset, with only 3,333 records, where each record uses 21 attributes to describe the profile of a customer of an unknown US mobile operator. The attributes are:

- `State`: the US state in which the customer resides, indicated by a two-letter abbreviation; for example, OH or NJ
- `Account Length`: the number of days that this account has been active
- `Area Code`: the three-digit area code of the corresponding customer’s phone number
- `Phone`: the remaining seven-digit phone number
- `Int’l Plan`: whether the customer has an international calling plan: yes/no
- `VMail Plan`: whether the customer has a voice mail feature: yes/no
- `VMail Message`: presumably the average number of voice mail messages per month
- `Day Mins`: the total number of calling minutes used during the day
- `Day Calls`: the total number of calls placed during the day
- `Day Charge`: the billed cost of daytime calls
- `Eve Mins, Eve Calls, Eve Charge`: the billed cost for calls placed during the evening
- `Night Mins`,`Night Calls`,`Night Charge`: the billed cost for calls placed during nighttime
- `Intl Mins`, `Intl Calls`, `Intl Charge`: the billed cost for international calls
- `CustServ Calls`: the number of calls placed to Customer Service
- `Churn?`: whether the customer left the service: true/false

The last attribute, `Churn?`, is known as the target attribute–the attribute that we want the ML model to predict. Because the target attribute is binary, our model will be performing binary prediction, also known as binary classification.

Let's begin exploring the data:

In [5]:
```
# Frequency tables for each categorical feature
for column in churn.select_dtypes(include=['object']).columns:
    display(pd.crosstab(index=churn[column], columns='% observations', normalize='columns'))

# Histograms for each numeric features
display(churn.describe())
%matplotlib inline
hist = churn.hist(bins=30, sharey=True, figsize=(10, 10))
```

col_0	% observations
State	
AK	0.015602
AL	0.024002
AR	0.016502
AZ	0.019202
CA	0.010201
CO	0.019802
CT	0.022202
DC	0.016202
DE	0.018302
FL	0.018902
GA	0.016202
HI	0.015902
IA	0.013201
ID	0.021902
IL	0.017402
IN	0.021302
KS	0.021002
KY	0.017702
LA	0.015302
MA	0.019502
MD	0.021002
ME	0.018602
MI	0.021902
MN	0.025203
MO	0.018902
MS	0.019502
MT	0.020402
NC	0.020402
ND	0.018602
NE	0.018302
NH	0.016802
NJ	0.020402

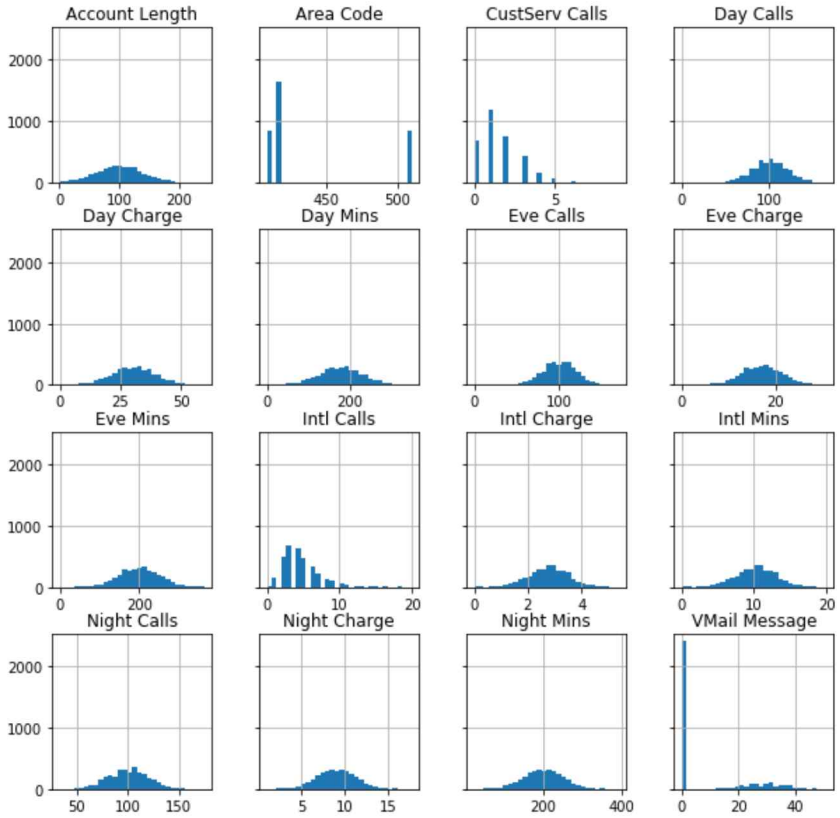
col_0	% observations
Phone	
327-1058	0.0003
327-1319	0.0003
327-3053	0.0003
327-3587	0.0003
327-3850	0.0003
327-3954	0.0003
327-4795	0.0003
327-5525	0.0003
327-5817	0.0003
327-6087	0.0003
327-6179	0.0003
327-6194	0.0003
327-6764	0.0003
327-6989	0.0003
327-8495	0.0003
327-8732	0.0003
327-9289	0.0003
327-9341	0.0003
327-9957	0.0003
328-1206	0.0003
328-1222	0.0003
328-1373	0.0003
328-1522	0.0003
328-1768	0.0003
328-2110	0.0003
328-2236	0.0003
328-2478	0.0003
328-2647	0.0003
328-2982	0.0003
328-3266	0.0003
...	...
421-7205	0.0003

col_0	% observations
Int'l Plan	
no	0.90309
yes	0.09691

col_0	% observations
VMail Plan	
no	0.723372
yes	0.276628

col_0	% observations
Churn?	
False.	0.855086
True.	0.144914

	Account Length	Area Code	VMail Message	Day Mins	Day Calls	Day Charge	Eve Mins	Eve Calls	Eve Charge	Night Mins	Night Calls	Night Charge	Intl Mins	Intl Calls	
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	33
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.980348	100.114311	17.083540	200.872037	100.107711	9.039325	10.237294	4.479448	2.
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.713844	19.922625	4.310668	50.573847	19.568609	2.275873	2.791840	2.461214	0.
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	23.200000	33.000000	1.040000	0.000000	0.000000	0.
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.600000	87.000000	14.160000	167.000000	87.000000	7.520000	8.500000	3.000000	2.
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.400000	100.000000	17.120000	201.200000	100.000000	9.050000	10.300000	4.000000	2.
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.300000	114.000000	20.000000	235.300000	113.000000	10.590000	12.100000	6.000000	3.
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.700000	170.000000	30.910000	395.000000	175.000000	17.770000	20.000000	20.000000	5.



We can see immediately that:

- `State` appears to be quite evenly distributed
- `Phone` takes on too many unique values to be of any practical use. It's possible parsing out the prefix could have some value, but without more context on how these are allocated, we should avoid using it.
- Only 14% of customers churned, so there is some class imabalance, but nothing extreme.
- Most of the numeric features are surprisingly nicely distributed, with many showing bell-like gaussianity. `VMail Message` being a notable exception (and `Area Code` showing up as a feature we should convert to non-numeric).

In [6]:
```
churn = churn.drop('Phone', axis=1)
churn['Area Code'] = churn['Area Code'].astype(object)
```

Next let's look at the relationship between each of the features and our target variable.

In [7]:
```
for column in churn.select_dtypes(include=['object']).columns:
    if column != 'Churn?':
        display(pd.crosstab(index=churn[column], columns=churn['Churn?'], normalize='columns'))

for column in churn.select_dtypes(exclude=['object']).columns:
    print(column)
    hist = churn[[column, 'Churn?']].hist(by='Churn?', bins=30)
    plt.show()
```

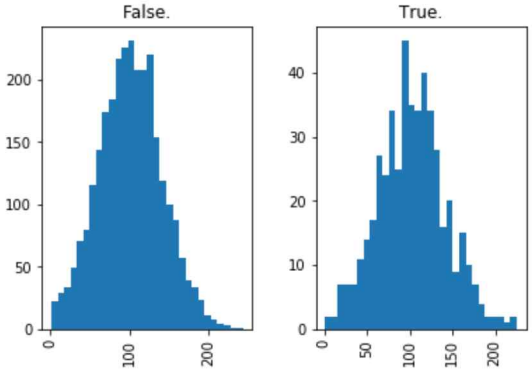
Churn?	False.	True.
State		
AK	0.017193	0.006211
AL	0.025263	0.016563
AR	0.015439	0.022774
AZ	0.021053	0.008282
CA	0.008772	0.018634
CO	0.020000	0.018634
CT	0.021754	0.024845
DC	0.017193	0.010352
DE	0.018246	0.018634
FL	0.019298	0.016563
GA	0.016140	0.016563
HI	0.017544	0.006211
IA	0.014386	0.006211
ID	0.022456	0.018634
IL	0.018596	0.010352
IN	0.021754	0.018634
KS	0.020000	0.026915
KY	0.017895	0.016563
LA	0.016491	0.008282
MA	0.018947	0.022774
MD	0.018596	0.035197
ME	0.017193	0.026915
MI	0.020000	0.033126
MN	0.024211	0.031056
MO	0.019649	0.014493
MS	0.017895	0.028986
MT	0.018947	0.028986
NC	0.020000	0.022774
ND	0.019649	0.012422
NE	0.019649	0.010352
NH	0.016491	0.018634
NJ	0.017544	0.037267

Churn?	False.	True.
Area Code		
408	0.251228	0.252588
415	0.497895	0.488613
510	0.250877	0.258799

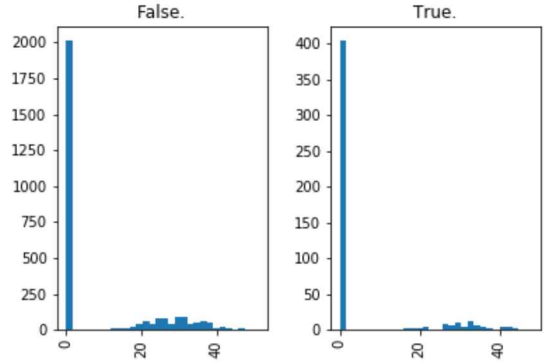
Churn?	False.	True.
Int'l Plan		
no	0.934737	0.716356
yes	0.065263	0.283644

Churn?	False.	True.
VMail Plan		
no	0.704561	0.834369
yes	0.295439	0.165631

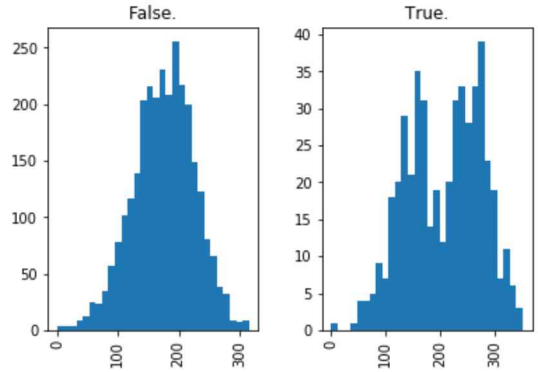
Account Length



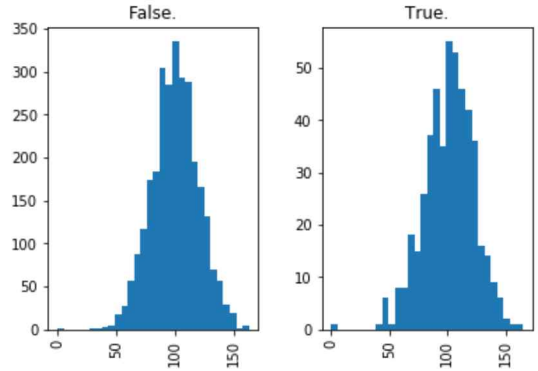
VMail Message



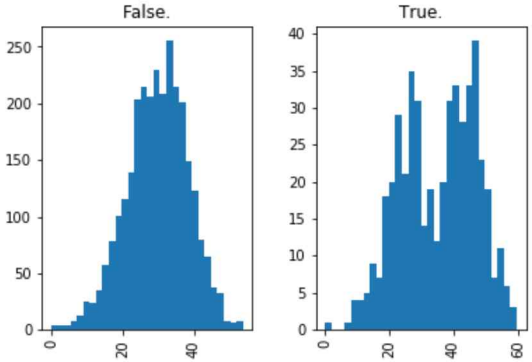
Day Mins



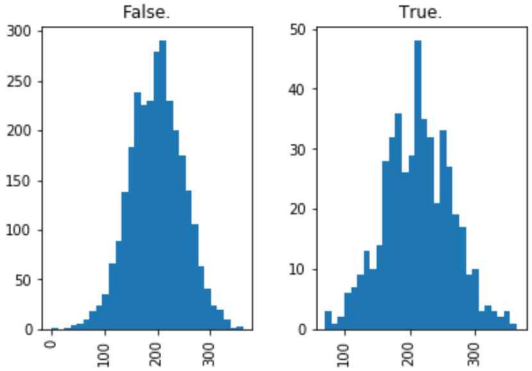
Day Calls



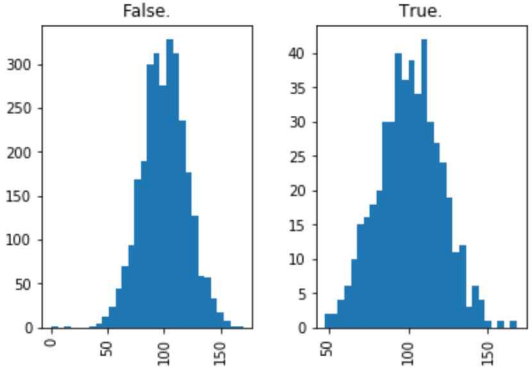
Day Charge



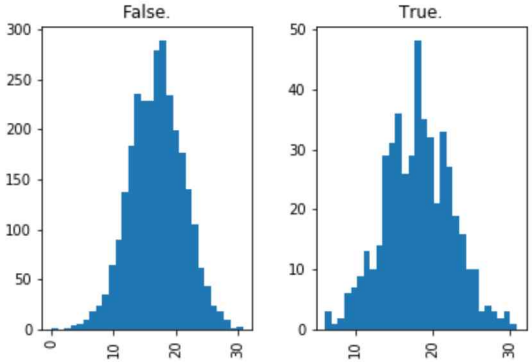
Eve Mins



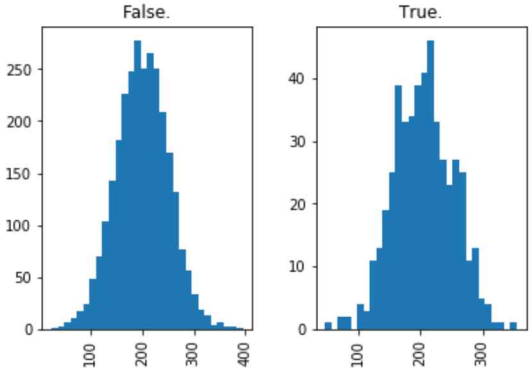
Eve Calls



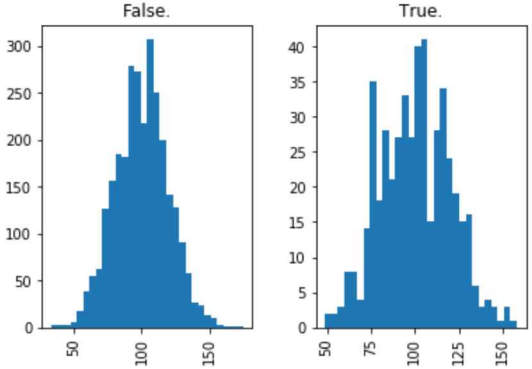
Eve Charge



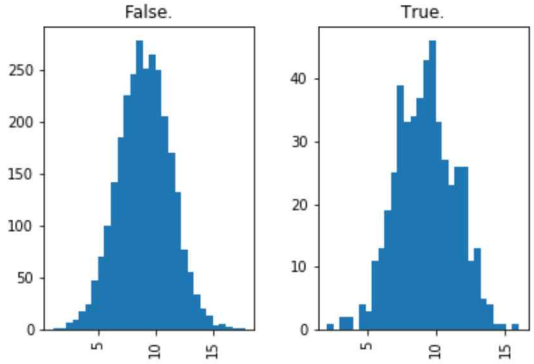
Night Mins



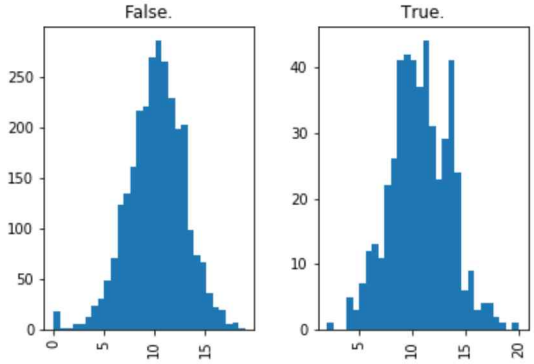
Night Calls



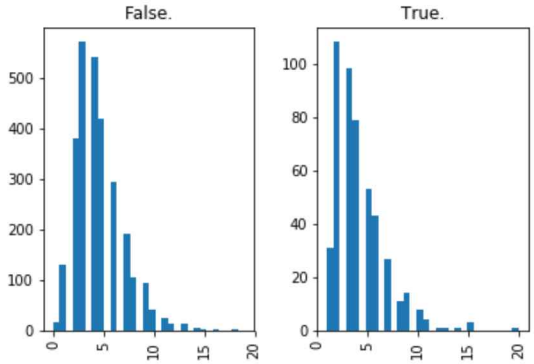
Night Charge



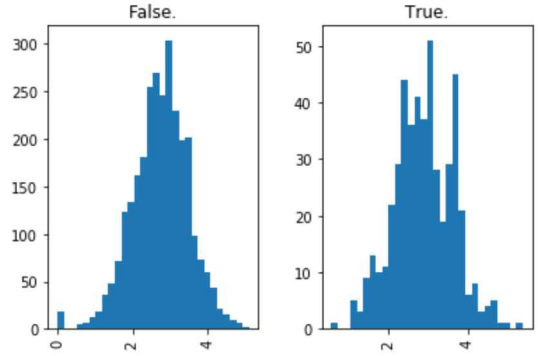
Intl Mins



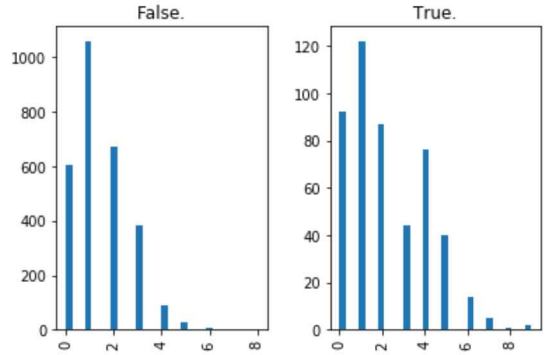
Intl Calls



Intl Charge



CustServ Calls



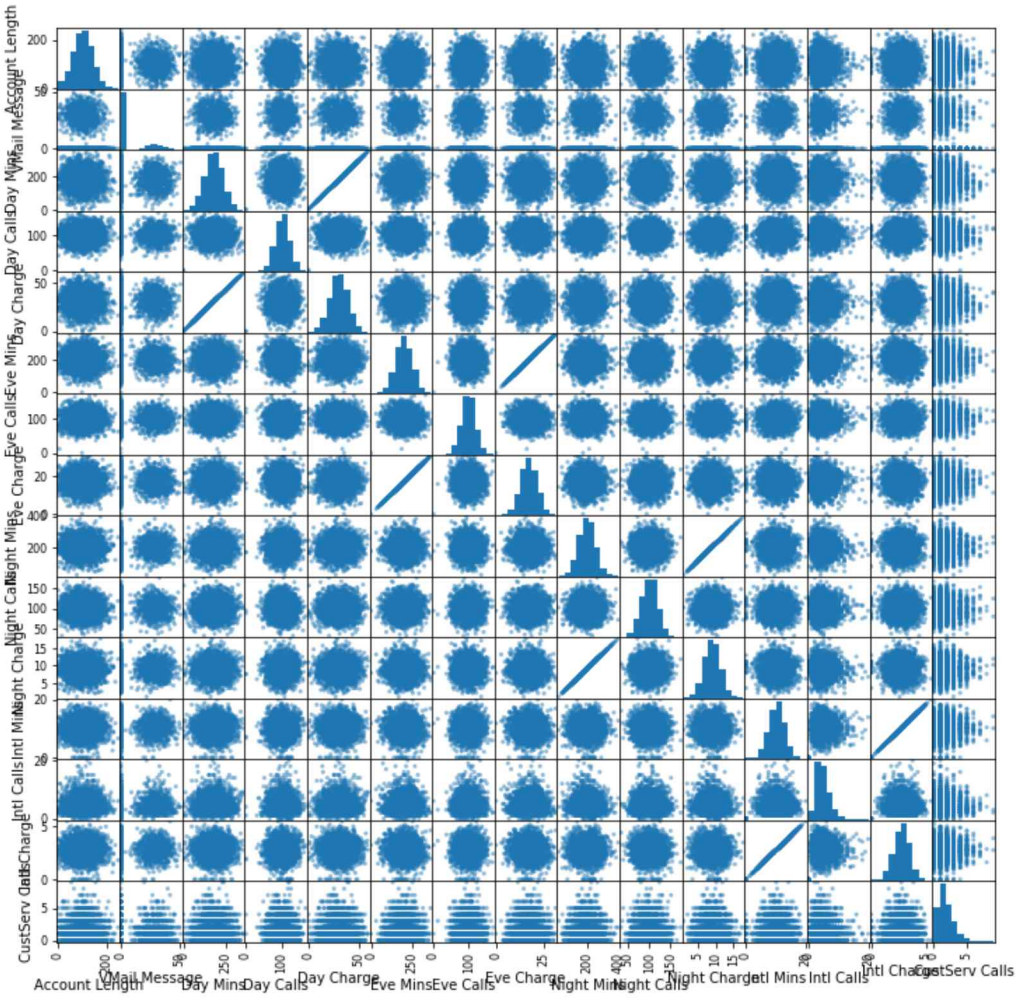
Interestingly we see that churners appear:

- Fairly evenly distributed geographically
- More likely to have an international plan
- Less likely to have a voicemail plan
- To exhibit some bimodality in daily minutes (either higher or lower than the average for non-churners)
- To have a larger number of customer service calls (which makes sense as we'd expect customers who experience lots of problems may be more likely to churn)

In addition, we see that churners take on very similar distributions for features like `Day Mins` and `Day Charge`. That's not surprising as we'd expect minutes spent talking to correlate with charges. Let's dig deeper into the relationships between our features.

In [8]:
```
display(churn.corr())
pd.plotting.scatter_matrix(churn, figsize=(12, 12))
plt.show()
```

	Account Length	VMail Message	Day Mins	Day Calls	Day Charge	Eve Mins	Eve Calls	Eve Charge	Night Mins	Night Calls	Night Charge	Intl Mins	Intl Calls	Intl Charge	CustServ Calls
Account Length	1.000000	-0.004628	0.006216	0.038470	0.006214	-0.006757	0.019260	-0.006745	-0.008955	-0.013176	-0.008960	0.009514	0.020661	0.009546	-0.003796
VMail Message	-0.004628	1.000000	0.000778	-0.009548	0.000776	0.017562	-0.005864	0.017578	0.007681	0.007123	0.007663	0.002856	0.013957	0.002884	-0.013263
Day Mins	0.006216	0.000778	1.000000	0.006750	1.000000	0.007043	0.015769	0.007029	0.004323	0.022972	0.004300	-0.010155	0.008033	-0.010092	-0.013423
Day Calls	0.038470	-0.009548	0.006750	1.000000	0.006753	-0.021451	0.006462	-0.021449	0.022938	-0.019557	0.022927	0.021565	0.004574	0.021666	-0.018942
Day Charge	0.006214	0.000776	1.000000	0.006753	1.000000	0.007050	0.015769	0.007036	0.004324	0.022972	0.004301	-0.010157	0.008032	-0.010094	-0.013427
Eve Mins	-0.006757	0.017562	0.007043	-0.021451	0.007050	1.000000	-0.011430	1.000000	-0.012584	0.007586	-0.012593	-0.011035	0.002541	-0.011067	-0.012985
Eve Calls	0.019260	-0.005864	0.015769	0.006462	0.015769	-0.011430	1.000000	-0.011423	-0.002093	0.007710	-0.002056	0.008703	0.017434	0.008674	0.002423
Eve Charge	-0.006745	0.017578	0.007029	-0.021449	0.007036	1.000000	-0.011423	1.000000	-0.012592	0.007596	-0.012601	-0.011043	0.002541	-0.011074	-0.012987
Night Mins	-0.008955	0.007681	0.004323	0.022938	0.004324	-0.012584	-0.002093	-0.012592	1.000000	0.011204	0.999999	-0.015207	-0.012353	-0.015180	-0.009288
Night Calls	-0.013176	0.007123	0.022972	-0.019557	0.022972	0.007586	0.007710	0.007596	0.011204	1.000000	0.011188	-0.013605	0.000305	-0.013630	-0.012802
Night Charge	-0.008960	0.007663	0.004300	0.022927	0.004301	-0.012593	-0.002056	-0.012601	0.999999	0.011188	1.000000	-0.015214	-0.012329	-0.015186	-0.009277
Intl Mins	0.009514	0.002856	-0.010155	0.021565	-0.010157	-0.011035	0.008703	-0.011043	-0.015207	-0.013605	-0.015214	1.000000	0.032304	0.999993	-0.009640
Intl Calls	0.020661	0.013957	0.008033	0.004574	0.008032	0.002541	0.017434	0.002541	-0.012353	0.000305	-0.012329	0.032304	1.000000	0.032372	-0.017561
Intl Charge	0.009546	0.002884	-0.010092	0.021666	-0.010094	-0.011067	0.008674	-0.011074	-0.015180	-0.013630	-0.015186	0.999993	0.032372	1.000000	-0.009675
CustServ Calls	-0.003796	-0.013263	-0.013423	-0.018942	-0.013427	-0.012985	0.002423	-0.012987	-0.009288	-0.012802	-0.009277	-0.009640	-0.017561	-0.009675	1.000000



We see several features that essentially have 100% correlation with one another. Including these feature pairs in some machine learning algorithms can create catastrophic problems, while in others it will only introduce minor redundancy and bias. Let's remove one feature from each of the highly correlated pairs: Day Charge from the pair with Day Mins, Night Charge from the pair with Night Mins, Intl Charge from the pair with Intl Mins:

In [9]:
```
churn = churn.drop(['Day Charge', 'Eve Charge', 'Night Charge', 'Intl Charge'], axis=1)
```

Now that we've cleaned up our dataset, let's determine which algorithm to use. As mentioned above, there appear to be some variables where both high and low (but not intermediate) values are predictive of churn. In order to accommodate this in an algorithm like linear regression, we'd need to generate polynomial (or bucketed) terms. Instead, let's attempt to model this problem using gradient boosted trees. Amazon SageMaker provides an XGBoost container that we can use to train in a managed, distributed setting, and then host as a real-time prediction endpoint. XGBoost uses gradient boosted trees which naturally account for non-linear relationships between features and the target variable, as well as accommodating complex interactions between features.

Amazon SageMaker XGBoost can train on data in either a CSV or LibSVM format. For this example, we'll stick with CSV. It should:

- Have the predictor variable in the first column
- Not have a header row

But first, let's convert our categorical features into numeric features.

In [10]:
```
model_data = pd.get_dummies(churn)
model_data = pd.concat([model_data['Churn?_True.'], model_data.drop(['Churn?_False.', 'Churn?_True.'], axis=1)], axis=1)
```

And now let's split the data into training, validation, and test sets. This will help prevent us from overfitting the model, and allow us to test the models accuracy on data it hasn't already seen.

In [11]:
```
train_data, validation_data, test_data = np.split(model_data.sample(frac=1, random_state=1729), [int(0.7 * len(model_data)), int(0.9 * len(model_data))])
train_data.to_csv('train.csv', header=False, index=False)
validation_data.to_csv('validation.csv', header=False, index=False)
```

Now we'll upload these files to S3.

In [12]:
```
boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'train/train.csv')).upload_file('train.csv')
boto3.Session().resource('s3').Bucket(bucket).Object(os.path.join(prefix, 'validation/validation.csv')).upload_file('validation.csv')
```

Train

Moving onto training, first we'll need to specify the locations of the XGBoost algorithm containers.

In [13]:
```
from sagemaker.amazon.amazon_estimator import get_image_uri
container = get_image_uri(boto3.Session().region_name, 'xgboost')
```

Then, because we're training with the CSV file format, we'll create `s3_input`s that our training function can use as a pointer to the files in S3.

In [14]:
```
s3_input_train = sagemaker.s3_input(s3_data='s3://{}/{}/train'.format(bucket, prefix), content_type='csv')
s3_input_validation = sagemaker.s3_input(s3_data='s3://{}/{}/validation/'.format(bucket, prefix), content_type='csv')
```

Now, we can specify a few parameters like what type of training instances we'd like to use and how many, as well as our XGBoost hyperparameters. A few key hyperparameters are:

- `max_depth` controls how deep each tree within the algorithm can be built. Deeper trees can lead to better fit, but are more computationally expensive and can lead to overfitting. There is typically some trade-off in model performance that needs to be explored between a large number of shallow trees and a smaller number of deeper trees.
- `subsample` controls sampling of the training data. This technique can help reduce overfitting, but setting it too low can also starve the model of data.
- `num_round` controls the number of boosting rounds. This is essentially the subsequent models that are trained using the residuals of previous iterations. Again, more rounds should produce a better fit on the training data, but can be computationally expensive or lead to overfitting.
- `eta` controls how aggressive each round of boosting is. Larger values lead to more conservative boosting.
- `gamma` controls how aggressively trees are grown. Larger values lead to more conservative models.

More detail on XGBoost's hyperparmeters can be found on their GitHub page (https://github.com/dmlc/xgboost/blob/master/doc/parameter.md).

In [15]:
```
sess = sagemaker.Session()

xgb = sagemaker.estimator.Estimator(container,
                                    role,
                                    train_instance_count=1,
                                    train_instance_type='ml.m4.xlarge',
                                    output_path='s3://{}/{}/output'.format(bucket, prefix),
                                    sagemaker_session=sess)
xgb.set_hyperparameters(max_depth=5,
                        eta=0.2,
                        gamma=4,
                        min_child_weight=6,
                        subsample=0.8,
                        silent=0,
                        objective='binary:logistic',
                        num_round=100)

xgb.fit({'train': s3_input_train, 'validation': s3_input_validation})
```

```
INFO:sagemaker:Creating training-job with name: xgboost-2018-09-07-16-10-45-539
```

```
...................
Arguments: train
[2018-09-07:16:13:42:INFO] Running standalone xgboost training.
[2018-09-07:16:13:42:INFO] File size need to be processed in the node: 0.46mb. Available memory size in the node: 8586.4mb
[2018-09-07:16:13:42:INFO] Determined delimiter of CSV input is ','
[16:13:42] S3DistributionType set as FullyReplicated
[16:13:42] 2333x69 matrix with 160977 entries loaded from /opt/ml/input/data/train?format=csv&label_column=0&delimiter=,
[2018-09-07:16:13:42:INFO] Determined delimiter of CSV input is ','
[16:13:42] S3DistributionType set as FullyReplicated
[16:13:42] 666x69 matrix with 45954 entries loaded from /opt/ml/input/data/validation?format=csv&label_column=0&delimiter=,
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 22 extra nodes, 4 pruned nodes, max_depth=5
[0]#011train-error:0.058723#011validation-error:0.088589
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 18 extra nodes, 4 pruned nodes, max_depth=5
[1]#011train-error:0.051436#011validation-error:0.07958
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 22 extra nodes, 2 pruned nodes, max_depth=5
[2]#011train-error:0.045864#011validation-error:0.073574
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 22 extra nodes, 2 pruned nodes, max_depth=5
[3]#011train-error:0.046292#011validation-error:0.073574
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 22 extra nodes, 2 pruned nodes, max_depth=5
[4]#011train-error:0.047578#011validation-error:0.072072
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 24 extra nodes, 2 pruned nodes, max_depth=5
[5]#011train-error:0.045435#011validation-error:0.072072
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 20 extra nodes, 2 pruned nodes, max_depth=5
[6]#011train-error:0.045006#011validation-error:0.073574
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 16 extra nodes, 6 pruned nodes, max_depth=5
[7]#011train-error:0.043721#011validation-error:0.075075
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 18 extra nodes, 4 pruned nodes, max_depth=5
[8]#011train-error:0.042006#011validation-error:0.072072
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 16 extra nodes, 16 pruned nodes, max_depth=4
[9]#011train-error:0.041149#011validation-error:0.066066
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 14 extra nodes, 12 pruned nodes, max_depth=4
[10]#011train-error:0.039006#011validation-error:0.067568
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 14 extra nodes, 14 pruned nodes, max_depth=4
[11]#011train-error:0.038148#011validation-error:0.064565
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 18 extra nodes, 4 pruned nodes, max_depth=5
[12]#011train-error:0.039006#011validation-error:0.067568
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 10 extra nodes, 8 pruned nodes, max_depth=3
[13]#011train-error:0.038577#011validation-error:0.067568
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 14 extra nodes, 8 pruned nodes, max_depth=5
[14]#011train-error:0.037291#011validation-error:0.066066
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 8 extra nodes, 10 pruned nodes, max_depth=3
[15]#011train-error:0.037291#011validation-error:0.066066
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 12 extra nodes, 4 pruned nodes, max_depth=5
[16]#011train-error:0.036862#011validation-error:0.067568
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 20 extra nodes, 0 pruned nodes, max_depth=5
[17]#011train-error:0.036434#011validation-error:0.064565
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 10 extra nodes, 12 pruned nodes, max_depth=3
[18]#011train-error:0.036005#011validation-error:0.064565
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 14 extra nodes, 6 pruned nodes, max_depth=5
[19]#011train-error:0.034291#011validation-error:0.066066
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 10 extra nodes, 8 pruned nodes, max_depth=4
[20]#011train-error:0.034719#011validation-error:0.066066
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 0 extra nodes, 16 pruned nodes, max_depth=0
[21]#011train-error:0.034291#011validation-error:0.066066
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 6 extra nodes, 14 pruned nodes, max_depth=3
[22]#011train-error:0.034291#011validation-error:0.067568
[23]#011train-error:0.032576#011validation-error:0.069069
[16:13:42] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 12 extra nodes, 6 pruned nodes, max_depth=4
```

Host

Now that we've trained the algorithm, let's create a model and deploy it to a hosted endpoint.

In [16]:
```
xgb_predictor = xgb.deploy(initial_instance_count=1,
                           instance_type='ml.m4.xlarge')
```

```
INFO:sagemaker:Creating model with name: xgboost-2018-09-07-16-14-27-665
INFO:sagemaker:Creating endpoint with name xgboost-2018-09-07-16-10-45-539

--------------------------------------------------------------!
```

Evaluate

Now that we have a hosted endpoint running, we can make real-time predictions from our model very easily, simply by making an http POST request. But first, we'll need to setup serializers and deserializers for passing our `test_data` NumPy arrays to the model behind the endpoint.

In [17]:
```
xgb_predictor.content_type = 'text/csv'
xgb_predictor.serializer = csv_serializer
xgb_predictor.deserializer = None
```

Now, we'll use a simple function to:

1. Loop over our test dataset
2. Split it into mini-batches of rows
3. Convert those mini-batchs to CSV string payloads
4. Retrieve mini-batch predictions by invoking the XGBoost endpoint
5. Collect predictions and convert from the CSV output our model provides into a NumPy array

In [18]:
```
def predict(data, rows=500):
    split_array = np.array_split(data, int(data.shape[0] / float(rows) + 1))
    predictions = ''
    for array in split_array:
        predictions = ','.join([predictions, xgb_predictor.predict(array).decode('utf-8')])

    return np.fromstring(predictions[1:], sep=',')

predictions = predict(test_data.as_matrix()[:, 1:])
```

There are many ways to compare the performance of a machine learning model, but let's start by simply by comparing actual to predicted values. In this case, we're simply predicting whether the customer churned (1) or not (0), which produces a simple confusion matrix.

In [19]:
```
pd.crosstab(index=test_data.iloc[:, 0], columns=np.round(predictions), rownames=['actual'], colnames=['predictions'])
```

Out[19]:

predictions	0.0	1.0
actual		
0	282	4
1	9	39

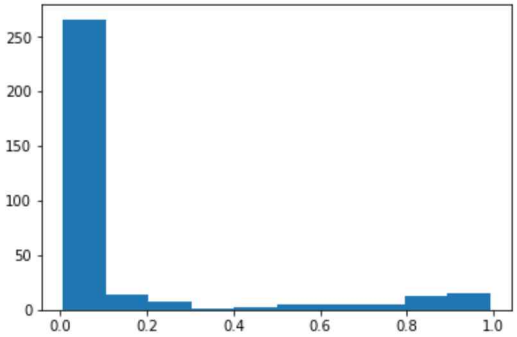
Note, due to randomized elements of the algorithm, you results may differ slightly.

Of the 48 churners, we've correctly predicted 39 of them (true positives). And, we incorrectly predicted 4 customers would churn who then ended up not doing so (false positives). There are also 9 customers who ended up churning, that we predicted would not (false negatives).

An important point here is that because of the `np.round()` function above we are using a simple threshold (or cutoff) of 0.5. Our predictions from `xgboost` come out as continuous values between 0 and 1 and we force them into the binary classes that we began with. However, because a customer that churns is expected to cost the company more than proactively trying to retain a customer who we think might churn, we should consider adjusting this cutoff. That will almost certainly increase the number of false positives, but it can also be expected to increase the number of true positives and reduce the number of false negatives.

To get a rough intuition here, let's look at the continuous values of our predictions.

In [20]:
```
plt.hist(predictions)
plt.show()
```



The continuous valued predictions coming from our model tend to skew toward 0 or 1, but there is sufficient mass between 0.1 and 0.9 that adjusting the cutoff should indeed shift a number of customers' predictions. For example...

In [21]:
```
pd.crosstab(index=test_data.iloc[:, 0], columns=np.where(predictions > 0.3, 1, 0))
```

Out[21]:

col_0	0	1
Churn?_True.		
0	279	7
1	8	40

We can see that changing the cutoff from 0.5 to 0.3 results in 1 more true positives, 3 more false positives, and 1 fewer false negatives. The numbers are small overall here, but that's 6-10% of customers overall that are shifting because of a change to the cutoff. Was this the right decision? We may end up retaining 3 extra customers, but we also unnecessarily incentivized 5 more customers who would have stayed. Determining optimal cutoffs is a key step in properly applying machine learning in a real-world setting. Let's discuss this more broadly and then apply a specific, hypothetical solution for our current problem.

Relative cost of errors

Any practical binary classification problem is likely to produce a similarly sensitive cutoff. That by itself isn't a problem. After all, if the scores for two classes are really easy to separate, the problem probably isn't very hard to begin with and might even be solvable with simple rules instead of ML.

More important, if I put an ML model into production, there are costs associated with the model erroneously assigning false positives and false negatives. I also need to look at similar costs associated with correct predictions of true positives and true negatives. Because the choice of the cutoff affects all four of these statistics, I need to consider the relative costs to the business for each of these four outcomes for each prediction.

Assigning costs

What are the costs for our problem of mobile operator churn? The costs, of course, depend on the specific actions that the business takes. Let's make some assumptions here.

First, assign the true negatives the cost of $0. Our model essentially correctly identified a happy customer in this case, and we don't need to do anything.

False negatives are the most problematic, because they incorrectly predict that a churning customer will stay. We lose the customer and will have to pay all the costs of acquiring a replacement customer, including foregone revenue, advertising costs, administrative costs, point of sale costs, and likely a phone hardware subsidy. A quick search on the Internet reveals that such costs typically run in the hundreds of dollars so, for the purposes of this example, let's assume $500. This is the cost of false negatives.

Finally, for customers that our model identifies as churning, let's assume a retention incentive in the amount of $100. If my provider offered me such a concession, I'd certainly think twice before leaving. This is the cost of both true positive and false positive outcomes. In the case of false positives (the customer is happy, but the model mistakenly predicted churn), we will "waste" the $100 concession. We probably could have spent that $100 more effectively, but it's possible we increased the loyalty of an already loyal customer, so that's not so bad.

Finding the optimal cutoff

It's clear that false negatives are substantially more costly than false positives. Instead of optimizing for error based on the number of customers, we should be minimizing a cost function that looks like this:

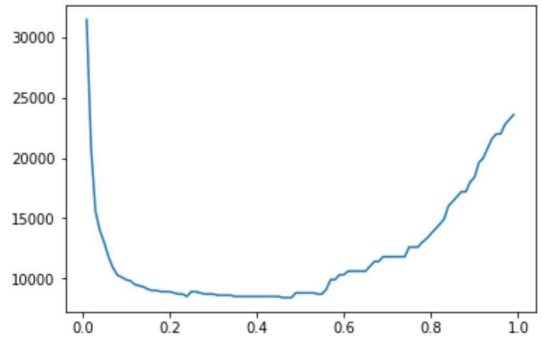
```txt
$500 * FN(C) + $0 * TN(C) + $100 * FP(C) + $100 * TP(C)
```

FN(C) means that the false negative percentage is a function of the cutoff, C, and similar for TN, FP, and TP. We need to find the cutoff, C, where the result of the expression is smallest.

A straightforward way to do this, is to simply run a simulation over a large number of possible cutoffs. We test 100 possible values in the for loop below.

In [22]:
```
cutoffs = np.arange(0.01, 1, 0.01)
costs = []
for c in cutoffs:
    costs.append(np.sum(np.sum(np.array([[0, 100], [500, 100]]) *
                               pd.crosstab(index=test_data.iloc[:, 0],
                                           columns=np.where(predictions > c, 1, 0)))))

costs = np.array(costs)
plt.plot(cutoffs, costs)
plt.show()
print('Cost is minimized near a cutoff of:', cutoffs[np.argmin(costs)], 'for a cost of:', np.min(costs))
```




```
Cost is minimized near a cutoff of: 0.46 for a cost of: 8400
```

The above chart shows how picking a threshold too low results in costs skyrocketing as all customers are given a retention incentive. Meanwhile, setting the threshold too high results in too many lost customers, which ultimately grows to be nearly as costly. The overall cost can be minimized at $8400 by setting the cutoff to 0.46, which is substantially better than the $20k+ I would expect to lose by not taking any action.

Extensions

This notebook showcased how to build a model that predicts whether a customer is likely to churn, and then how to optimally set a threshold that accounts for the cost of true positives, false positives, and false negatives. There are several means of extending it including:

- Some customers who receive retention incentives will still churn. Including a probability of churning despite receiving an incentive in our cost function would provide a better ROI on our retention programs.
- Customers who switch to a lower-priced plan or who deactivate a paid feature represent different kinds of churn that could be modeled separately.
- Modeling the evolution of customer behavior. If usage is dropping and the number of calls placed to Customer Service is increasing, you are more likely to experience churn then if the trend is the opposite. A customer profile should incorporate behavior trends.
- Actual training data and monetary cost assignments could be more complex.
- Multiple models for each type of churn could be needed.

Regardless of additional complexity, similar principles described in this notebook are likely apply.

(Optional) Clean-up

If you're ready to be done with this notebook, please run the cell below. This will remove the hosted endpoint you created and avoid any charges from a stray instance being left on.

In [23]:
```
sagemaker.Session().delete_endpoint(xgb_predictor.endpoint)
```

```
INFO:sagemaker:Deleting endpoint with name: xgboost-2018-09-07-16-10-45-539
```