Classification with the *sklearn* ML Framework

David Gerbing
The School of Business
Portland State University
gerbing@pdx.edu

Table of Contents

Preliminaries

A classic application of supervised machine learning classification is customer churn. The ability to successfully forecast a customer of a company's services and products about to no longer be a customer allows the company to commit resources to attempt to salvage the relationship.

The following data file contains information on over 7000 customers of a telecom service, including former customers who left the service plan within the last 30 days the data was collected.

Data: http://web.pdx.edu/~gerbing/data/churn_clean.csv

The data has been cleaned according to the analysis for last week, so no need to repeat the cleaning process, or the data exploration.

Misc

```
from datetime import datetime as dt
now = dt.now()
print ("Analysis on", now.strftime("%Y-%m-%d"), "at", now.strftime("%H:%M"))
```

```
Analysis on 2021-08-02 at 00:55
```

```
import os
os.getcwd()
```

```
'/content'
```

```
import pandas as pd
import matplotlib.pyplot as plt
```

Get and Structure Data

a. Read the cleaned data into a data frame, and display its dimensions.

```
d = pd.read_csv('http://web.pdx.edu/~gerbing/data/churn_clean.csv')
#d = pd.read_csv('data/churn_clean.csv')
d.shape
```

```
(7032, 10)
```

b. Display the variable names and the first six rows of data and verify variable types.

```
d.head()
```

	Charges	TotalCharges	MtoM	Paperless	Check	Phone	tenure	Dependents	Inte
0	29.85	29.85	1	1	0	0	1	0	
1	56.95	1889.50	0	0	1	1	34	0	
2	53.85	108.15	1	1	1	1	2	0	
3	42.30	1840.75	0	0	0	0	45	0	
4	70.70	151.65	1	1	0	1	2	0	

```
d.dtypes
```

```
Charges         float64
TotalCharges    float64
MtoM              int64
Paperless         int64
Check             int64
Phone             int64
tenure            int64
Dependents        int64
Internet          int64
Churn             int64
dtype: object
```

All variables are numeric, either float (with decimal digits) or integers. The target variable *Churn* is already scaled as the needed 0/1 integer coding.

c. Create the features and target data structures. Arbitrarily score 1 for Male. Create the lists classes and features for the graph later on.

```
classes = ['Stay', 'Churn']  # for graph
y = d['Churn']
features = ['Charges', 'MtoM', 'Paperless', 'Check',
            'Phone', 'tenure', 'Dependents', 'Internet']
X = d[features]
X.shape
```

```
(7032, 8)
```

d. Do a MinMax transformation to get all data into a 0 to 1 range. Verify.

```
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
mm_scaler = preprocessing.MinMaxScaler()
X = mm_scaler.fit_transform(X)
X = pd.DataFrame(X, columns=features)
X.head()
```

	Charges	MtoM	Paperless	Check	Phone	tenure	Dependents	Internet
0	0.115423	1.0	1.0	0.0	0.0	0.000000	0.0	0.0
1	0.385075	0.0	0.0	1.0	1.0	0.464789	0.0	0.0
2	0.354229	1.0	1.0	1.0	1.0	0.014085	0.0	0.0
3	0.239303	0.0	0.0	0.0	0.0	0.619718	0.0	0.0
4	0.521891	1.0	1.0	0.0	1.0	0.014085	0.0	0.0

Variables *Charges* and *tenure* are now on a scale from 0 to 1.

```
print(X.Charges.min())
print(X.Charges.max())
```

```
0.0
0.9999999999999999
```

```
print(X.tenure.min())
print(X.tenure.max())
```

```
0.0
1.0
```

Grid search: Hyperparameter tuning with cross-validation

```
from sklearn.tree import DecisionTreeClassifier
dt_model = DecisionTreeClassifier(max_depth=5)
```

e. Do a grid search with a 3-fold cross-validation. Search on the following parameters and values: maximum depth with values of 3 and 4, and maximum features with values of 4, 6, and 8.

```
from sklearn.model_selection import KFold

kf3 = KFold(n_splits=3, shuffle=True, random_state=1)
```

```
from sklearn.model_selection import GridSearchCV

params = {'max_depth': [3, 4],
          'max_features': [4, 6, 8]}
grid_search = GridSearchCV(dt_model, param_grid=params, cv=kf3,
                           scoring=('accuracy', 'recall', 'precision'), refit=False,
                           return_train_score=True)
grid_search.fit(X,y)
```

```
GridSearchCV(cv=KFold(n_splits=3, random_state=1, shuffle=True),
             error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=5,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [3, 4], 'max_features': [4, 6, 8]},
             pre_dispatch='2*n_jobs', refit=False, return_train_score=True,
             scoring=('accuracy', 'recall', 'precision'), verbose=0)
```

f. Display all the results of the cross-validation grid search.

```
d_results = pd.DataFrame(grid_search.cv_results_).round(3)
d_results = d_results.drop(['params'], axis='columns')
d_results.transpose()
```

	0	1	2	3	4	5
mean_fit_time	0.011	0.006	0.005	0.005	0.006	0.006
std_fit_time	0.006	0.001	0	0	0	0
mean_score_time	0.008	0.006	0.005	0.005	0.005	0.006
std_score_time	0.003	0	0	0	0	0.002
param_max_depth	3	3	3	4	4	4
param_max_features	4	6	8	4	6	8
split0_test_accuracy	0.765	0.794	0.797	0.79	0.799	0.799
split1_test_accuracy	0.78	0.781	0.781	0.767	0.788	0.788
split2_test_accuracy	0.778	0.778	0.778	0.749	0.784	0.784
mean_test_accuracy	0.774	0.784	0.785	0.769	0.79	0.79
std_test_accuracy	0.007	0.007	0.008	0.017	0.006	0.006
rank_test_accuracy	5	4	3	6	1	1
split0_train_accuracy	0.759	0.779	0.782	0.787	0.79	0.789
split1_train_accuracy	0.791	0.79	0.79	0.778	0.796	0.794
split2_train_accuracy	0.791	0.791	0.791	0.749	0.793	0.795
mean_train_accuracy	0.78	0.787	0.787	0.772	0.793	0.793
std_train_accuracy	0.015	0.005	0.004	0.016	0.002	0.002
split0_test_recall	0.527	0.427	0.404	0.432	0.496	0.496
split1_test_recall	0.363	0.374	0.374	0.22	0.482	0.491
split2_test_recall	0.355	0.355	0.355	0.548	0.358	0.462
mean_test_recall	0.415	0.385	0.378	0.4	0.445	0.483
std_test_recall	0.079	0.031	0.02	0.136	0.062	0.015
rank_test_recall	3	5	6	4	2	1
split0_train_recall	0.511	0.393	0.373	0.433	0.489	0.488
split1_train_recall	0.393	0.399	0.399	0.242	0.484	0.494
split2_train_recall	0.372	0.372	0.372	0.569	0.373	0.477
mean_train_recall	0.425	0.388	0.381	0.415	0.448	0.487
std_train_recall	0.061	0.012	0.012	0.134	0.054	0.007
split0_test_precision	0.549	0.66	0.685	0.643	0.647	0.647
split1_test_precision	0.655	0.653	0.653	0.699	0.634	0.63
split2_test_precision	0.675	0.675	0.675	0.538	0.699	0.643

g. Display the most relevant results, the means.

```
d_summary = d_results[['param_max_depth', 'param_max_features', 'mean_test_accuracy',
                       'mean_test_recall', 'mean_test_precision', 'mean_train_accuracy',
                       'mean_train_recall', 'mean_train_precision']]
d_summary = d_summary.rename(columns= {
                       'param_max_depth': 'depth',
                       'param_max_features': 'features',
                       'mean_test_accuracy': 'test_accuracy',
                       'mean_test_recall': 'test_recall',
                       'mean_test_precision': 'test_precision',
                       'mean_train_accuracy': 'train_accuracy',
                       'mean_train_recall': 'train_recall',
                       'mean_train_precision': 'train_precision'})
d_summary
```

	depth	features	test_accuracy	test_recall	test_precision	train_accuracy
0	3	4	0.774	0.415	0.626	0.780
1	3	6	0.784	0.385	0.662	0.787
2	3	8	0.785	0.378	0.671	0.787
3	4	4	0.769	0.400	0.627	0.772
4	4	6	0.790	0.445	0.660	0.793
5	4	8	0.790	0.483	0.640	0.793

h. Main management goal is to detect churners before they churn, which means focus on avoiding false positives. So, focus on precision. Why is the model with a depth of 3 and 6 features a good model to choose?

The model is parsimonious, not quite the best fit but not much loss in precision from the best-fitting model.

Estimate Model on All Data

i. Given sufficient fit, estimate the model on the full data set as the best estimates are generally obtained with the most data.

```
model = DecisionTreeClassifier(max_features=6, max_depth=3,
                               min_samples_split=5, min_samples_leaf=4)
mf = model.fit(X,y)
```

j. Calculate $\hat{y}$.

```
y_fit = model.predict(X)
```

```
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score
print ('Accuracy:  %.3f' % accuracy_score(y, y_fit))
print ('Recall:  %.3f' % recall_score(y, y_fit))
print ('Precision:  %.3f' % precision_score(y, y_fit))
print ('F1:  %.3f' % f1_score(y, y_fit))

from sklearn.metrics import confusion_matrix
pd.DataFrame(confusion_matrix(y, y_fit))
```

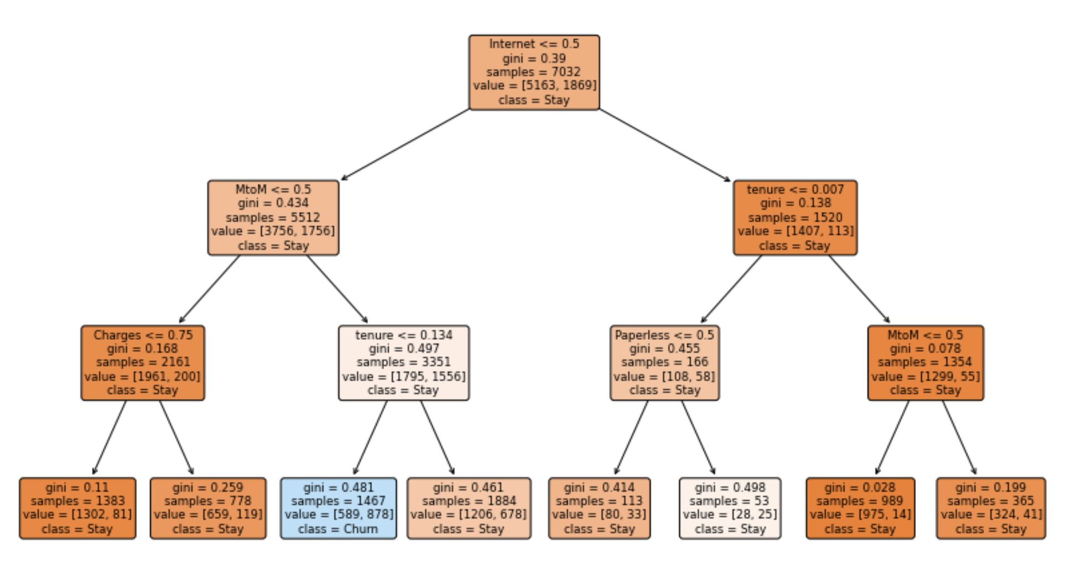
```
Accuracy:  0.775
Recall:  0.470
Precision:  0.599
F1:  0.526
```

	0	1
0	4574	589
1	991	878

Illustrate the Model

k. Draw the tree diagram.

```
from sklearn import tree
plt.figure(figsize=(14,8))
tree.plot_tree(mf, feature_names=features, class_names=classes, rounded=True, filled=True)
plt.savefig('dt_Churn.pdf')
```



l. Focus on specific leaves.

- *Specify the decision rules that does the best job of detecting churners.*

The leaf with the best detection of churning is the only leaf that classifies the customer as a future churner,

MtoM <= 0.5, Charges <= 0.501, Paperless <= 0.5, 753 churners out of 753+400=1153 customers

- *Specify the decision rules that does the worst job in the sense of too many false positives.*

The largest number of false positives, predicted to stay when actually churn:

MtoM <= 0.5, Charges <= 0.501, tenure <= 0.035, 261 churners out of 321+261=582 customers

Apply the Model

m. Apply the model to a person who has ...

- Charges = 45
- MtoM = 29
- Paperless = 1
- Check = 1
- Phone = 0
- tenure = 0
- Dependents = 1
- Internet = 0

```
print(X.columns)
```

```
Index(['Charges', 'MtoM', 'Paperless', 'Check', 'Phone', 'tenure',
       'Dependents', 'Internet'],
      dtype='object')
```

```
#X_new = [[45, 29, 1,1,0,0,1,0]]
X_new = [[45, 29, 1,1,0,0,1,0]]
y_new = model.predict(X_new)
print("Predicted group membership:", y_new)
y_prob = model.predict_proba(X_new)
print(round(y_prob[0,1], 3))
```

```
Predicted group membership: [1]
0.599
```