CASE STUDY - Unsupervised Learning

In [1]:
```
import os
import joblib
import time
import numpy as np
import pandas as pd
import scipy.stats as stats
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.compose import ColumnTransformer
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.impute import SimpleImputer
from sklearn.cluster import KMeans, SpectralClustering
from sklearn.preprocessing import StandardScaler, OneHotEncoder, OrdinalEnc
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, f1_score
from sklearn.metrics import silhouette_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.mixture import BayesianGaussianMixture
from sklearn.svm import SVC
import imblearn.pipeline as pl
from imblearn.pipeline import make_pipeline
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE, SVMSMOTE


plt.style.use('seaborn')
%matplotlib inline

DATA_DIR = os.path.join("..","data")
```

```
Using TensorFlow backend.
```

Synopsis

> We are now going to predict customer retention. There are many models and many transforms to consider. Use your knowledge of pipelines and functions to ensure that your code makes it easy to compare and iterate over.

> Marketing has asked you to make a report on customer retention. They would like you to come up with information that can be used to improve current marketing strategy efforts. The current plan is for marketing at AAVAiL to collect more features on subscribers and they would like to use your report as a proof-of-concept in order to get buyin for this effort.

Outline

1. Create a churn prediction baseline model
2. Use clustering as part of your prediction pipeline
3. Run and experiment to see if re-sampling techniques improve your model

Data

Here we load the data as we have already done.

`aavail-target.csv`

In [2]:
```
df = pd.read_csv(os.path.join(DATA_DIR, r"aavail-target.csv"))

## pull out the target and remove uneeded columns
_y = df.pop('is_subscriber')
y = np.zeros(_y.size)
y[_y==0] = 1
df.drop(columns=['customer_id', 'customer_name'], inplace=True)
df.head()
```

Out[2]:

	country	age	subscriber_type	num_streams
0	united_states	21	aavail_premium	23
1	singapore	30	aavail_unlimited	12
2	united_states	21	aavail_premium	22
3	united_states	20	aavail_basic	19
4	singapore	21	aavail_premium	23

QUESTION 1

Using the train_test_split() function, create a stratified train test split of the data

In [3]:
```
## YOUR CODE HERE
X_train, X_test, y_train, y_test = train_test_split(df, y, test_size=0.25,
```

QUESTION 2

Create a baseline model. We are going to test whether clustering followed by a model improves the results. Then, we will test whether re-sampling techniques provide improvements. Use a pipeline or another method, but create a baseline model given the data. Here is the ColumnTransformer we have used before:

In [4]:
```
## preprocessing pipeline
numeric_features = ['age', 'num_streams']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())])

categorical_features = ['country', 'subscriber_type']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('encod', OrdinalEncoder())])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])
```

In [8]:
```
# YOUR CODE HERE


# Create an instance of a binary classifier.
clf = RandomForestClassifier()

# Create a pipeline that binds the preprocessing transformer and the classi
pipe = Pipeline(steps=[('pre', preprocessor),
                       ('rf', clf)])

# Here we apply a grid search to optimize the hyperparamters of the classif
param_grid = {
    'rf__n_estimators': [20, 50, 100, 150],
    'rf__max_depth': [4, 5, 6, 7, 8],
    'rf__criterion': ['gini', 'entropy']
}
grid = GridSearchCV(pipe, param_grid=param_grid, cv=3, n_jobs=-1, scoring='

# Fit the pipeline to the training data.
grid.fit(X_train, y_train)
best_params = grid.best_params_

# Predict the dependent variable of the test set.
y_pred = grid.predict(X_test)

# Print the f1_score of the prediction.
print("f1_score", round(f1_score(y_test, y_pred, average='binary'), 3))
```

```
f1_score 0.594
```

QUESTION 3

The next objective is to create version of the classifier that uses identified clusters. Here is a class to get you started. It is a transformer like those that we have been working with and there is an example of how to use it just below. In this example 4 clusters were specified and their one-hot encoded versions were appended to the feature matrix. Now using pipelines and/or functions compare the performance using cluster profiling as part of your matrix to the baseline. You may compare multiple models and multiple clustering algorithms here.

In [9]:
```
class KmeansTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, n_clusters=4):
        self.n_clusters = n_clusters
        self.km = KMeans(n_clusters=self.n_clusters, n_init=20)

    def transform(self, X, *_):
        labels = self.km.predict(X)
        return np.hstack((X, labels.reshape(-1, 1)))

    def fit(self, X, y=None, *_):
        self.km.fit(X)
        labels = self.km.predict(X)
        self.silhouette_score = round(silhouette_score(X, labels, metric='m
        return self

class GmmTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, n_clusters=4):
        self.n_clusters = n_clusters
        self.gmm = BayesianGaussianMixture(n_components=self.n_clusters, co
                                           max_iter=500, n_init=10, warm_st
    def transform(self, X,*_):
        probs = self.gmm.predict_proba(X) + np.finfo(float).eps
        return np.hstack((X, probs))

    def fit(self, X, y=None, *_):
        self.gmm.fit(X)
        labels = self.gmm.predict(X)
        self.silhouette_score = round(silhouette_score(X, labels, metric='m
        return self



## example for kmeans
preprocessor.fit(X_train)
X_train_pre = preprocessor.transform(X_train)
kt = KmeansTransformer(4)
kt.fit(X_train_pre)
X_train_kmeans = kt.transform(X_train_pre)
print(X_train_pre.shape)
print(X_train_kmeans.shape)

## example for GMM
preprocessor.fit(X_train)
X_train_pre = preprocessor.transform(X_train)
gt = GmmTransformer(4)
gt.fit(X_train_pre)
X_train_gmm = gt.transform(X_train_pre)
print(X_train_pre.shape)
print(X_train_gmm.shape)
```

```
(750, 4)
(750, 5)
(750, 4)
(750, 8)
```

In [10]:
```
## YOUR CODE HERE
# This cell might take several minutes to run


def run_clustering_pipeline(umodel):
    """
    This function evaluates different Pipelines comprised of the preprocess
    a clustering transformer and a classifier estimator.
    INPUT : The name of the clustering transformer : 'gmm' or 'kmeans'
    OUTPUT : The list of f1_scores of the pipeline on the test set for the
    """

    fscores = [] # this list will store the f1_score of the different model
    for n_clusters in np.arange(3, 8):

        # Create an instance of a binary classifier (The same as the one yo
        estimator = RandomForestClassifier()
        param_grid = {
                        'n_estimators': [20, 50, 100, 150],
                        'max_depth': [4, 5, 6, 7, 8],
                        'criterion': ['gini', 'entropy']
                    }
        clf = GridSearchCV(estimator, param_grid=param_grid, cv=3, n_jobs=-

        if umodel == 'gmm':
            # Create an instance of the Gmm transformer with n_clusters clu
            cluster = GmmTransformer(n_clusters)
        elif umodel == 'kmeans':
            # Create an instance of the Kmean transformer with n_clusters c
            cluster = KmeansTransformer(n_clusters)
        else:
            raise Exception("invalid unsupervised learning model")

        # Create a Pipeline that binds the preprocessing transformer, the c
        pipe = Pipeline(steps=[('pre', preprocessor),
                               ('clustering', cluster),
                               ('classifier', clf)])

        # Fit the pipeline on training set
        pipe.fit(X_train, y_train)
        # Predict the test set
        y_pred = pipe.predict(X_test)
        # Compute the f1 score and add this score to the fscores list.
        score = round(f1_score(y_test, y_pred, average='binary'), 3)
        fscores.append(score)

    return fscores

## Run the different iteration of the model
cp_results = {}
cp_results['kmeans'] = run_clustering_pipeline('kmeans')
cp_results['gmm'] = run_clustering_pipeline('gmm')

## Display table of results
df_cp = pd.DataFrame(cp_results)
df_cp["n_clusters"] = [str(i) for i in np.arange(3,8)]
df_cp.set_index("n_clusters", inplace=True)
```

```
df_cp.head(n=10)
```

Out[10]:

n_clusters	kmeans	gmm
3	0.592	0.571
4	0.589	0.587
5	0.605	0.554
6	0.598	0.600
7	0.592	0.591

SOLUTION NOTE

This is a fairly small dataset with a small number of features. The utility of adding clustering to the pipeline is generally more apparent in higher dimensional data sets.

QUESTION 4

Run an experiment to see if you can you improve on your workflow with the addition of re-sampling techniques? For instance, you can copy the structure of the function created in the previous question and add a re-sampling transformer to the pipeline.

In [11]:
```
## YOUR CODE HERE
# This cell might take several minutes to run

def run_clustering_pipeline(umodel):
    """
    This function evaluates different Pipelines constituated of the preproc
    a clustering transformer, a re-sampling transformer and a classifier es
    INPUT : The name of the clustering transformer : 'gmm' or 'kmeans'
    OUTPUT : The list of f1_scores of the pipeline on the test set for the
    """

    fscores = []  # this list will store the f1_score of the different mode
    for n_clusters in np.arange(3,8):

        # Create an instance of a binary classifier (The same as the one yo
        estimator = RandomForestClassifier()
        param_grid = {
                    'n_estimators': [20, 50, 100, 150],
                    'max_depth': [4, 5, 6, 7, 8],
                    'criterion': ['gini', 'entropy']
                    }
        clf = GridSearchCV(estimator, param_grid=param_grid, cv=3, n_jobs=-

        if umodel == 'gmm':
            # Create an instance of the Gmm transformer with n_clusters clu
            cluster = GmmTransformer(n_clusters)
        elif umodel == 'kmeans':
            # Create an instance of the Kmean transformer with n_clusters c
            cluster = KmeansTransformer(n_clusters)
        else:
            raise Exception("invalid unsupervised learning model")


        # Create a Pipeline that binds the preprocessing transformer, the c
        # the re-sampling transformer and the classifier
        pipe = pl.Pipeline(steps=[('pre', preprocessor),
                                  ('clustering', cluster),
                                  ('smote', SMOTE(random_state=42)),
                                  ('classifier', clf)])

        # Fit the pipeline on training set
        pipe.fit(X_train,y_train)
        # Predict the test set
        y_pred = pipe.predict(X_test)
        # Compute the f1 score and add this score to the fscores list.
        score = round(f1_score(y_test, y_pred,average='binary'),3)
        fscores.append(score)

    return(fscores)

## Run the different iteration of the model
cp_results = {}
cp_results['kmeans'] = run_clustering_pipeline('kmeans')
cp_results['gmm'] = run_clustering_pipeline('gmm')


## Display table of results
```

```
df_cp = pd.DataFrame(cp_results)
df_cp["n_clusters"] = [str(i) for i in np.arange(3,8)]
df_cp.set_index("n_clusters",inplace=True)
df_cp.head(n=10)
```

Out[11]:

n_clusters	kmeans	gmm
3	0.612	0.612
4	0.603	0.603
5	0.601	0.614
6	0.615	0.625
7	0.599	0.626

SOLUTION NOTE

The inclusion of customer profiles does not significantly improve the overall model performance pipeline for either model. There may be some minor improvement depending on the random seed, but since it does not degrade model performance either it can be useful in the context of marketing. The clusters are customer profiles that are tied to predictive performance. The re-sampling does help the random forest classifiers obtain similar performance results to SVM in this case.