Technische Hochschule Brandenburg
Predictive Analytics and Big Data SoSe 2021

Analyse der Telecom Datensätze und Churn Vorhersage

Projekt vorgelegt von Gruppe 5:
Jonathan Lovell Darmawan
Lau Kwai Fan
Terrence Loe

Betreuer: Prof. Dr. Ivo Keller

Brandenburg/H., den 30. Juni 2021

Table of Contents

1 Einleitung

Das Ziel des Projekts war es, die Daten von den Telecom-Benutzern zu analysieren und eventuell einen Rückschluss auf die Kundenabwanderung (auf Englisch „Churn“) des Telecomservices ziehen zu können.

2 Überblick des Datensatzes

Die originalen Datensätze wurden von der Kaggle Website heruntergeladen (Quelle: https://www.kaggle.com/radmirzosimov/telecom-users-dataset). Zunächst wurde der Datensatz im ein Jupiter Notebook geladen. Die Attribute „Unnamed: 0“ und „CustomerID“ wurden zuerst entfernt, weil sie scheinbar für die Kundenabwanderung des Telecomservices irrelevant sind. Das Attribut „TotalCharges“ wurde auch entfernt, weil es ein Produkt von den Attributen „tenure“ und „MonthlyCharges“ ist, welche zur Analyse betrachtet werden.

```
filename = 'telecom_users.csv'
df = pd.read_csv(filename)
df = df.drop(['Unnamed: 0', 'customerID', 'TotalCharges'], axis=1)
df.head()
```

Der Datensatz beinhaltet jetzt insgesamt 5986 Datensätze und 19 Dimensionen. Die folgende Tabelle zeigt einen Überblick der Datensätze.

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection
0	Male	0	Yes	Yes	72	Yes	Yes	No	No internet service	No internet service	No internet service
1	Female	0	No	No	44	Yes	No	Fiber optic	No	Yes	Yes
2	Female	1	Yes	No	38	Yes	Yes	Fiber optic	No	No	No
3	Male	0	No	No	4	Yes	No	DSL	No	No	No
4	Male	0	No	No	2	Yes	No	DSL	Yes	No	Yes

TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	Churn
No internet service	No internet service	No internet service	Two year	No	Credit card (automatic)	24.10	No
No	Yes	No	Month-to-month	Yes	Credit card (automatic)	88.15	No
No	No	No	Month-to-month	Yes	Bank transfer (automatic)	74.95	Yes
No	No	Yes	Month-to-month	Yes	Electronic check	55.90	No
No	No	No	Month-to-month	No	Electronic check	53.45	No

3 Datennormalisierung

Alle Werte von kategorischen Attributen wurden auf Zahlen kodiert, dadurch können alle Daten mit den gleichen Datentype berechnet werden.

```
# encode the categorical attributes
from sklearn.preprocessing import LabelEncoder
df['gender'].replace(to_replace='Male', value=1, inplace=True)
df['gender'].replace(to_replace='Female',  value=0, inplace=True)
df['Dependents'].replace(to_replace='Yes', value=1, inplace=True)
df['Dependents'].replace(to_replace='No',  value=0, inplace=True)
df['PaperlessBilling'].replace(to_replace='Yes', value=1, inplace=True)
df['PaperlessBilling'].replace(to_replace='No',  value=0, inplace=True)
df['Partner'].replace(to_replace='Yes', value=1, inplace=True)
df['Partner'].replace(to_replace='No',  value=0, inplace=True)
for col in ["Bank transfer (automatic)","Credit card (automatic)","Electronic check","Mailed check"]:
    df['PaymentMethod'] = LabelEncoder().fit_transform(df['PaymentMethod'])
for col in ["No","Fiber optic","DSL"]:
    df['InternetService'] = LabelEncoder().fit_transform(df['InternetService'])
for col in ["No","Yes","No internet service"]:
    df['OnlineSecurity'] = LabelEncoder().fit_transform(df['OnlineSecurity'])
for col in ["No","Yes","No internet service"]:
    df['OnlineBackup'] = LabelEncoder().fit_transform(df['OnlineBackup'])
for col in ["No","Yes","No internet service"]:
    df['DeviceProtection'] = LabelEncoder().fit_transform(df['DeviceProtection'])
for col in ["No","Yes","No internet service"]:
    df['TechSupport'] = LabelEncoder().fit_transform(df['TechSupport'])
for col in ["No","Yes","No internet service"]:
    df['StreamingTV'] = LabelEncoder().fit_transform(df['StreamingTV'])
for col in ["Two year","Month-to-month","One year","Two year"]:
    df['Contract'] = LabelEncoder().fit_transform(df['Contract'])
for col in ["Two year","Month-to-month","One year","Two year"]:
    df['Contract'] = LabelEncoder().fit_transform(df['Contract'])
for col in ["Yes","No","No internet service"]:
    df['StreamingMovies'] = LabelEncoder().fit_transform(df['StreamingMovies'])
for col in ["Yes","No"]:
    df['PhoneService'] = LabelEncoder().fit_transform(df['PhoneService'])
for col in ["Yes","No","No internet service"]:
    df['MultipleLines'] = LabelEncoder().fit_transform(df['MultipleLines'])
for col in ["Yes","No"]:
    df['Churn'] = LabelEncoder().fit_transform(df['Churn'])
```

Daher wurden alle Zahlen auf Maximal 1 normalisiert:

```
df["tenure"] = df["tenure"].div(10).div(7.2).round(2)
df["MonthlyCharges"] = df["MonthlyCharges"].div(100).div(1.19).round(2)
df["MultipleLines"] = df["MultipleLines"].div(2).round(2)
df["InternetService"] = df["InternetService"].div(2).round(2)
df["OnlineSecurity"] = df["OnlineSecurity"].div(2).round(2)
df["OnlineBackup"] = df["OnlineBackup"].div(2).round(2)
df["DeviceProtection"] = df["DeviceProtection"].div(2).round(2)
df["TechSupport"] = df["TechSupport"].div(2).round(2)
df["StreamingTV"] = df["StreamingTV"].div(2).round(2)
df["StreamingMovies"] = df["StreamingMovies"].div(2).round(2)
df["Contract"] = df["Contract"].div(2).round(2)
df["PaymentMethod"] = df["PaymentMethod"].div(3).round(2)
df.describe()
```

Mit der Python Methode .describe() wurde die normalisierten Daten verifiziert:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup
count	5986.000000	5986.000000	5986.000000	5986.000000	5986.000000	5986.000000	5986.000000	5986.000000	5986.000000	5986.000000
mean	0.509522	0.161377	0.485132	0.299198	0.450809	0.901437	0.474942	0.435099	0.394003	0.456983
std	0.499951	0.367909	0.499821	0.457945	0.340962	0.298100	0.474098	0.368912	0.429974	0.440754
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000	0.120000	1.000000	0.000000	0.000000	0.000000	0.000000
50%	1.000000	0.000000	0.000000	0.000000	0.400000	1.000000	0.500000	0.500000	0.500000	0.500000
75%	1.000000	0.000000	1.000000	1.000000	0.780000	1.000000	1.000000	0.500000	1.000000	1.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	Churn
5986.000000	5986.000000	5986.000000	5986.000000	5986.000000	5986.000000	5986.000000	5986.000000	5986.000000
0.451136	0.397678	0.493067	0.498580	0.347310	0.589375	0.525060	0.544592	0.265119
0.440145	0.430864	0.442795	0.442847	0.416442	0.491988	0.356611	0.253051	0.441433
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.150000	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.330000	0.300000	0.000000
0.500000	0.500000	0.500000	0.500000	0.000000	1.000000	0.670000	0.590000	0.000000
1.000000	1.000000	1.000000	1.000000	0.500000	1.000000	0.670000	0.760000	1.000000
1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

Die behandelten Datensätze werden wie folgt dargestellt:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection
0	1	0	1	1	1.00	1	1.0	1.0	0.5	0.5	0.5
1	0	0	0	0	0.61	1	0.0	0.5	0.0	1.0	1.0
2	0	1	1	0	0.53	1	1.0	0.5	0.0	0.0	0.0
3	1	0	0	0	0.06	1	0.0	0.0	0.0	0.0	0.0
4	1	0	0	0	0.03	1	0.0	0.0	1.0	0.0	1.0

TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	Churn
0.5	0.5	0.5	1.0	0	0.33	0.20	0
0.0	1.0	0.0	0.0	1	0.33	0.74	0
0.0	0.0	0.0	0.0	1	0.00	0.63	1
0.0	0.0	1.0	0.0	1	0.67	0.47	0
0.0	0.0	0.0	0.0	0	0.67	0.45	0

4 Identifizierung guter Attribute

Drei Methoden zur Identifizierung der besten Attribute wurden verwendet:

- Correlation
- Random Forest Estimator
- ID3

4.1 Correlation

Die Korrelationen zwischen den Attributen wurden mittels des Peason Verfahrens berechnet. Dadurch wurde die Beziehung zwischen jeweils zwei Attributen quantifiziert.

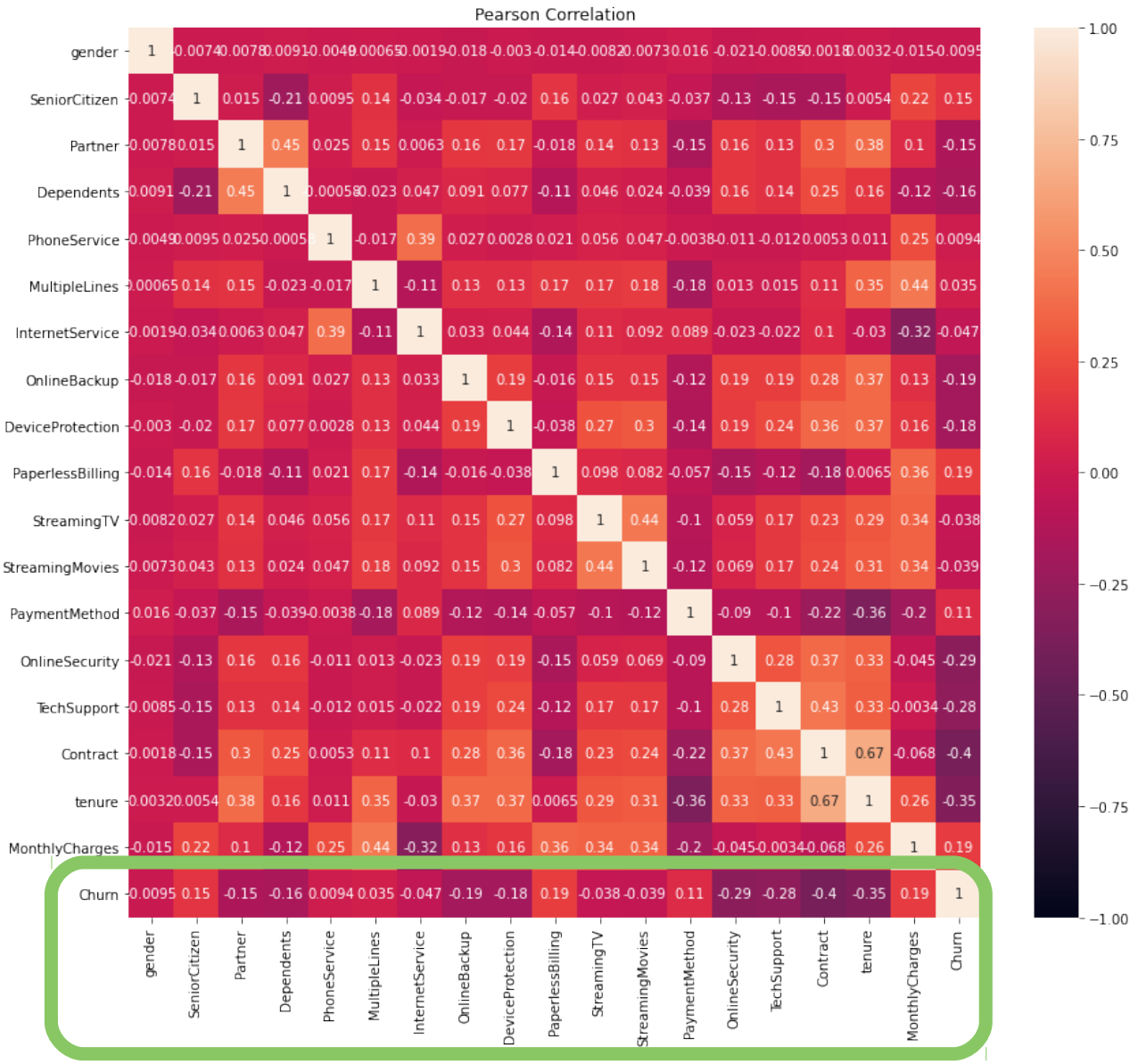
```
# define categorical features
features_cat = ['gender','SeniorCitizen', 'Partner', 'Dependents',
                'PhoneService', 'MultipleLines', 'InternetService',
                'OnlineBackup', 'DeviceProtection','PaperlessBilling',
                'StreamingTV', 'StreamingMovies',
                'PaymentMethod','OnlineSecurity','TechSupport',
                'Contract','tenure','MonthlyCharges', 'Churn']
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# Pearson (linear) correlation
corr_pearson = df[features_cat].corr(method='pearson')

fig = plt.figure(figsize = (14,12))
sns.heatmap(corr_pearson, annot=True, vmin=-1, vmax=+1)
plt.title('Pearson Correlation')
plt.show()
```

Die Korrelationen wurden dann in der folgenden Heatmap zugeordnet. Besonders relevant zum Projektziel ist die Korrelation zwischen dem Attribut „Churn“ und den anderen Attributen. Mit der Heatmap kann man feststellen, dass die Attribute „Contract“ und „tenure“ relativ stark negativ korreliert sind (jeweils mit einem Wert von -0,4 und -0,35). Also wenn sich der Wert von „Churn“ vergrößert, verkleinern sich jeweils die Werte von „Contract“ und von „tenure“. „Contract“ und „tenure“ sind deswegen 2 gute Attribute, um sie später zum Clustering der Datensätze zu benutzen.



4.2 Random Forest

Die Random Forest Methode wurde verwendet, um die wichtigen Attribute für das Attribut „Churn“ zu finden.

```
# standard package
import time

# ML tools
import h2o
from h2o.estimators import H2ORandomForestEstimator
```

```
# define categorical features
cat = ['gender', 'SeniorCitizen', 'Partner', 'Dependents','PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection','TechSupport','StreamingTV', 'StreamingMovies',
       'Contract', 'PaperlessBilling','PaymentMethod']
# define numerical features
num = ['tenure','MonthlyCharges']
# define target
target = 'Churn'
```

```
# select predictors
predictors = num + cat
print('Number of predictors: ', len(predictors))
print(predictors)
```

```
Number of predictors:  18
['tenure', 'MonthlyCharges', 'gender', 'SeniorCitizen', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines',
'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingM
ovies', 'Contract', 'PaperlessBilling', 'PaymentMethod']
```

```
# initialize the h2o module
h2o.init()
```

```
# upload dataframe in h2o environment
df_hex = h2o.H2OFrame(df)

# train and test split into 70:30
train_hex, test_hex = df_hex.split_frame(ratios=[0.7], seed=999)
```

```
Parse progress: |████████████████████████████████████████████████████████| 100%
```

```
# cross validation
n_CV = 5
```

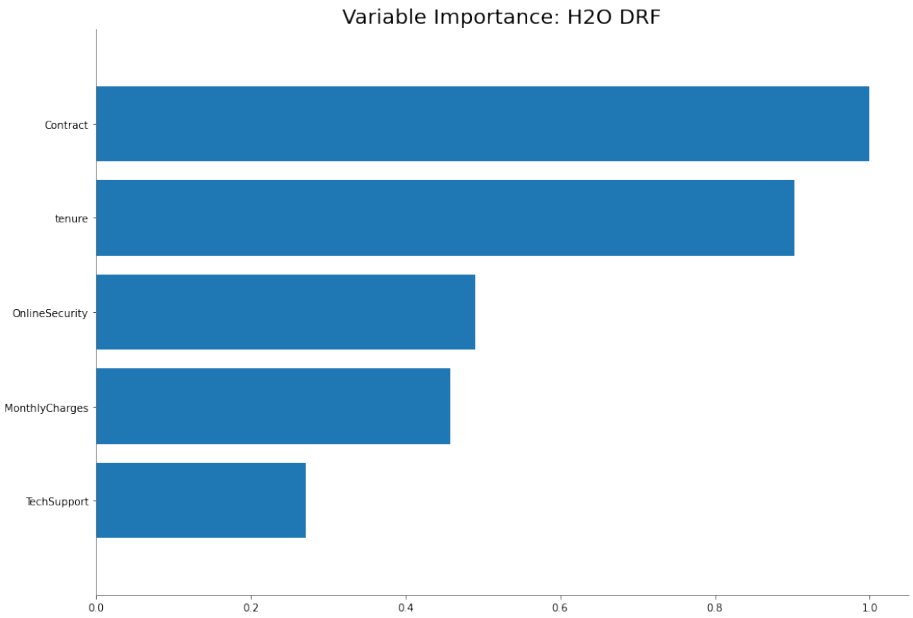
```
# define Random Forest model
fit_2 = H2ORandomForestEstimator(ntrees=50,
                                 max_depth=15,
                                 min_rows=5,
                                 nfolds=n_CV,
                                 model_id='DRF_1',
                                 seed=999)
```

```
# train model
t1 = time.time()
fit_2.train(x=predictors,
            y=target,
            training_frame=train_hex)
t2 = time.time()
print('Elapsed time [s]: ', np.round(t2-t1,2))
```

```
drf Model Build progress: |████████████████████████████████████████████| 100%
Elapsed time [s]:  12.81
```

Die Wichtigkeit der Attribute für das Attribut „Churn“ ist in den folgenden Balkendiagramm dargestellt.

```
# show variable importance
fit_2.varimp_plot(5)
```



Die 5 wichtigsten Attribute für das Attribut „Churn“ sind deswegen „Contract“, „tenure“, „Online Security“, „MonthlyCharges“ und „TechSupport“.

4.3 ID3

ID3 wurde verwendet, um einen Entscheidungsbaum für die Datensätze zu erzeugen. Ein bestes Attribut wird iterativ mittels des ID3 Algorithmus identifiziert, indem die Entropien aller Attribute berechnet und verglichen werden. Das Attribut mit der minimalen Entropie wird dann ausgewählt und davon wird eine Entscheidungsnote gebildet. Die terminalen Blätter beziehen sich auf ein Zielattribut d.h. das Attribut „Churn“.

```
from sklearn.model_selection import train_test_split, cross_validate

# split the dataset into test data and train data
df_train, df_test = train_test_split(df)
```

Standardmäßig wird der Entscheidungsbaum so tief wie möglich sein, wenn keine Einschränkung für die Tiefe gesetzt wird. Dadurch wird ID3 die Tendenz haben, in einem Overfitting-Problem zu resultieren. Um Overfitting zu vermeiden, wurde die Baumtiefe beschränkt. Um die optimale Baumtiefe zu finden, wurde eine Cross-Validation durchgeführt.

```
# use cross-validation to find the optimal tree depth

# get the accuracies of different candidate depths from 2 to 20
m_candidate = [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
res = dict()

for m in m_candidate:
    pred3 = DecisionTreeClassifier(max_depth=m)
    res[m] = cross_validate(pred3,
                            X=df_train.drop(["Churn"], axis=1),
                            y=df_train.Churn,
                            cv=10,
                            return_train_score=False,
                            scoring='accuracy')

resdf = DataFrame({(i, j): res[i][j]
                            for i in res.keys()
                            for j in res[i].keys()}).T

resdf.loc[(slice(None), 'test_score'), :]
```

```
# get the mean accuracy of candidate depths from 2 to 20
# depth = 5 yields the highest accuracy => maximum depth
resdf.loc[(slice(None), 'test_score'), :].mean(axis=1)
```

```
2   test_score    0.736912
3   test_score    0.780126
4   test_score    0.780349
5   test_score    0.784359
6   test_score    0.783023
7   test_score    0.778118
8   test_score    0.773221
9   test_score    0.760074
10  test_score    0.750721
11  test_score    0.744043
12  test_score    0.736021
13  test_score    0.726655
14  test_score    0.724879
15  test_score    0.720867
16  test_score    0.724875
17  test_score    0.723093
18  test_score    0.724207
19  test_score    0.718639
20  test_score    0.722646
dtype: float64
```

Von den obigen Berechnungen kann man feststellen, dass eine Baumtiefe mit 5 Ebenen zur höchsten ID3-Genauigkeit führt. Ein Entscheidungsbaum mit der maximalen Baumtiefe = 5 wurde anschließend generiert.

```
# use ID3 to generate a tree again by limiting maximum tree depth to be 5
tree4 = DecisionTreeClassifier(criterion = "entropy",max_depth=5)
tree4 = tree4.fit(X=df_train.drop(["Churn"], axis=1),
                  y=df_train.Churn)

dot_data = StringIO()

export_graphviz(tree4, out_file=dot_data, filled=True, rounded=True, special_characters=True,
                feature_names=["gender", "SeniorCitizen", "Partner", "Dependents", "tenure", "PhoneService",
                               "MultipleLines", "InternetService", "OnlineSecurity", "OnlineBackup",
                               "DeviceProtection", "TechSupport", "StreamingTV", "StreamingMovies", "Contract",
                               "PaperlessBilling", "PaymentMethod", "MonthlyCharges"],
                class_names=["No","Yes"],
                proportion=True)

graph2 = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph2.create_png())
```

```
# write the tree as png format
graph2.write_png("telecom14.png")
```

Der vollständige Entscheidungsbaum ist in Abbildung 1 gezeigt.

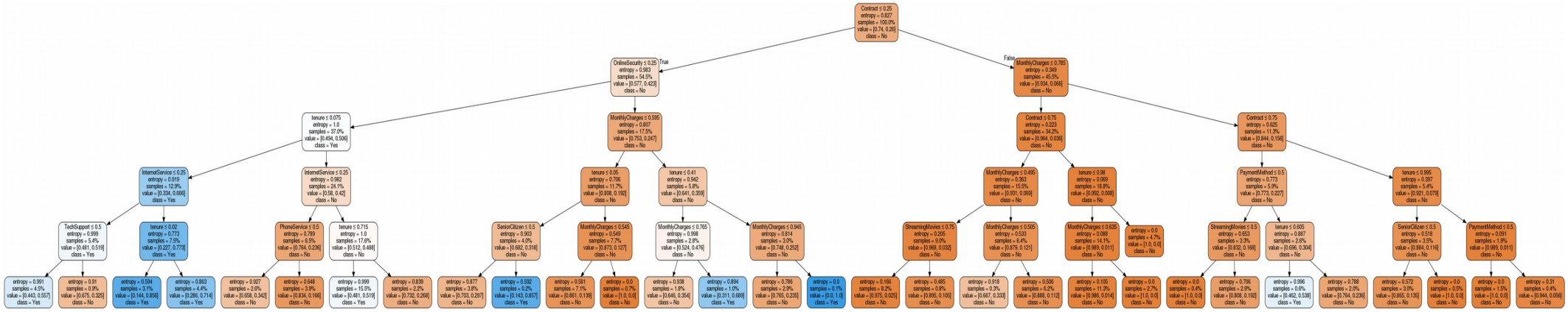
Danach wurden die Attribute, die oben im Baum stehen, ausgewählt. Sie sind „Contract“, „OnlineSecurity“, „MonthlyCharges“ und „tenure“.

Die folgende Tabelle zeigt die Ergebnisse von der Identifizierung guter Attribute mit der jeweiligen Methode:

Verfahren	Gute Attribute zum Clustering der Datensätzen
Correlation	„Contract“ und „tenure“
Random Forest Estimator	„Contract“, „tenure“, „Online Security“, „MonthlyCharges“ und „TechSupport“
ID3 Entscheidungsbaum	„Contract“, „OnlineSecurity“, „MonthlyCharges“ und „tenure“

Wir haben uns entschieden, dass „Contract“, „OnlineSecurity“, „MonthlyCharges“ und „tenure“ die 4 guten Attribute zum Clustering der Datensätze sind.

Abbildung 1: Entscheidungsbaum für Telecom-Benutzer Datensätzen



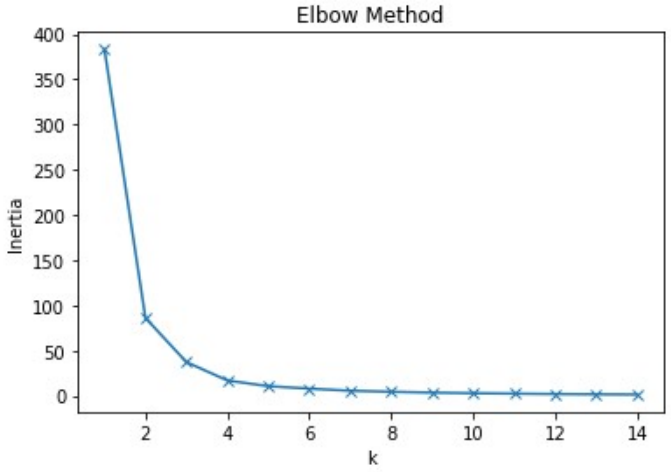
5 Bestes Attribut und Clustering mit K-Means

Um die Datensätze sinnvoll zu klassifizieren, muss das beste Attribut oder müssen die besten Attribut-Kombinationen identifiziert werden. Dafür wurde die Elbow Methode verwendet. Die ausgewählten guten Attribute von #4 wurden durch den K-Means Algorithmus verarbeitet. Dadurch wurden die Inertia für die Anzahl der Cluster von 1 bis 14 berechnet. Das Attribut oder die Attribut-Kombinationen, die die schönste Elbow Kurve ergeben, führt zur besten Clusterstruktur der Datensätze.

Wir haben eventuell herausgefunden, dass das Attribut „MonthlyCharges“ allein die schönste Elbow Kurve vom K-Means ergibt:

```
#MonthlyCharges
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
inertia = []
K = range(1,15)
for k in K:
    km = KMeans(n_clusters=k, init ='k-means++')
    km = km.fit(X3)
    inertia.append(km.inertia_)
plt.plot(K, inertia, marker= "x")
plt.xlabel('k')
#plt.xticks(np.arange(15))
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.show()
```

Die schönste Elbow Kurve vom Attribut „MonthlyCharges“ wurde wie folgt in einem Liniendiagramm dargestellt:



Von der obigen Elbow Kurve wurde festgelegt, dass die optimale Anzahl der Cluster 3 ist. Danach wurden die einzelnen Daten einem Cluster zugeordnet, dessen Centroid den kürzesten Abstand zu den jeweiligen Daten hat. Die Centroids der 3 Cluster sind mit kmeans.cluster_centers_ zu bestimmen.

```
kmeans.cluster_centers_
```

```
array([[0.79094873],
       [0.19679328],
       [0.51744614]])
```

Die Clusterzuordnung der Daten ist wie folgt:

```
df_monthlyCharges_clusters = df3.assign(cluster = lambda x: y_pred)
df_monthlyCharges_clusters
```

	MonthlyCharges	cluster
0	0.20	1
1	0.74	0
2	0.63	2
3	0.47	2
4	0.45	2
...	...	...
5981	0.80	0
5982	0.77	0
5983	0.18	1
5984	0.84	0
5985	0.17	1

5986 rows × 2 columns

Semantische Begriffe für die 3 Cluster wurden ausgedacht, indem wir die Eigenschaften der Cluster mit der Methode .describe() betrachtet haben:

```
df_monthlyCharges_cluster0 = df_monthlyCharges_clusters[(df_monthlyCharges_clusters["cluster"] == 0)]
df_monthlyCharges_cluster0
```

```
df_monthlyCharges_cluster1 = df_monthlyCharges_clusters[(df_monthlyCharges_clusters["cluster"] == 1)]
df_monthlyCharges_cluster1
```

```
df_monthlyCharges_cluster2 = df_monthlyCharges_clusters[(df_monthlyCharges_clusters["cluster"] == 2)]
df_monthlyCharges_cluster2
```

```
df_monthlyCharges_cluster0.describe()
```

```
df_monthlyCharges_cluster1.describe()
```

```
df_monthlyCharges_cluster2.describe()
```

	MonthlyCharges	cluster
count	2477.000000	2477.0
mean	0.790949	0.0
std	0.086992	0.0
min	0.660000	0.0
25%	0.720000	0.0
50%	0.790000	0.0
75%	0.860000	0.0
max	1.000000	0.0

	MonthlyCharges	cluster
count	1606.000000	1606.0
mean	0.196793	1.0
std	0.047634	0.0
min	0.150000	1.0
25%	0.170000	1.0
50%	0.170000	1.0
75%	0.210000	1.0
max	0.350000	1.0

	MonthlyCharges	cluster
count	1903.000000	1903.0
mean	0.517446	2.0
std	0.088403	0.0
min	0.360000	2.0
25%	0.430000	2.0
50%	0.520000	2.0
75%	0.590000	2.0
max	0.650000	2.0

Cluster	Semantischer Begriff
0	„high_charges“
1	„low_charges“
2	„mid_charges“

Die semantischen Begriffe wurden den einzelnen Daten zugeordnet.

```
df_monthlyCharges_clusters = df_monthlyCharges_clusters.replace({'cluster': {0: 'high_charges'}})
df_monthlyCharges_clusters = df_monthlyCharges_clusters.replace({'cluster': {1: 'low_charges'}})
df_monthlyCharges_clusters = df_monthlyCharges_clusters.replace({'cluster': {2: 'mid_charges'}})
df_monthlyCharges_clusters
```

	MonthlyCharges	cluster
0	0.20	low_charges
1	0.74	high_charges
2	0.63	mid_charges
3	0.47	mid_charges
4	0.45	mid_charges
...	...	...
5981	0.80	high_charges
5982	0.77	high_charges
5983	0.18	low_charges
5984	0.84	high_charges
5985	0.17	low_charges

5986 rows × 2 columns

Die normalisierten Werte von „MonthlyCharges“ wurden zurück zu den originalen Werten konvertiert:

```
df4 = pd.read_csv(filename)
df_bayes = pd.concat([df4[['Churn','MonthlyCharges']], df_monthlyCharges_clusters['cluster']], axis=1)
df_bayes
```

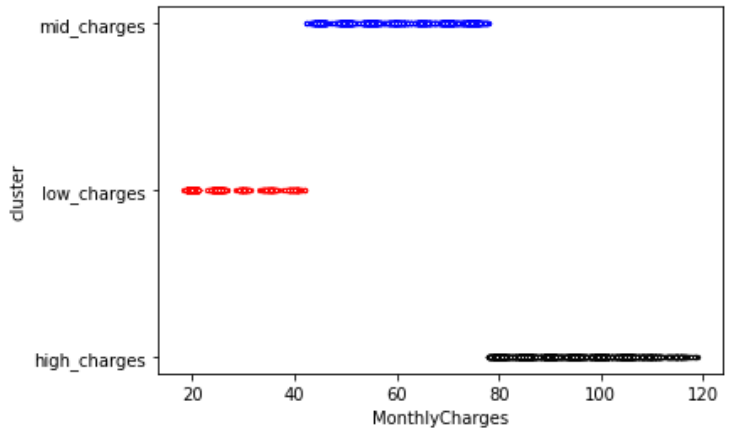
	Churn	MonthlyCharges	cluster
0	No	24.10	low_charges
1	No	88.15	high_charges
2	Yes	74.95	mid_charges
3	No	55.90	mid_charges
4	No	53.45	mid_charges
...	...	...	...
5981	Yes	95.00	high_charges
5982	No	91.10	high_charges
5983	No	21.15	low_charges
5984	Yes	99.45	high_charges
5985	No	19.80	low_charges

5986 rows × 3 columns

Das Clustering der Datensätze ist wie folgt visualisiert:

```
ax = df5[df5['cluster']=='high_charges'].plot.scatter(x='MonthlyCharges', y='cluster', s=5, color='white', edgecolor='black')
df5[df5['cluster']=='low_charges'].plot.scatter(x='MonthlyCharges', y='cluster', s=5, color='white', ax=ax, edgecolor='red')
df5[df5['cluster']=='mid_charges'].plot.scatter(x='MonthlyCharges', y='cluster', s=5, color='white', ax=ax, edgecolor='blue')
```

```
<AxesSubplot:xlabel='MonthlyCharges', ylabel='cluster'>
```



6 Bayes Klassifikator

Weil festgestellt wurde, dass das Attribut „MonthlyCharges“ das beste Attribut ist, wurde damit ein Bayes Klassifikator entwickelt. Die Attribute „MonthlyCharges“ und „Churn“ wurden dafür benutzt. Mittels eines solchen Bayes Klassifkators kann man von den Rückschlußwahrscheinlichkeiten bestimmen, in welchen Klassen („churn“ oder „nicht_churn“) ein Kunde am wahrscheinlichsten eingeordnet werden würde.

Zunächst wurde ein neuer Dataframe erstellt:

```
df_bayes = df5.copy()
df_bayes
```

	Churn	MonthlyCharges	cluster
0	No	24.10	low_charges
1	No	88.15	high_charges
2	Yes	74.95	mid_charges
3	No	55.90	mid_charges
4	No	53.45	mid_charges
...	...	...	...
5981	Yes	95.00	high_charges
5982	No	91.10	high_charges
5983	No	21.15	low_charges
5984	Yes	99.45	high_charges
5985	No	19.80	low_charges

5986 rows × 3 columns

Danach wurde „MonthlyCharges“ in 11 Intervalle aufgeteilt. Die Häufigkeiten der Werte „No“ und „Yes“ vom Attribut „Churn“ für jedes Intervall wurden gezahlt:

```
bins = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120]
groups = df_bayes.groupby(['Churn', pd.cut(df_bayes.MonthlyCharges, bins)])
groups2 = groups.size().unstack()
groups2
```

MonthlyCharges	(10, 20]	(20, 30]	(30, 40]	(40, 50]	(50, 60]	(60, 70]	(70, 80]	(80, 90]	(90, 100]	(100, 110]	(110, 120]
Churn											
No	507	752	115	268	418	349	475	515	452	391	157
Yes	50	93	46	128	108	97	302	280	260	196	27

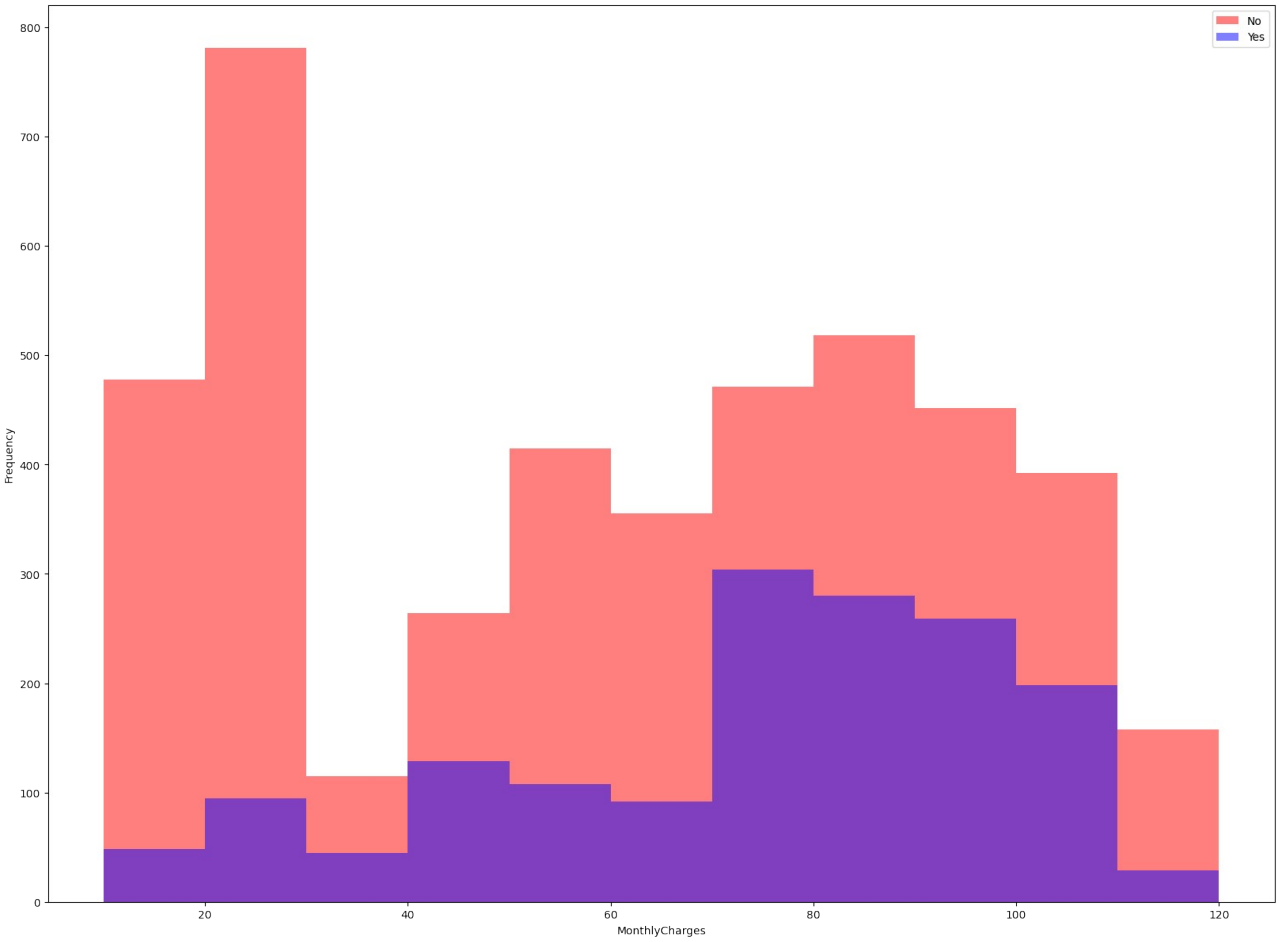
Danach wurden die Histogramme für die 2 Klassen („Yes“ und „No“) dargestellt:

```
x1=df_bayes.loc[df_bayes.Churn=="No", "MonthlyCharges"]
x2=df_bayes.loc[df_bayes.Churn=="Yes", "MonthlyCharges"]

kwargs = dict(alpha=0.5, bins=bins)

plt.rcParams.update({'figure.figsize' : (20,15), 'figure.dpi': 100})
plt.hist(x1, **kwargs, color='r', label = "No" )
plt.hist(x2, **kwargs, color='b', label = "Yes")
df_bayes["Churn"]
```

```
0        No
1        No
2       Yes
3        No
4        No
       ...
5981    Yes
5982     No
5983     No
5984    Yes
5985     No
Name: Churn, Length: 5986, dtype: object
```



Die Häufigkeit für das jeweilige Intervall ist:

```
total = groups2.sum()
total.name = 'Total'
groups2 = groups2.append(total.transpose())
groups2
```

MonthlyCharges	(10, 20]	(20, 30]	(30, 40]	(40, 50]	(50, 60]	(60, 70]	(70, 80]	(80, 90]	(90, 100]	(100, 110]	(110, 120]
Churn											
No	507	752	115	268	418	349	475	515	452	391	157
Yes	50	93	46	128	108	97	302	280	260	196	27
Total	557	845	161	396	526	446	777	795	712	587	184

Und die gesamte Apriori für die Klasse „Yes“ und „No“ ist:

```
df_apriori = groups2.sum(axis=1)
df_apriori
```

```
Churn
No       4399
Yes      1587
Total    5986
dtype: int64
```

Dann wurden die Apriori im Prozentsatz für die Klassen „Yes“ und „No“ berechnet:

```
df_apriori_procent_no = df_apriori.No/df_apriori.Total
df_apriori_procent_no
```

```
0.7348813899097895
```

```
df_apriori_procent_yes = df_apriori.Yes/df_apriori.Total
df_apriori_procent_yes
```

```
0.2651186100902105
```

Danach wurden die Likelihood*apriori für die Klassen „Yes“ und „No“ jeweils berechnet:

```
no_likelihood = df.values[0]
no_likelihood
```

```
array([507, 752, 115, 268, 418, 349, 475, 515, 452, 391, 157])
```

```
yes_likelihood = df.values[1]
yes_likelihood
```

```
array([ 50,  93,  46, 128, 108,  97, 302, 280, 260, 196,  27])
```

```
no_likelihood_mal_priori = []
```

```
for i in no_likelihood:
    p = i * df_apriori_procent_no
    no_likelihood_mal_priori.append(p)

no_likelihood_mal_priori
```

```
[372.58486468426327,
 552.6308052121617,
 84.51135983962578,
 196.94821249582358,
 307.180420982292,
 256.4736050785165,
 349.06866020715,
 378.46391580354157,
 332.16638823922483,
 287.33862345472767,
 115.37637821583695]
```

```
yes_likelihood_mal_priori = []
for i in yes_likelihood:
    p = i * df_apriori_procent_yes
    yes_likelihood_mal_priori.append(p)

yes_likelihood_mal_priori
```

```
[13.255930504510523,
 24.656030738389575,
 12.195456064149683,
 33.93518209154694,
 28.63280988974273,
 25.716505178750417,
 80.06582024724356,
 74.23321082525894,
 68.93083862345472,
 51.96324757768125,
 7.158202472435683]
```

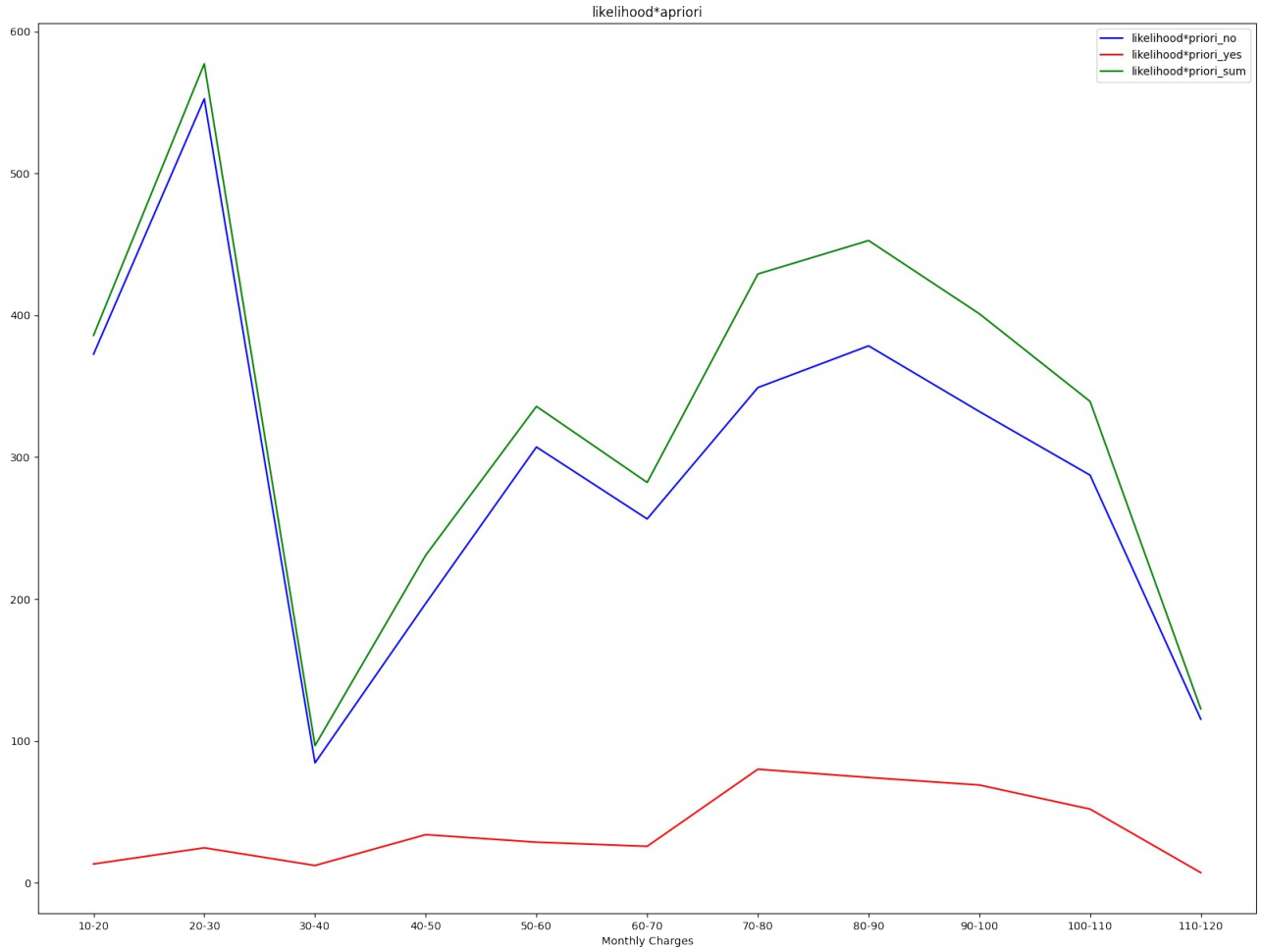
Danach wurden die Nenner für das jeweilige Intervall berechnet:

```
total = []
for i in range(0,11):
    t = yes_likelihood_mal_priori[i] + no_likelihood_mal_priori[i]
    total.append(t)
total
```

```
[385.84079518877377,
 577.2868359505512,
 96.70681590377546,
 230.88339458737053,
 335.81323087203475,
 282.19011025726695,
 429.13448045439355,
 452.6971266288005,
 401.0972268626796,
 339.30187103240894,
 122.53458068827263]
```

Die Likelihood*apriori für „Yes“ und „No“ und die Nenner der Bayes Formel können auch mit einem Liniendiagramm dargestellt werden:

```
x = (["10-20", "20-30","30-40", "40-50", "50-60", "60-70", "70-80", "80-90", "90-100", "100-110", "110-120"])
plt.title("likelihood*apriori")
plt.xlabel("Monthly Charges")
plt.plot(x, no_likelihood_mal_priori, color ="blue", label = "likelihood*priori_no")
plt.plot(x, yes_likelihood_mal_priori, color ="red", label = "likelihood*priori_yes")
plt.plot(x, total, color ="green" , label = "likelihood*priori_sum")
plt.legend()
plt.show()
```



Danach wurden die Aposteriori für die Klassen „Yes“ und „No“ berechnet:

```
aposteriori_yes = []
for i in range(0,11):
    a = yes_likelihood_mal_priori[i] / total[i]
    aposteriori_yes.append(a)

aposteriori_yes
```

```
[0.03435595890793512,
 0.04271019050311679,
 0.12610751321069572,
 0.14697974339902234,
 0.0852640910406939,
 0.09113184425671475,
 0.18657512713139487,
 0.16397985862660042,
 0.17185568487376757,
 0.15314754209745546,
 0.05841781505456092]
```

```
aposteriori_no = []
for i in range(0,11):
    a = no_likelihood_mal_priori[i] / total[i]
    aposteriori_no.append(a)

aposteriori_no
```

```
[0.9656440410920649,
 0.9572898094968834,
 0.8738924867893043,
 0.8530202566009777,
 0.914735908959306,
 0.9088681557432852,
 0.8134248728686052,
 0.8360201413733995,
 0.8281443151262323,
 0.8468524579025445,
 0.9415821849454391]
```

Der Bayes Klassifikator für die Klassen „Yes“ und „No“ kann auch wie folgt grafisch dargestellt werden:

```
plt.title("aposteriori")
plt.xlabel("Monthly Charges")
plt.plot(x, aposteriori_yes, label = "Churn")
plt.plot(x, aposteriori_no, label = "Not_Churn")
plt.legend()
plt.show()
```