Churn Prediction

The code uses a famous dataset about the purchase behavior of customers and then predicts if they are supposed to repeat their purchase or not. The code uses a *Logistic Regression* and a *Decision Tree* model to do Machine Learning.

The input features are: 'gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure', 'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges', 'TotalCharges'.

Train / test data sets proportions: 70 / 30

Churn percentage: 73.5%

Accuracy Rates

Logistic Regression: 75% (selected features: 73%) Decision Tree: 68% (selected features: 67%)

In [6]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
from pylab import rcParams
%matplotlib inline

from os.path import join
# import xlsxwriter

import pandas as pd
import numpy as np
import sklearn
from sklearn import linear_model
from sklearn.utils import shuffle
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import LabelEncoder

from sklearn import model_selection
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn import ensemble
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier

from sklearn import svm

# Adding libraries needed for plotting the significance of variables in prediction of target
from sklearn.feature_selection import SelectFromModel
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.feature_selection import RFE
from matplotlib import pyplot as plt

# Adding libraries needed to select most significant properties in prediction model (Logist Regression)
from sklearn.feature_selection import RFECV
```

```
# importing one hot encoder from sklearn
from sklearn.preprocessing import OneHotEncoder

import random

# import statsmodels.api as sm

# import graphviz
# import pydotplus
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree.export import export_text

from inspect import getmembers
```

In [7]:
```
def preparing_data(input_file, component, component_list, init_list, test_size):

    # Loading the CSV with pandas
    # at the same time reading categorical data

    data = pd.read_csv(input_file, dtype = {'gender': 'category'})
    new_df = data[init_list]

    # Exploration and Feature Selection
    # Data to plot
    sizes = data['Churn'].value_counts(sort = True)
    colors = ["grey","purple"]
    rcParams['figure.figsize'] = 5, 5
    labels = 'Churn = Yes', 'Churn = No'
    # Plot
    plt.pie(sizes, colors=colors, autopct='%1.1f%%', shadow=True, startangle=270, labels = labels)
    plt.title('Percentage of Churn in Dataset')
    plt.show()

    # Data Preparation and Feature Engineering
    # 1 - Dropping irrelevant data
    data.drop(['customerID'], axis=1, inplace=True)

    # 2. Missing Values
    nan_value = float("NaN")
    data.replace("", nan_value, inplace=True)
    data.dropna(subset = ["TotalCharges"], inplace=True)

    # 3. Converting Numerical Features From Object
    data['TotalCharges'] = pd.to_numeric(data['TotalCharges'])

    pos = data.loc[(data[component] == 'No')]
    neg = data.loc[(data[component] == 'Yes')]

    # Balancing data
    random.seed(0)
    if len(pos) <= len(neg):
        a = random.sample(range(len(neg)), len(pos)) # non duplicate indices
        final = neg.iloc[a, :]
        final = pd.concat([pos, final])
    else:
        a = random.sample(range(len(pos)), len(neg)) # non duplicate indices
```

```
        final = pos.iloc[a, :]
        final = pd.concat([neg, final])

    x = np.array(final.drop([component], 1))
    y = np.array(final[component])

    # Split the data in train and test data
    np.random.seed(0) # the "train_test_split" function uses "np.random.seed" to split the data
    x_train, x_test, y_train, y_test = sklearn.model_selection.train_test_split(x, y, test_size = test_size)

    x_df_train = pd.DataFrame({'gender': x_train[:, 0], 'SeniorCitizen': x_train[:, 1], 'Partner': x_train[:,
2], \
                               'Dependents': x_train[:, 3], 'tenure': x_train[:, 4], 'PhoneService': x_train
[:, 5], \
                               'MultipleLines': x_train[:, 6], 'InternetService': x_train[:, 7], \
                               'OnlineSecurity': x_train[:, 8], 'OnlineBackup': x_train[:, 9], \
                               'DeviceProtection': x_train[:, 10], 'TechSupport': x_train[:, 11], \
                               'StreamingTV': x_train[:, 12], 'StreamingMovies': x_train[:, 13], 'Contract':
x_train[:, 14], \
                               'PaperlessBilling': x_train[:, 15], 'PaymentMethod': x_train[:, 16], \
                               'MonthlyCharges': x_train[:, 17], 'TotalCharges': x_train[:, 18]})
    x_df_test = pd.DataFrame({'gender': x_test[:, 0], 'SeniorCitizen': x_test[:, 1], 'Partner': x_test[:, 2],
\
                              'Dependents': x_test[:, 3], 'tenure': x_test[:, 4], 'PhoneService': x_test[:, 5
], \
                              'MultipleLines': x_test[:, 6], 'InternetService': x_test[:, 7], 'OnlineSecurit
y': x_test[:, 8], \
                              'OnlineBackup': x_test[:, 9], 'DeviceProtection': x_test[:, 10], 'TechSupport':
x_test[:, 11], \
                              'StreamingTV': x_test[:, 12], 'StreamingMovies': x_test[:, 13], 'Contract': x_t
est[:, 14], \
                              'PaperlessBilling': x_test[:, 15], 'PaymentMethod': x_test[:, 16], \
                              'MonthlyCharges': x_test[:, 17], 'TotalCharges': x_test[:, 18]})

    return x_train, x_test, y_train, y_test, x, y, final.drop([component], 1), x_df_train, x_df_test

def select_top(x_train, y_train, y_test, df_train, df_test, component, algorithm, title):
    # Feature selection

    sel = SelectFromModel(algorithm)
    sel.fit(x_train, y_train)

    # print('df_train.columns', df_train.columns)
```

```
    # print('sel.get_support', sel.get_support())
    # print('sel.estimator.coef', sel.estimator.coef_[0])

    # Plotting and sorting the first 10 significant features
    df_columns = pd.DataFrame(df_train.columns)
    if title == 'Logistic Regression' or title == 'SVM':
        df_scores = pd.DataFrame(abs(sel.estimator.coef_[0]))
        df_coef = pd.DataFrame(sel.estimator.coef_[0])
    else:
        df_scores = pd.DataFrame(sel.estimator.feature_importances_)
        df_coef = pd.DataFrame(sel.estimator.feature_importances_)
    df_sign = pd.DataFrame(sel.get_support())
    featureScores = pd.concat([df_columns, df_scores, df_sign, df_coef], axis=1)
    featureScores.columns = ['property', 'Score', 'Support', 'Coef']
    featureScores.nlargest(10, 'Score').plot.barh(x='property', y='Score')
    plt.show()
    featureScores = featureScores[featureScores.Support == True]

    columns = []; signs = []; coefs = [];
    selected_feat = df_train.columns[(sel.get_support())]
    for i in featureScores.nlargest(len(selected_feat), 'Score')['property']: columns.append(i)
    for i in featureScores.nlargest(len(selected_feat), 'Score')['Coef']: coefs.append(i)
    print(columns, coefs)

    print('most siginifant columns:', columns)
    new_train = df_train[selected_feat]; new_test = df_test[selected_feat]

    if title == 'Logistic Regression' or title == 'SVM':
        indices = np.argsort(abs(sel.estimator.coef_))[::-1][:len(new_train.columns)]
        initial_importances = sel.estimator.coef_
    else:
        indices = np.argsort(sel.estimator.feature_importances_)[::-1][:len(new_train.columns)]
        initial_importances = sel.estimator.feature_importances_

    algorithm = algorithm.fit(new_train, y_train)
    result_top = algorithm.score(new_test, y_test)
    print(title + ' performance for predicting "' + component + '" is {0:.0%} (selected features)'.format(res
ult_top))
    if title == 'Logistic Regression' or title == 'SVM':
        return result_top, columns, coefs
    else:
        # Sorting selected features
        # algorithm.fit(x_train, y_train)
```

```
        # indices = np.argsort(algorithm.feature_importances_)[::-1][:len(new_train.columns)]
        selected_feat = df_train.columns[indices]
        # print(selected_feat, indices)

        columns = []
        for col in selected_feat: columns.append(col)
        """
        if title == 'Decision Tree' or title == 'Decision Tree (Entropy)':
            r = export_text(algorithm, feature_names=columns)
            # print(r)
            # print( getmembers( algorithm.tree_.children_left) )
            zip(df_train.columns[algorithm.tree_.feature], algorithm.tree_.threshold, algorithm.tree_.childre
n_left, \
            algorithm.tree_.children_right)

            dot_data = StringIO()
            tree.export_graphviz(algorithm, out_file=dot_data, filled=True, rounded=True, special_characters=
True, \
                                 feature_names = columns, class_names=['0','1'])
            graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
            graph.write_pdf(title + ' ' + component + ".pdf")
            Image(graph.create_png())
        """
        # return result_top, df_train.columns[indices], algorithm.feature_importances_[indices]
        sorted_importances = list(initial_importances)[:len(new_train.columns)]
        sorted_importances.sort(reverse = True)
        # print('shahab', df_train.columns, indices, initial_inportances, selected_feat, sorted_importances)
        return result_top, columns, sorted_importances

def predict_regression(x_train, x_test, y_train, y_test, component, df, df_train, df_test):
    # Creating the model
    # Logistic Regression
    # Using "LogisticRegression" function from "scikit-learn" library from Python. We have set below paramete
rs for:
    # multi_class='ovr' : means we have select our training dataset based on One vs Rest
    # solver = "liblinear", because our dataset is small

    # Scale your data
    scaler = StandardScaler(); scaler.fit(df_train)
    X_scaled = pd.DataFrame(scaler.transform(df_train), columns = df_train.columns)
    scaler_test = StandardScaler(); scaler_test.fit(df_test)
    X_scaled_test = pd.DataFrame(scaler_test.transform(df_test), columns = df_test.columns)
```

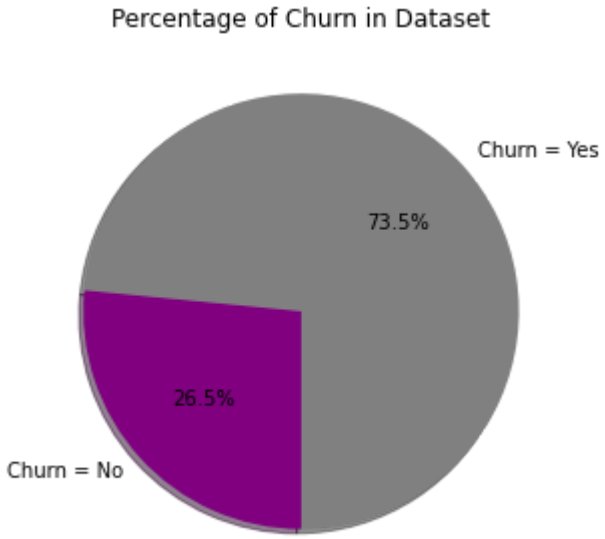
```
    # Results for all columns
    logreg = LogisticRegression(C=1e5, solver='liblinear', multi_class='ovr', max_iter = 1000)
    logreg.fit(X_scaled, y_train)
    result_all = logreg.score(X_scaled_test, y_test)
    print('Logistic Regression (accuracy rate) for predicting "' + component + '" is {0:.0%} all columns'.for
mat(result_all))

    logreg_top = LogisticRegression(C=1e5, solver='liblinear', multi_class='ovr', max_iter = 1000)
    logreg_top.fit(X_scaled, y_train)

    result_top, col_top, sign_top = \
    select_top(x_train, y_train, y_test, df_train, df_test, component, logreg_top, 'Logistic Regression')

    return result_all, result_top, col_top, sign_top

def predict_dt(x_train, x_test, y_train, y_test, component, df, df_train, df_test):

    # Decision Tree
    clf = DecisionTreeClassifier(random_state=0)
    clf = clf.fit(x_train, y_train)

    # tree.plot_tree(clf)
    """
    dot_data = tree.export_graphviz(clf, out_file=None, feature_names=df_train.columns, class_names=componen
t, \
                                    filled=True, rounded=True, special_characters=True)
    graph = graphviz.Source(dot_data)
    graph.render()

    """


    # print(clf.predict(x_test))
    result_all = clf.score(x_test, y_test)
    print('Decision Tree performance for predicting "' + component + '" is {0:.0%} (all columns)'.format(resu
lt_all))

    # result_top, col_top, coef = select_top(x_train, y_train, y_test, df_train, df_test, component, clf, 'De
cision Tree')
    result_top, col_top, coef = select_top(x_train, y_train, y_test, df_train, df_test, component, clf, 'Deci
sion Tree')
    print(col_top)
    print(coef)
```

```
    return result_all, result_top, col_top, coef
```

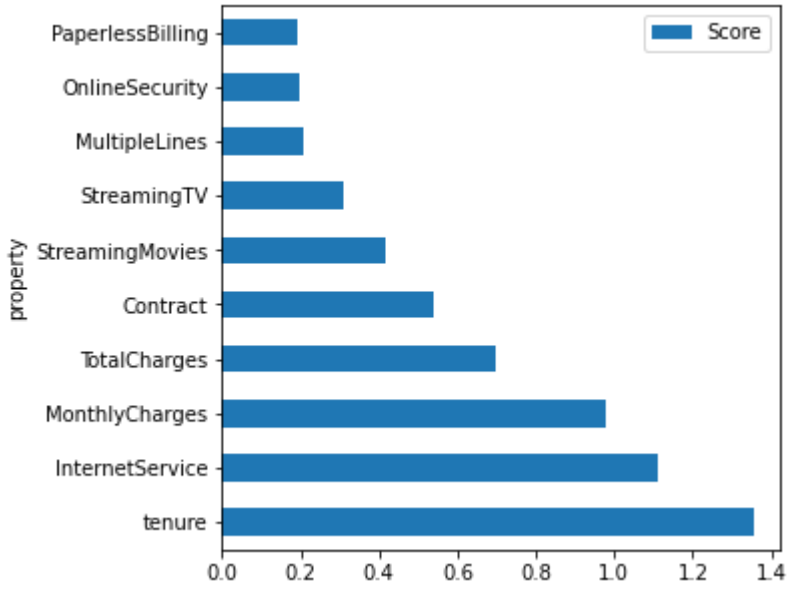
In [8]:

```
my_path = r'C:\Aarhus\Internship\VanHack\churn'
input_file = 'Telco-Customer-Churn.csv'
predict_file = 'predict_churn_results.xlsx'
header = ['method', 'Accuracy Rate (all)', 'Accuracy Rate (top)', 'Significant Features']

freq_file = 'freq_context.csv'
freq_header = ['method', 'Feature']
freq_list = []

component = ['Churn']

init_list = ['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure', 'PhoneService', 'MultipleLines', \
             'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'Streami
ngTV', \
             'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges', 'TotalChar
ges', \
             'Churn']
predict_list = []
test_size = 0.3

for c in component:
    x_train, x_test, y_train, y_test, x, y, df, df_train, df_test = \
        preparing_data(join(my_path, input_file), c, component, init_list, test_size)

    # Logistic Regression
    acc, acc_top, cols, coef = predict_regression(x_train, x_test, y_train, y_test, c, df, df_train, df_test)
    row = []; row.append(c); row.append('Logistic Regression'); row.append(acc); row.append(acc_top)
    col_score = []; col_score_str = ''
    for i in range(len(cols)):
        temp = []; temp.append(cols[i]); col_score_str += cols[i] + ': '; temp.append(coef[i])
        col_score_str += str(round(coef[i] * 100, 2)).format(1.0/3.0) + ', '; col_score.append(temp); del tem
p
        temp = []; temp.append('Logistic Regression'); temp.append(cols[i]); freq_list.append(temp); del temp
    row.append(col_score_str[:-2]); predict_list.append(row); del row

    # Decision Tree
    acc, acc_top, cols, coef = predict_dt(x_train, x_test, y_train, y_test, c, df, df_train, df_test)
    row = []; row = []; row.append(c); row.append('Decision Tree'); row.append(acc); row.append(acc_top)
    col_score = []; col_score_str = ''
    for i in range(len(cols)):
        temp = []; temp.append(cols[i]); col_score_str += cols[i] + ': '; temp.append(coef[i])
        col_score_str += str(round(coef[i] * 100, 2)).format(1.0/3.0) + '%, '; col_score.append(temp); del te
```

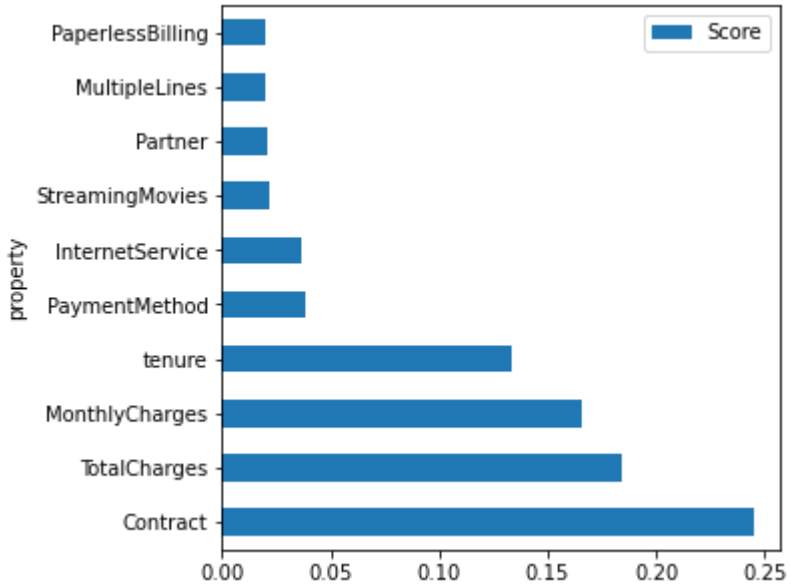
```
mp
        temp = []; temp.append('Decision Tree'); temp.append(cols[i]); freq_list.append(temp); del temp
    row.append(col_score_str[:-2]); predict_list.append(row); del row

    temp_list = ['']
    predict_list.append(temp_list)
```



Logistic Regression (accuracy rate) for predicting "Churn" is 75% all columns



['InternetService', 'Contract', 'StreamingMovies', 'StreamingTV', 'OnlineSecurity', 'PaperlessBilling', 'Seni
orCitizen', 'PhoneService'] [1.1102527328407639, -0.5368584025345132, 0.41576926290996824, 0.310281999917366
1, -0.19656045514854015, 0.19189489255930028, 0.09986300076391501, 0.014415493280989284]
most siginifant columns: ['InternetService', 'Contract', 'StreamingMovies', 'StreamingTV', 'OnlineSecurity',
'PaperlessBilling', 'SeniorCitizen', 'PhoneService']
Logistic Regression performance for predicting "Churn" is 73% (selected features)
Decision Tree performance for predicting "Churn" is 68% (all columns)



['Contract', 'TotalCharges', 'MonthlyCharges', 'tenure'] [0.24526542250272595, 0.18452934524428413, 0.1661083
053202223, 0.13379836828287034]
most siginifant columns: ['Contract', 'TotalCharges', 'MonthlyCharges', 'tenure']
Decision Tree performance for predicting "Churn" is 67% (selected features)
['Contract', 'TotalCharges', 'MonthlyCharges', 'tenure']
[0.020795435649770256, 0.01929476061364992, 0.015260801123134802, 0.008766853907582688]

In [9]:
```
after_list = ['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure', 'PhoneService', 'MultipleLines',
 \
              'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'Stream
ingTV', \
              'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges', 'TotalCha
rges']
temp = ''; temp1 = ''
for i in range(len(after_list)):
    temp += "'" + after_list[i] + "': x_train[:, " + str(i) + "], "
    temp1 += "'" + after_list[i] + "': x_test[:, " + str(i) + "], "
print(temp)
print(temp1)
```

```
'gender': x_train[:, 0], 'SeniorCitizen': x_train[:, 1], 'Partner': x_train[:, 2], 'Dependents': x_train[:,
3], 'tenure': x_train[:, 4], 'PhoneService': x_train[:, 5], 'MultipleLines': x_train[:, 6], 'InternetServic
e': x_train[:, 7], 'OnlineSecurity': x_train[:, 8], 'OnlineBackup': x_train[:, 9], 'DeviceProtection': x_trai
n[:, 10], 'TechSupport': x_train[:, 11], 'StreamingTV': x_train[:, 12], 'StreamingMovies': x_train[:, 13], 'C
ontract': x_train[:, 14], 'PaperlessBilling': x_train[:, 15], 'PaymentMethod': x_train[:, 16], 'MonthlyCharge
s': x_train[:, 17], 'TotalCharges': x_train[:, 18],
'gender': x_test[:, 0], 'SeniorCitizen': x_test[:, 1], 'Partner': x_test[:, 2], 'Dependents': x_test[:, 3],
'tenure': x_test[:, 4], 'PhoneService': x_test[:, 5], 'MultipleLines': x_test[:, 6], 'InternetService': x_tes
t[:, 7], 'OnlineSecurity': x_test[:, 8], 'OnlineBackup': x_test[:, 9], 'DeviceProtection': x_test[:, 10], 'Te
chSupport': x_test[:, 11], 'StreamingTV': x_test[:, 12], 'StreamingMovies': x_test[:, 13], 'Contract': x_test
[:, 14], 'PaperlessBilling': x_test[:, 15], 'PaymentMethod': x_test[:, 16], 'MonthlyCharges': x_test[:, 17],
'TotalCharges': x_test[:, 18],
```

In [ ]: