Churn prediction fundamentals

MACHINE LEARNING FOR MARKETING IN PYTHON

Karolis Urbonas
Head of Analytics & Science, Amazon

What is churn?

- Churn happens when a customer stops buying / engaging
- The business context could be **contractual** or **non-contractual**
- Sometimes churn can be viewed as either **voluntary** or **involuntary**

Types of churn

Main churn typology is based on two business model types:

- Contractual (phone subscription, TV streaming subscription)

- Non-contractual (grocery shopping, online shopping)

Modeling different types of churn

Typically:

- Non-contractual churn is **harder** to define and model, as there's no explicit customer decision
- We will model contractual churn in the telecom business model

Encoding churn

- Typically 1/0, with 1 = Churn, 0 = No Churn
- Could be a string `Churn` / `No Churn` or `Yes` / `No` - best practice to transform as 1 and 0

```
set(telcom['Churn'])
```

```
{0, 1}
```

Exploring churn distribution

```
telcom.groupby(['Churn']).size() / telcom.shape[0] * 100
```

```
Churn
0    73.421502
1    26.578498
dtype: float64
```

Split to training and testing data

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(telcom, test_size = .25)
```

Separate features and target variables

Separate column names by data types

```
target  = ['Churn']
custid  = ['customerID']
cols    = [col for col in telcom.columns if col not in custid + target]
```

Build training and testing datasets

```
train_X = train[cols]
train_Y = train[target]
test_X  = test[cols]
test_Y  = test[target]
```

Let's go practice!

MACHINE LEARNING FOR MARKETING IN PYTHON

Predict churn with logistic regression

MACHINE LEARNING FOR MARKETING IN PYTHON

Karolis Urbonas
Head of Analytics & Science, Amazon

Introduction to logistic regression

- Statistical classification model for binary responses
- Models log-odds of the probability of the target
- Assumes linear relationship between log-odds target and predictors
- Returns coefficients and prediction probability

$$\log_b \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

Modeling steps

1. **Split** data to training and testing
2. **Initialize** the model
3. **Fit** the model on the training data
4. **Predict** values on the testing data
5. **Measure** model performance on testing data

Fitting the model

Import the Logistic Regression classifier

```
from sklearn.linear_model import LogisticRegression
```

Initialize Logistic Regression instance

```
logreg = LogisticRegression()
```

Fit the model on the training data

```
logreg.fit(train_X, train_Y)
```

Model performance metrics

Key metrics:

- **Accuracy** - The % of correctly predicted labels (both Churn and non Churn)
- **Precision** - The % of total model's positive class predictions (here - predicted as Churn) that were correctly classified
- **Recall** - The % of total positive class samples (all churned customers) that were correctly classified

Measuring model accuracy

```python
from sklearn.metrics import accuracy_score

pred_train_Y = logreg.predict(train_X)
pred_test_Y = logreg.predict(test_X)

train_accuracy = accuracy_score(train_Y, pred_train_Y)
test_accuracy = accuracy_score(test_Y, pred_test_Y)

print('Training accuracy:', round(train_accuracy,4))
print('Test accuracy:', round(test_accuracy, 4))
```

```
Training accuracy: 0.8108
Test accuracy: 0.8009
```

Measuring precision and recall

```
from sklearn.metrics import precision_score, recall_score

train_precision = round(precision_score(train_Y, pred_train_Y), 4)
test_precision = round(precision_score(test_Y, pred_test_Y), 4)

train_recall = round(recall_score(train_Y, pred_train_Y), 4)
test_recall = round(recall_score(test_Y, pred_test_Y), 4)

print('Training precision: {}, Training recall: {}'.format(train_precision, train_recall
print('Test precision: {}, Test recall: {}'.format(train_recall, test_recall))
```

```
Training precision: 0.6725, Training recall: 0.5736
Test precision: 0.5736, Test recall: 0.4835
```

Regularization

- Introduces penalty coefficient in the model building phase
- Addresses over-fitting (when patterns are "memorized by the model")
- Some regularization techniques also perform feature selection e.g. L1
- Makes the model more generalizable to unseen samples

L1 regularization and feature selection

- `LogisticRegression` from `sklearn` performs L2 regularization by default
- L1 regularization or also called LASSO can be called explicitly, and this approach performs feature selection by shrinking some of the model coefficients to zero.

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(penalty='l1', C=0.1, solver='liblinear')
logreg.fit(train_X, train_Y)
```

  - `C` parameter needs to be **tuned** to find the optimal value

Tuning L1 regularization

```
C = [1, .5, .25, .1, .05, .025, .01, .005, .0025]
l1_metrics = np.zeros((len(C), 5))
l1_metrics[:,0] = C

for index in range(0, len(C)):
    logreg = LogisticRegression(penalty='l1', C=C[index], solver='liblinear')
    logreg.fit(train_X, train_Y)
    pred_test_Y = logreg.predict(test_X)

    l1_metrics[index,1] = np.count_nonzero(logreg.coef_)
    l1_metrics[index,2] = accuracy_score(test_Y, pred_test_Y)
    l1_metrics[index,3] = precision_score(test_Y, pred_test_Y)
    l1_metrics[index,4] = recall_score(test_Y, pred_test_Y)

col_names = ['C','Non-Zero Coeffs','Accuracy','Precision','Recall']
print(pd.DataFrame(l1_metrics, columns=col_names)
```

Choosing optimal C value

	C	Non-Zero Coeffs	Accuracy	Precision	Recall
0	1.000	22.000	0.800	0.656	0.481
1	0.500	22.000	0.799	0.652	0.481
2	0.250	21.000	0.802	0.660	0.486
3	0.100	20.000	0.803	0.665	0.479
4	0.050	18.000	0.802	0.663	0.479
5	0.025	13.000	0.797	0.658	0.448
6	0.010	5.000	0.790	0.662	0.387
7	0.005	3.000	0.783	0.685	0.301
8	0.003	2.000	0.746	0.833	0.022

Choosing optimal C value

	C	Non-Zero Coeffs	Accuracy	Precision	Recall
0	1.000	22.000	0.800	0.656	0.481
1	0.500	22.000	0.799	0.652	0.481
2	0.250	21.000	0.802	0.660	0.486
3	0.100	20.000	0.803	0.665	0.479
4	0.050	18.000	0.802	0.663	0.479
5	0.025	13.000	0.797	0.658	0.448
6	0.010	5.000	0.790	0.662	0.387
7	0.005	3.000	0.783	0.685	0.301
8	0.003	2.000	0.746	0.833	0.022

Let's run some logistic regression models!

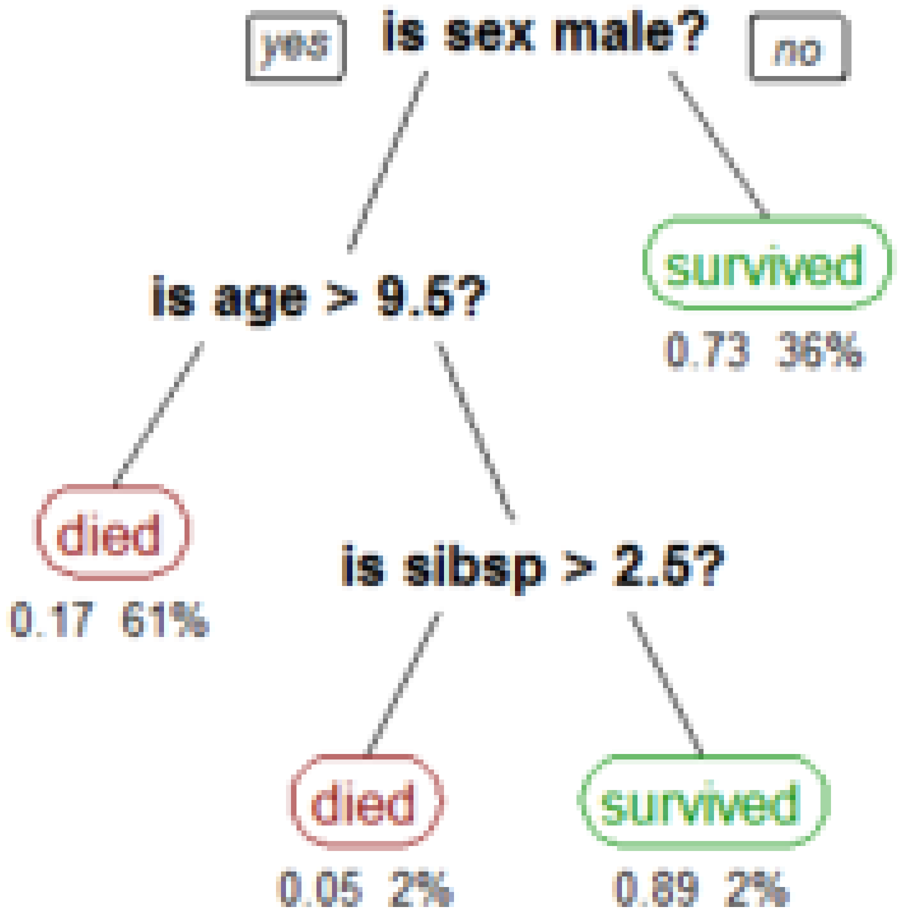
MACHINE LEARNING FOR MARKETING IN PYTHON

Predict churn with decision trees

MACHINE LEARNING FOR MARKETING IN PYTHON

Karolis Urbonas
Head of Analytics & Science, Amazon

Introduction to decision trees



Modeling steps

1. **Split** data to training and testing
2. **Initialize** the model
3. **Fit** the model on the training data
4. **Predict** values on the testing data
5. **Measure** model performance on testing data

Fitting the model

Import the decision tree module

```
from sklearn.tree import DecisionTreeClassifier
```

Initialize the Decision Tree model

```
mytree = DecisionTreeClassifier()
```

Fit the model on the training data

```
treemodel = mytree.fit(train_X, train_Y)
```

Measuring model accuracy

```python
from sklearn.metrics import accuracy_score

pred_train_Y = mytree.predict(train_X)
pred_test_Y = mytree.predict(test_X)

train_accuracy = accuracy_score(train_Y, pred_train_Y)
test_accuracy = accuracy_score(test_Y, pred_test_Y)

print('Training accuracy:', round(train_accuracy,4))
print('Test accuracy:', round(test_accuracy, 4))
```

```
Training accuracy: 0.9973
Test accuracy: 0.7196
```

Measuring precision and recall

```
from sklearn.metrics import precision_score, recall_score

train_precision = round(precision_score(train_Y, pred_train_Y), 4)
test_precision = round(precision_score(test_Y, pred_test_Y), 4)

train_recall = round(recall_score(train_Y, pred_train_Y), 4)
test_recall = round(recall_score(test_Y, pred_test_Y), 4)

print('Training precision: {}, Training recall: {}'.format(train_precision, train_recall
print('Test precision: {}, Test recall: {}'.format(train_recall, test_recall))
```

```
Training precision: 0.9993, Training recall: 0.9906
Test precision: 0.9906, Test recall: 0.4878
```

Tree depth parameter tuning

```
depth_list = list(range(2,15))
depth_tuning = np.zeros((len(depth_list), 4))
depth_tuning[:,0] = depth_list
for index in range(len(depth_list)):
    mytree = DecisionTreeClassifier(max_depth=depth_list[index])
    mytree.fit(train_X, train_Y)
    pred_test_Y = mytree.predict(test_X)
    depth_tuning[index,1] = accuracy_score(test_Y, pred_test_Y)
    depth_tuning[index,2] = precision_score(test_Y, pred_test_Y)
    depth_tuning[index,3] = recall_score(test_Y, pred_test_Y)
col_names = ['Max_Depth','Accuracy','Precision','Recall']
print(pd.DataFrame(depth_tuning, columns=col_names))
```

Choosing optimal depth

	Max_Depth	Accuracy	Precision	Recall
0	2.000	0.774	0.700	0.329
1	3.000	0.774	0.700	0.329
2	4.000	0.774	0.705	0.327
3	5.000	0.780	0.636	0.492
4	6.000	0.778	0.638	0.467
5	7.000	0.776	0.669	0.388
6	8.000	0.769	0.626	0.427
7	9.000	0.764	0.609	0.429
8	10.000	0.747	0.559	0.445
9	11.000	0.751	0.564	0.473
10	12.000	0.725	0.508	0.453
11	13.000	0.727	0.511	0.486
12	14.000	0.721	0.499	0.478

Choosing optimal depth

	Max_Depth	Accuracy	Precision	Recall
0	2.000	0.774	0.700	0.329
1	3.000	0.774	0.700	0.329
2	4.000	0.774	0.705	0.327
3	5.000	0.780	0.636	0.492
4	6.000	0.778	0.638	0.467
5	7.000	0.776	0.669	0.388
6	8.000	0.769	0.626	0.427
7	9.000	0.764	0.609	0.429
8	10.000	0.747	0.559	0.445
9	11.000	0.751	0.564	0.473
10	12.000	0.725	0.508	0.453
11	13.000	0.727	0.511	0.486
12	14.000	0.721	0.499	0.478

Let's build a decision tree!

MACHINE LEARNING FOR MARKETING IN PYTHON

Identify and interpret churn drivers

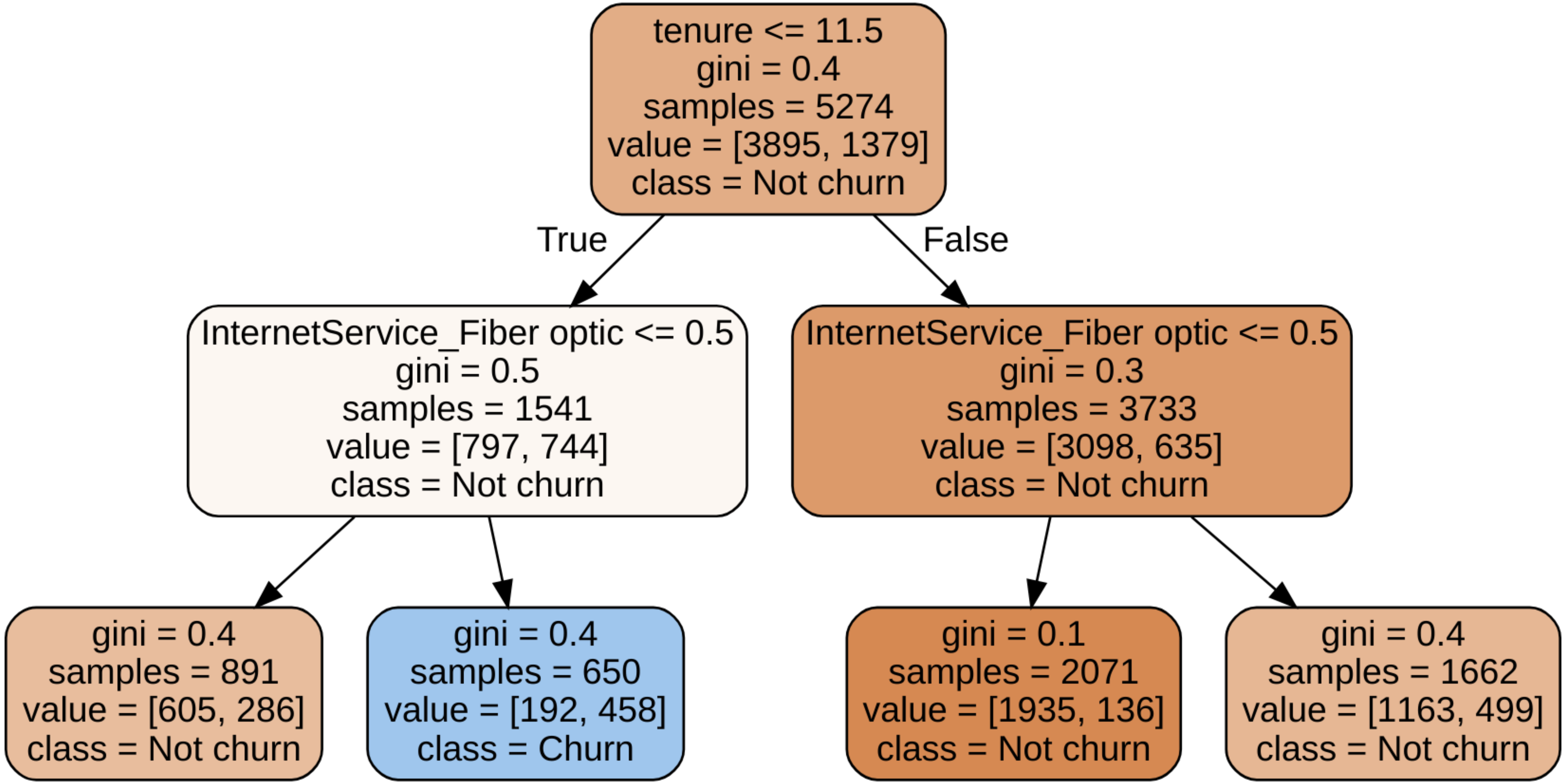
MACHINE LEARNING FOR MARKETING IN PYTHON

Karolis Urbonas
Head of Analytics & Science, Amazon

Plotting decision tree rules

```
from sklearn import tree
import graphviz
exported = tree.export_graphviz(
            decision_tree=mytree,
            out_file=None,
            feature_names=cols,
            precision=1,
            class_names=['Not churn','Churn'],
            filled = True)
graph = graphviz.Source(exported)
display(graph)
```

Interpreting decision tree chart



Logistic regression coefficients

- Logistic regression returns beta coefficients
- Can be interpreted as change in **log-odds** of churn associated with 1 unit increase in the feature

$$\log_b \frac{p}{1-p} = \beta_0 + \beta_1 x_1 + \beta_2 x_2$$

Extracting logistic regression coefficients

- Coefficients can be extracted using `.coef_` method on fitted Logistic Regression instance

```
logreg.coef_
```

```
array([[ 0.        ,  0.09784772,  0.        , -0.03935476, -0.82068131,
        -0.41231806, -0.14319622, -0.01746504, -0.41830733,  0.        ,
         0.        ,  0.07138468,  0.        ,  0.        ,  0.        ,
         0.        , -0.41424363, -0.59539021,  0.        ,  0.18846525,
         0.        , -0.90766135,  0.90151342,  0.        ]])
```

Transforming logistic regression coefficients

- **Log-odds** is difficult to interpret
- Solution - calculate **exponent** of the coefficients
- This gives us the change in **odds** associated with 1 unit increase in the feature

```
coefficients = pd.concat([pd.DataFrame(train_X.columns),
               pd.DataFrame(np.transpose(logit.coef_))],
               axis = 1)
coefficients.columns = ['Feature', 'Coefficient']

coefficients['Exp_Coefficient'] = np.exp(coefficients['Coefficient'])

coefficients = coefficients[coefficients['Coefficient']!=0]
print(coefficients.sort_values(by=['Coefficient']))
```

Meaning of transformed coefficients

	Feature	Coefficient	Exp_Coefficient
21	tenure	-0.908	0.403
4	PhoneService_Yes	-0.821	0.440
17	Contract_Two year	-0.595	0.551
8	TechSupport_Yes	-0.418	0.658
16	Contract_One year	-0.414	0.661
5	OnlineSecurity_Yes	-0.412	0.662
6	OnlineBackup_Yes	-0.143	0.867
3	Dependents_Yes	-0.039	0.961
7	DeviceProtection_Yes	-0.017	0.983
11	PaperlessBilling_Yes	0.071	1.074
1	SeniorCitizen_Yes	0.098	1.103
19	PaymentMethod_Electronic check	0.188	1.207
22	MonthlyCharges	0.902	2.463

Let's practice!

MACHINE LEARNING FOR MARKETING IN PYTHON