Machine Learning: Customer Churn Prediction Model for a Home Internet Telecommunications Company

Written Assignment

Analytical Software and Frameworks

Name: Leander Tripp

Degree Program: M. Sc. Finance, Accounting, Taxation

Student ID.: 32009913

Address: Düsseldorfer Str. 45, 45481 Mülheim an der Ruhr, Germany

E-Mail: leander.tripp@iu-study.org

Semester: 3rd Semester

Course Director: Prof. Dr. Andrew Adjah Sai

Submission Date: 31st May 2023

Table of Contents

List of Figures

List of Abbreviations

API	Application programming interface
AUC	Area under the curve
AWS	Amazon Web Services
COVID-19	Coronavirus disease 2019
CPU	Central processing unit
CSS	Cascading Style Sheets
CSV	Comma separated values
CUDA	Compute unified device architecture
GPU	Graphics processing unit
GUI	Graphical user interface
HTML	Hypertext markup language
JSON	JavaScript object notation
NoSQL	Not only structured query language
OLAP	Online analytical processing
OLTP	Online transaction processing
RAM	Random access memory
ReLU	Rectified linear unit
ROC	Receiver operating characteristic
SQL	Structured query language
XML	Extensible markup language

1. Introduction

1.1 Problem Definition

Home internet telecommunication companies have long been working to identify customers who are at risk of cancelling their contract. By identifying these customers, they can proactively engage with them by offering promotions and incentives, encouraging them to continue using their services. The telecommunications industry is a highly competitive environment, with customers frequently encountering offers from multiple home internet providers.

In order to determine which customers are most likely to cancel their contract, the company ought to initially gather data on all customers for at least one year. It is crucial to retain customer records in the database even after contract cancellation in addition to incorporating a feature into the customer recordset that indicates churn. This practice will enable the telecommunications company to develop a predictive machine learning model based on historical data.

Selecting the right machine learning framework for the production environment is highly important, as it significantly impacts the future development and maintenance costs over the lifetime of the model. The customer churn prediction model should incorporate self-updating capabilities, allowing it to adapt to evolving circumstances, such as the COVID-19 pandemic. Having a reliable home internet connection became essential in a work-from-home environment, leading to increased customer churn and necessitating dynamic prediction models (Shukla et al., 2022, p. 470-471).

Additionally, machine learning models perform best on servers equipped with multiple graphics processing units. Although establishing the necessary processing capabilities on-site is feasible, employing cloud-based services such as AWS server farms is advisable, since it enables the telecommunications company to easily scale the processing resources as needed.

1.2 Objective

The aim of this work is to develop a fully featured customer churn machine learning model suitable for a production environment of a home internet telecommunications company. This model must adhere to the previously stated requirements of being fully adjustable by continually analysing patterns within the underlying customer data. Additionally, the model should be accessible through a front-end graphical user interface (GUI) using a separate application or web application. This application takes the features used for developing the machine learning model as inputs, subsequently outputting the probability of a customer churning. Building the model and its ecosystem with the Python programming language enhances future support, compatibility, and upgradability.

Finally, deployment of the model should improve the company’s data infrastructure, enabling sales employees to target customers more effectively, thereby improving overall productivity.

1.3 Work Structure

The second part of this work introduces basic concepts for databases and machine learning. Databases and data storage are important for feeding the machine learning model with relevant data. The Python machine learning frameworks introduced in the second part of this work represent different options for building the customer churn prediction model.

Construction of the model starts with the generation of an artificial customer dataset. Subsequently, the artificial neural network is created using Google's TensorFlow and Keras modules. The model's performance is evaluated using the Receiver Operator Characteristic and Area under the Curve (AUC) metrics, ensuring accuracy and effectiveness.

Finally, the model is tested by designing a rudimentary tkinter user interface as well as developing a Flask web application powered by Python, HTML and CSS. The second part of this work features code snippets in the form of screenshots. [1]

2. Machine Learning: Customer Churn Prediction Model

2.1 Data and Data Storage Systems

In statistics, data is considered a measurement or observation which provides insights about the object under investigation (Soriano-Valdez et al., 2021, p. 12-13). Differentiating between data and information is essential. Furthermore, an individual has to comprehend the data in order to extract the most important pertinent information (Giese et al., 2020, p. 1237). This allows the user to engage effectively with the data, enabling well-founded and well-informed decisions.

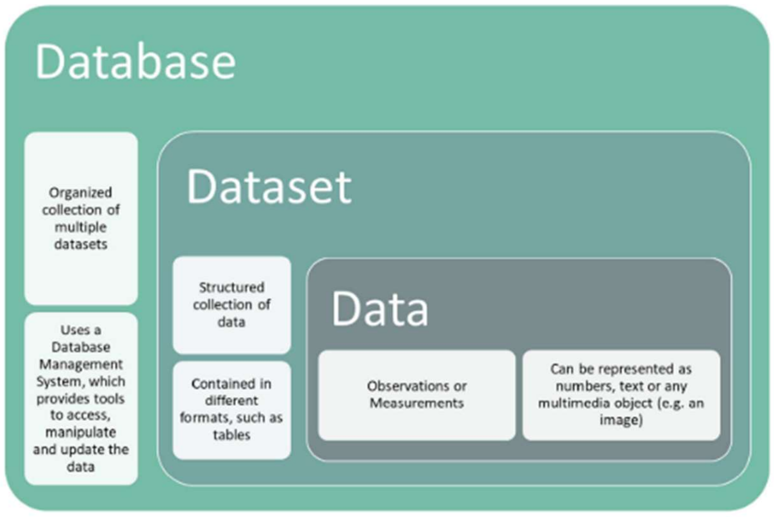

Database
Organized collection of multiple datasets
Uses a Database Management System, which provides tools to access, manipulate and update the data
Dataset
Structured collection of data
Contained in different formats, such as tables
Data
Observations or Measurements
Can be represented as numbers, text or any multimedia object (e.g. an image)

Figure 1: Data, Dataset and Database (Soriano-Valdez et al., 2021, p. 13)

Data can be stored in different formats, such as text, videos, sounds or images. From such data, various data points can be derived. For instance, when examining data related to different

[1] The complete 500+ line codebase for this project is accessible at the following link: https://github.com/leander-ms/ChurnModel

computers, potential data points might include distinct CPUs, capacities of random-access memory (RAM) or graphics processing units (Soriano-Valdez et al., 2021, p. 13). The data corresponding to the various computers can be organized into a tabular format, forming a dataset. Multiple different types of datasets can then form a database (figure 1).

While basic data collection methods such as XML or CSV files and even Excel spreadsheets can qualify as databases, the term “database” typically denotes online transaction processing (OLTP) application databases (Cardon, 2018, p. 2). OLTP databases are typically a couple hundred megabytes to a few gigabytes in size and used during the basic day-to-day operation of an organization (Chaudhuri & Dayal, 1997, p. 1). One transaction usually affects a few records, which are accessed via the primary keys of the table. These databases are also referred to as “operational databases”.

On the other hand, data warehouses employ online analytical processing (OLAP) databases. Although both OLAP and OLTP databases utilize the same programming language (SQL) for data querying, OLAP databases typically operate atop multiple OLTP databases (Cardon, 2018, p. 2-3). Data warehouses, which span hundreds of gigabytes, are capable of managing intricate multidimensional queries that access and analyse millions of records simultaneously (Chaudhuri & Dayal, 1997, p. 1-2).

Another important distinction to consider when examining databases is the disparity between relational and non-relational database systems. Relational databases are controlled and queried using SQL, allowing users to create tables with columns and indexes, views, stored procedures and functions (Capris et al., 2022, p. 214-215). Although relational databases present a reliable and secure choice for data storage and processing, they also possess limitations. The exponential growth in data volume and complexity, driven by the advent of the internet and Web 2.0, require data storage solutions that are both highly adaptable and capable of horizontal scalability (Moniruzzaman & Hossain, 2013, p. 2). These capabilities are provided by non-relational databases, now often referred to as NoSQL databases (Capris et al., 2022, p. 215). Contrary to SQL, NoSQL is not a programming language; instead, it refers to a collection of non-relational data management systems, which do not store data in tables or use SQL queries (Moniruzzaman & Hossain, 2013, p. 1). The most popular NoSQL database, MongoDB (DB-Engines Ranking, n.d.), employs the JSON format, which enables application developers to quickly add new data fields within a document model (Chauhan, 2019, p. 90). A MongoDB database encompasses numerous collections. Each collection contains multiple documents. Documents within a single collection are not required to share identical key-value pairs. Moreover, common fields can accommodate varying data types, further enhancing the flexibility and adaptability of the database structure.

2.2 Machine Learning

Machine Learning addresses the challenge of predicting the future based on past data, as posed by the question “how can we know the future if all we have is data about the past?” (Unpingco, 2016, p. 197). It achieves this through the development of algorithmic predictions based on the input data provided. The final result of a machine learning model is a network defined by an interconnected structure of multiple parameterized functions, similar to how the human brain operates by connecting neurons through synapses (Novac et al., 2022, p. 4). This approach contrasts with classical statistical theory, which focuses on analysing and explaining data, as well as describing different phenomena (Unpingco, 2016, p. 202).

Currently, machine learning finds application in a wide range of areas, such as recommendation engines for social networks, fraud detection, web search, and credit scoring (Domingos, 2012, p. 78). Machine learning protocols can be categorized into supervised, unsupervised, and reinforcement learning models (Madhumala & Tiwari, 2020, p. 40). Supervised learning starts out by labelling the data before conducting the training on the marked data sets. This is useful when the model’s objective is to predict the probability of a certain outcome (Jiang et al., 2020, p. 3-4). In a multilayer neural network with n input nodes and m output nodes, supervised machine learning seeks to establish a mapping function between the input and output patterns by computing the weights of the network in a way that effectively maps the input to the output (Chen, 1996, p. 1220).

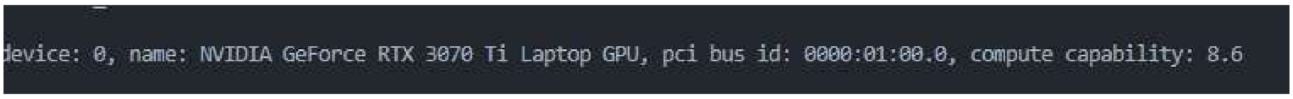
Unsupervised machine learning employs unlabelled training data to identify the underlying categories within the data, subsequently enabling the categorization of data points outside the training set (Biamonte et al., 2017, p. 5). For this reason, unsupervised machine learning is often referred to as “pattern recognition” and used in fields such as cancer research and city-planning (Kassambara, 2017, p. 3).

Reinforcement learning features agents studying their environment in order to take actions that yield the maximum possible reward (Soriano-Valdez et al., 2021, p. 19-20). Models employing reinforcement learning methods utilize unlabelled inputs. The model uses the parameters state (*s*), action (*a*), reward (*r*), policy (*p*) and value (*v*). To improve the model, the agent receives its current state s_t at each time step *t*. It then selects its next action *a* from an action space following its policy *p*. Each action *a* to reach the new state s_{t+1} can be mapped to a scalar reward *r* (Li, 2018, p. 7). This process continues until the agent reaches the desired state, from where it restarts. The agent aims to maximize the reward depending on the chosen policy.

2.3 Machine Learning Frameworks

A machine learning framework or library is a specialized tool built for developers to streamline the development of machine learning applications. Each tool has different capabilities and is tailored to the properties of different categories of machine learning models and serving specific objectives. This paper will cover some of the major machine learning frameworks available for the Python programming language. Python is a high-level, multi-paradigm, interpreted programming language commonly used for building server-side applications. It powers enterprise-level web applications like YouTube and Dropbox with the Django and Flask frameworks, big data analysis with NumPy and Pandas, as well as highly adaptable machine learning and artificial intelligence environments.

One of the most popular machine learning frameworks is Google's TensorFlow. Google's work on large-scale neural networks started in 2011, with the first public version of TensorFlow releasing in 2015 (Abadi et al., 2016, p. 1-2). TensorFlow was built on top of DistBelief, an internal Google software tool, which is used for reinforcement learning and unsupervised learning in products such as Google Search, YouTube, Google Translate and others. While TensorFlow is capable of running on consumer-level CPUs locally, it also has the ability to execute models across a broad spectrum, from a single GPU to thousands of GPUs simultaneously, making it highly flexible for both enterprise and research environments.



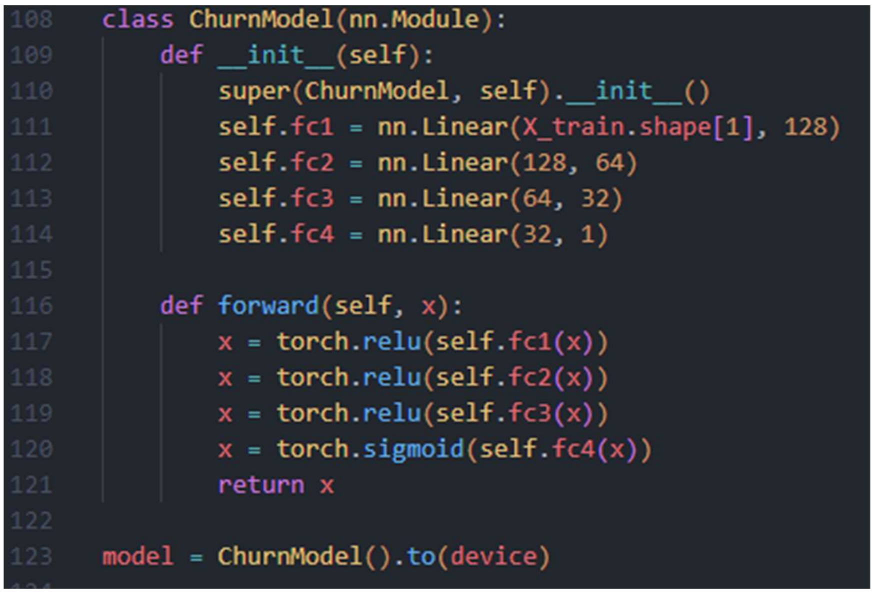
```
levice: 0, name: NVIDIA GeForce RTX 3070 Ti Laptop GPU, pci bus id: 0000:01:00.0, compute capability: 8.6
```

Figure 2: TensorFlow choosing processing unit for the machine learning model

So far, NVIDIA is the only graphics card manufacturer to invest heavily into deep learning, meaning modern frameworks such as TensorFlow and PyTorch will only run on NVIDIA graphics cards (Chollet, 2021, p. 66-67). New NVIDIA graphics cards feature thousands of cores explicitly designed to handle tensor- and CUDA-workloads for machine learning. This data-parallel architecture of modern GPUs offers a performance improvement of five to ten times compared to traditional CPU-based processing.

TensorFlow recognizes each GPU as a device, and every device object involved in processing the model is responsible for allocating and deallocating memory within the system, along with arranging the kernels requested by the higher levels of the TensorFlow operation (Abadi et al., 2016, p. 4). The term "Tensor" in TensorFlow refers to the data structure of the framework. A tensor is a typed, multi-dimensional array, which can be a string, signed or unsigned integer or other data type. These tensors can have a varying number of dimensions and are used for computation, manipulation or passing data through the various operations. This allows the creation and implementation of machine learning models.

TensorFlow's main competitor is the PyTorch framework, which was launched by Facebook in 2016 and strictly adheres to the core principles of the Python programming language (Novac et al., 2022, p. 7). It focuses on keeping the design as simple as possible, with its creators stating that a more complex design leads to a more complex integration and a loss of performance in the long run (figure 3). Like TensorFlow, PyTorch enables the execution of operators on GPUs by utilizing the CUDA stream mechanism, which allows for the concurrent execution of Python code on the CPU and tensor operations on the GPU (Paszke et al., 2019, p. 6). Although PyTorch achieves a more compact model size due to its simple design, TensorFlow regularly outperforms it in terms of training time (Munjal et al., 2022, p. 12-16).



```python
class ChurnModel(nn.Module):
    def __init__(self):
        super(ChurnModel, self).__init__()
        self.fc1 = nn.Linear(X_train.shape[1], 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, 32)
        self.fc4 = nn.Linear(32, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = torch.sigmoid(self.fc4(x))
        return x

model = ChurnModel().to(device)
```

Figure 3: Defining feedforward neural network architecture with three rectified linear unit activation functions and a log-sigmoid output function using PyTorch. Model is transferred to GPU using 'to(device)' function

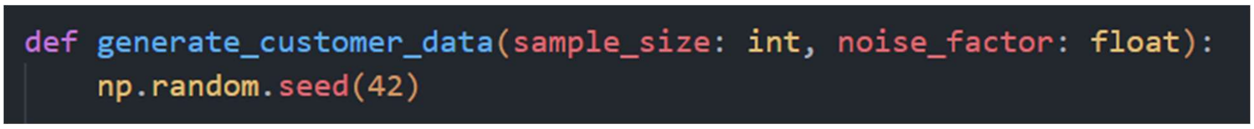
Another easily accessible machine learning framework for Python programming is Scikit-Learn. It not only provides classification-, regression- and clustering algorithms, but also an extensive set of support functionality which can be used in combination with other machine learning frameworks, such as the "train_test_split" or "GridSearchCV" methods for splitting the dataset and finding optimum hyperparameters, as well as evaluation metrics like "roc_curve" and "auc" (Hackeling, 2014, pp. 16, 44, 76, 83).

Keras is a machine learning API running on top of the TensorFlow framework, enabling fast and automated experimentation. It offers two different APIs for defining the machine learning model in the Sequential Class and the functional API (Chollet, 2021, p. 63). In the following compilation step, the optimizers and loss functions for the model are defined. Moreover, Keras provides built-in support for data pre-processing, data augmentation as well as model evaluation, streamlining the entire deep learning development process (Chollet, 2021, pp. 69-77).

2.4 Machine Learning Model for Telecommunications Company: Assumptions

To train the machine learning model, it needs to be provided with customer data. It is assumed that the telecommunications company stores its customer information in a relational database, utilizing either local or cloud storage that can be readily accessed using libraries like Pandas or PyODBC for the Python programming language. Furthermore, it is assumed that the data is retained for at least one year after a customer has left the company. Because actual customer data of home internet telecommunications companies is not publicly available, the dataset is generated within a function of the Python script. This enables the creation of a "perfect world" within the script, providing all data points for every customer. However, if certain data points were missing in a more realistic scenario, they could be interpolated as part of a data preparation step within the same Python script.

The ”generate_customer_data” function of the model creates a relational table with customer data for age, income, usage, satisfaction, postcode, competitors and Churn. To make the dataset reproducible for a comparison between different machine learning frameworks, we first set a random seed for the NumPy library. This allows the creation of reproducible, pseudo-random numbers for age, income, usage, satisfaction, and postcode.

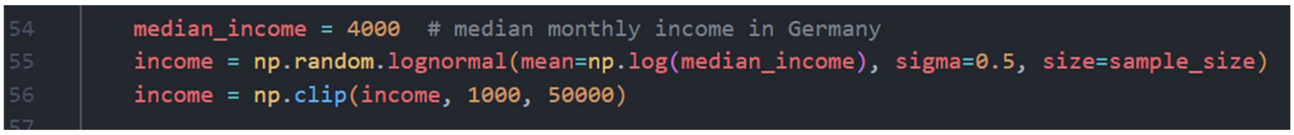


```python
def generate_customer_data(sample_size: int, noise_factor: float):
    np.random.seed(42)
```

Figure 4: Defining the generate_customer_data function with input parameters sample_size and noise_factor, as well as setting a seed for reproducibility.

The age for each customer is a pseudo-randomly generated integer value between 18 and 101 years. It is assumed that customers below the age of 18 are not able to sign a valid contract with the telecommunications company, having to refer to their parents.

The customers’ income values follow a log-normal distribution around Germany’s 50th percentile income of approximately 4000€ per month (Destatis, 2023). In order to control the spread, the standard deviation is set to 0.5. Finally, all values are limited to a range of 1,000€ to 50,000€ (figure 5, 6).



```python
    median_income = 4000  # median monthly income in Germany
    income = np.random.lognormal(mean=np.log(median_income), sigma=0.5, size=sample_size)
    income = np.clip(income, 1000, 50000)
```

Figure 5: Generating income values with a lognormal distribution using NumPy

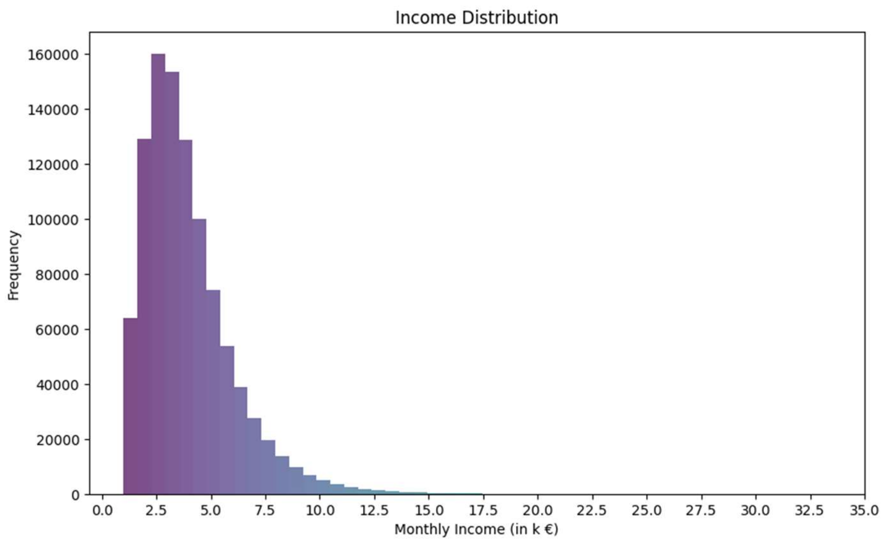
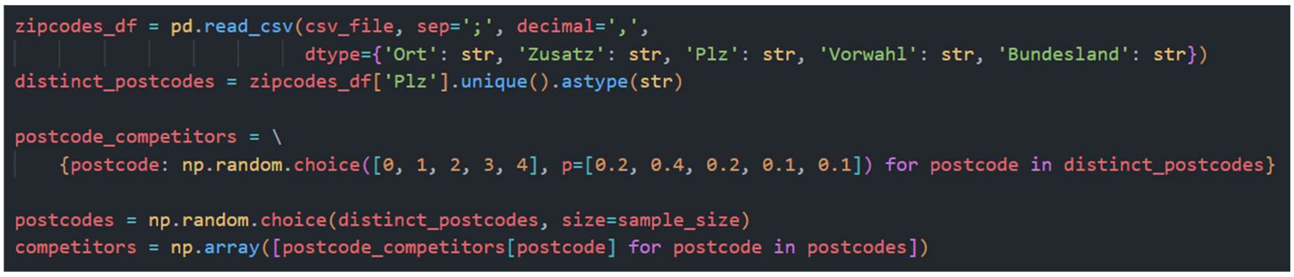


Figure 6: Logarithmic income distribution for a sample size of 10^6 customers

Similar to the customers' age, the usage and satisfaction values are generated as pseudo-random integers, with usage ranging from 0 to 100 and satisfaction from 1 to 10. It is assumed that all customers have recently partaken in a survey, making satisfaction data is widely available.

To assign postcodes to customers, first, a CSV file containing all German postcodes needs to be downloaded[2]. Next, each distinct postcode is assigned a count of competitors, with the most likely scenario presenting one competitor in a postcode (duopoly), followed by no competitors (monopoly) being equally probable as having two competitors, with three and four competitors presenting the least likely scenarios. The mapping of the count of competitors to a postcode is achieved through the key-value pair mechanic of Python dictionaries (figure 7). Finally, a random postcode is drawn from the postcode-competitors dataset for each customer. Since postcodes in Germany range from 01067 to 99998, they are loaded into memory as a string to preserve leading zeros, since loading 01067 as an integer would store it as 1067.



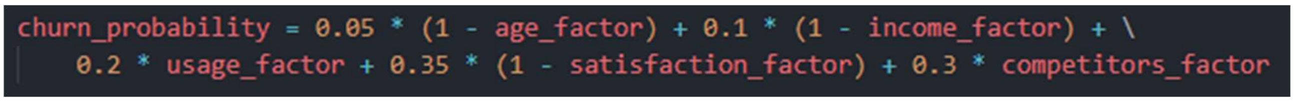
```python
zipcodes_df = pd.read_csv(csv_file, sep=';', decimal=',',
                          dtype={'Ort': str, 'Zusatz': str, 'Plz': str, 'Vorwahl': str, 'Bundesland': str})
distinct_postcodes = zipcodes_df['Plz'].unique().astype(str)

postcode_competitors = \
    {postcode: np.random.choice([0, 1, 2, 3, 4], p=[0.2, 0.4, 0.2, 0.1, 0.1]) for postcode in distinct_postcodes}

postcodes = np.random.choice(distinct_postcodes, size=sample_size)
competitors = np.array([postcode_competitors[postcode] for postcode in postcodes])
```

Figure 7: Loading postcodes and assigning count of competitors through a probability function in NumPy Choice

The churn probability of each customer within the sample dataset is calculated by factoring in the customers age, income, satisfaction, and number of competitors within his or her postcode. First, the four factors are normalized. Then, the churn probability is calculated by weighting each factor (figure 8).



```python
churn_probability = 0.05 * (1 - age_factor) + 0.1 * (1 - income_factor) + \
    0.2 * usage_factor + 0.35 * (1 - satisfaction_factor) + 0.3 * competitors_factor
```

Figure 8: Calculating churn probability

The customer's age has a relatively small impact on the churn probability, with an inverse weighting of 5%. This assumes that younger customers tend to be more tech-savvy, and, therefore, more likely to pick up better offers through comparison of phone providers online. Similarly, the income factor also affects churn probability but has a slightly higher weighting of 10%. It is assumed that customers with a lower income are more prone to churning, as they stand to gain more from saving a few euros in relation to their income. Customers with a higher network usage are more likely to churn, as a reliable connection is crucial for these customers. Higher network usage also indicates that the

[2] Csv file containing all German postcodes: https://gist.github.com/jbspeakr/4565964

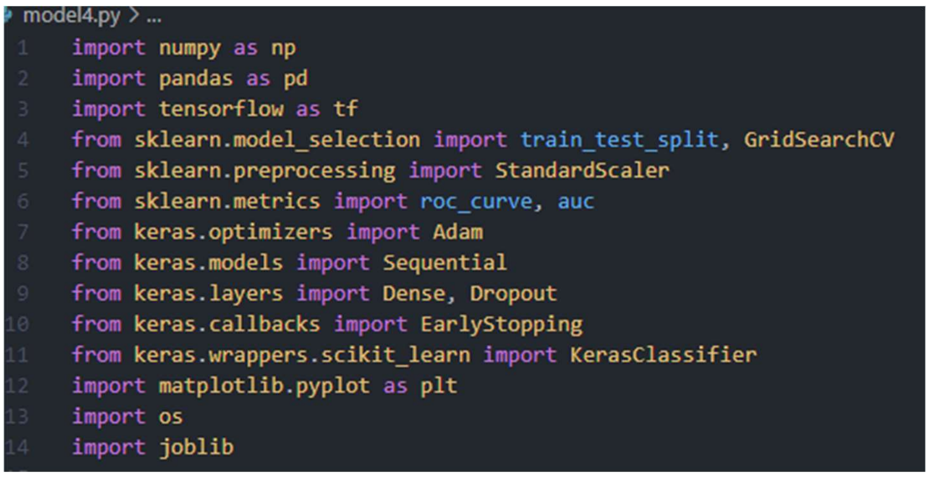
internet and technology play a significant role in the individual's life, which suggests that they are likely to be well-informed about the subject matter.

Customers who rated their satisfaction as low on the satisfaction survey are also significantly more likely to churn. Customer satisfaction is the most significant factor at 35%, slightly surpassing the number of local competitors at 30%. This means that customer churn is highly unlikely, though not impossible, for customers residing in postcodes where the phone company enjoys a monopoly. It is further assumed that the home internet market is subject to extremely inelastic demand, since 95% of German citizens own internet-capable devices, and a lack of home internet connection would render these devices largely useless (Destatis, 2023).

Ultimately, to make the dataset less linear and more realistic, random noise is introduced to the churn probability. This noise can be adjusted through the function's 'noise_factor' parameter (figure 4). The churn probability for each customer, represented as a floating-point value between 0 and 1, is modified by a random noise value between -0.1 and 0.1. At last, the churn threshold is set to 0.65, indicating that every customer with a churn probability of 65% or above (after incorporating noise) is considered to have left the provider in the sample dataset.

2.5 Machine Learning Model for Telecommunications Company: Architecture

The model is implemented using the TensorFlow, SciKit-Learn and Keras libraries (figure 9).[3] SciKit-Learn is used for data pre-processing tasks, such as scaling the features and splitting the data into a training and testing dataset. Additionally, "GridSearchCV" function is used to identify the optimal hyperparameters for the learning rate, dropout rate, and the number of neurons in each layer of the model. "roc_curve" and "auc" are used to plot the receiver operator characteristic (ROC) curve and computation of the area under the curve (AUC) score.



```python
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import roc_curve, auc
from keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.callbacks import EarlyStopping
from keras.wrappers.scikit_learn import KerasClassifier
import matplotlib.pyplot as plt
import os
import joblib
```

Figure 9: Python imports for the machine learning model

[3] Source code for the machine learning model accessible at: https://github.com/leander-ms/ChurnModel/blob/main/model4.py

For a continuous optimization of the churn model in a production environment, several components from the Keras API, a framework running on top of TensorFlow, are imported. Adam is a gradient-based, computationally efficient optimizer (Kingma & Ba, 2017, p. 1-2). Its ability to adjust the learning rate make it well-suited for data-heavy environments, such as those encountered by home internet providers. Sequential and Dense components are used to construct a feedforward network with fully connected layers, enabling the creation of the deep learning neural network. Dropout helps to prevent overfitting by randomly zeroing the outputs of some neurons during training. EarlyStopping further aids in mitigating overfitting by tracking a certain metric during training, (e. g. validation loss in the churn model) and halting the training process if that metric has not improved over a predefined number of epochs. Finally, KerasClassifier allows the usage of SciKit-Learn methods along with a Keras model.

Prior to the construction of the model and input of the data, the data features – age, income, usage, satisfaction, and competitors – must be standardized. This is accomplished by using the "StandardScaler" utility from the SciKit-Learn library. The features are standardized by subtracting their mean and subsequently dividing by their standard deviation. This results in all the features exhibiting a mean of 0 and a standard deviation of 1, rendering them comparable and suitable for deep learning networks.

After standardizing the dataset, it is divided into training and validation subsets. The model is trained on 80% of the scaled data, which constitutes the training dataset. The remaining 20% of the scaled dataset is reserved as the validation dataset for assessing the model's performance later.

The churn model for the telecommunications company features three hidden layers with rectified linear unit (ReLU) activation functions and an output layer employing a sigmoid activation function (figure 10). Rectified linear unit functions are defined as $f(x) = max(0, x)$ and therefore return a value of 0 for any negative input values, whereas positive input values are returned unaltered. This allows the model to learn complex, non-linear patterns, which often can be found in real-world data. Furthermore, rectified linear unit functions are computationally efficient, since the weights and biases of the model are recalculated during the backpropagation phase of the algorithm using the gradient descent criteria. The derivatives of a rectified linear unit function are always either 0 or 1, resulting in more resource efficient computations.

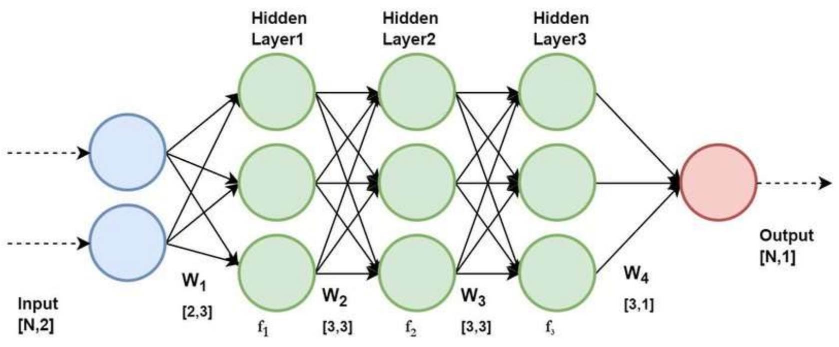
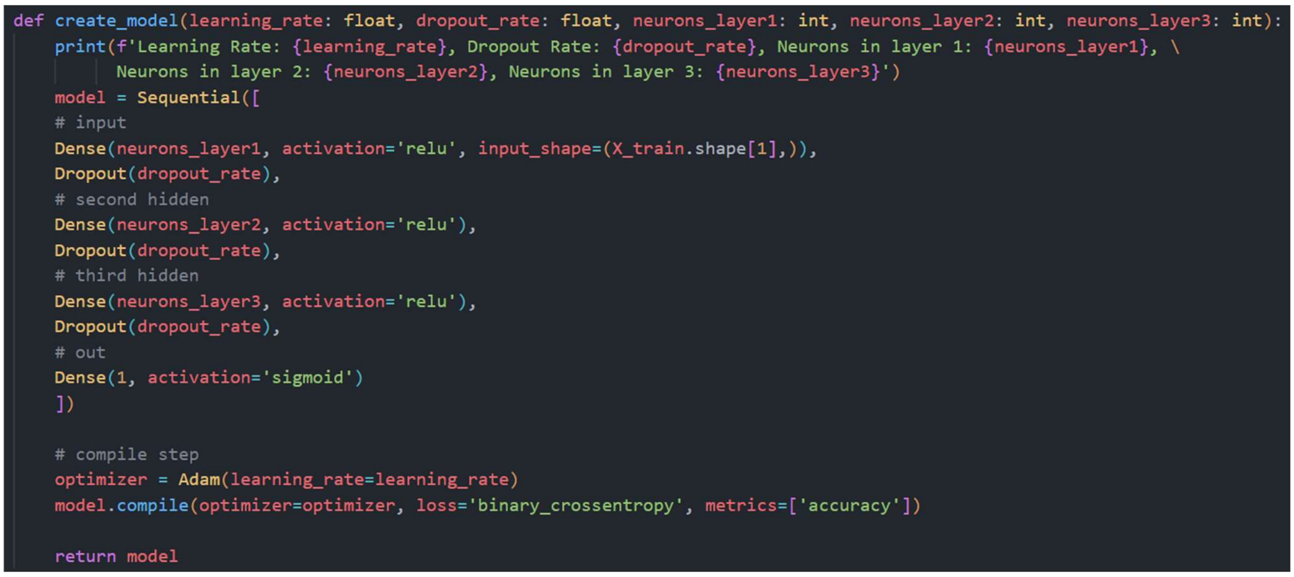


Figure 10: Neural network with three hidden layers (simplified, number of neurons in each layer will be determined by grid search) (Daanouni et al., 2019, p. 3)

Finally, the output layer employs a log-sigmoid function with a single neuron, representing the binary classification problem at hand. The log-sigmoid activation function generates an output between 0 and 1 and is defined as *f(x) = 1/(1+exp(-x))*. By employing a log-sigmoid function in the output layer, the model's output can be interpreted as the probability of a customer churning.



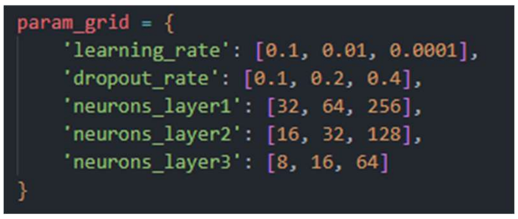
```
def create_model(learning_rate: float, dropout_rate: float, neurons_layer1: int, neurons_layer2: int, neurons_layer3: int):
    print(f'Learning Rate: {learning_rate}, Dropout Rate: {dropout_rate}, Neurons in layer 1: {neurons_layer1}, \
          Neurons in layer 2: {neurons_layer2}, Neurons in layer 3: {neurons_layer3}')
    model = Sequential([
    # input
    Dense(neurons_layer1, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(dropout_rate),
    # second hidden
    Dense(neurons_layer2, activation='relu'),
    Dropout(dropout_rate),
    # third hidden
    Dense(neurons_layer3, activation='relu'),
    Dropout(dropout_rate),
    # out
    Dense(1, activation='sigmoid')
    ])

    # compile step
    optimizer = Adam(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])

    return model
```

Figure 11: Creating the neural network in TensorFlow

The model itself is constructed within a Python function, making it fully adjustable for the grid search, which iteratively adjusts the input parameters of the function to find the model with the highest possible score (figure 11). The "GridSearchCV" module uses cross validation and averages multiple training and testing scores to identify the optimal combination of hyperparameters. The tested hyperparameters are provided to "GridSearchCV" via a dictionary (figure 12).



```
param_grid = {
    'learning_rate': [0.1, 0.01, 0.0001],
    'dropout_rate': [0.1, 0.2, 0.4],
    'neurons_layer1': [32, 64, 256],
    'neurons_layer2': [16, 32, 128],
    'neurons_layer3': [8, 16, 64]
}
```

Figure 12: Dictionary "param_grid" to define the hyperparameters tested by the GridSearch

2.6 Evaluating the Model's Performance

The ideal parameters determined by the grid search were a dropout rate of 10%, a learning rate of 0.0001, 256 neurons in layer 1, 128 neurons in layer 2 and 64 neurons in layer 3. The minimum training loss was captured at 0.1008, with the minimum validation loss captured slightly higher at 0.1015 (figure 13). The training stopped after the 29th epoch to avoid overfitting the data. Overfitting occurs when the model captures not only the underlying patterns in the training data but also its inherent noise, resulting in poor generalization to unseen data (Dietterich, 1995, p. 326-327). The initial training loss was captured at 0.1157, indicating that the model improved its performance on both datasets. The minimum training loss of 0.1008 indicates that the model's churn prediction was off by 10.08% compared to the validation dataset (figure 13).

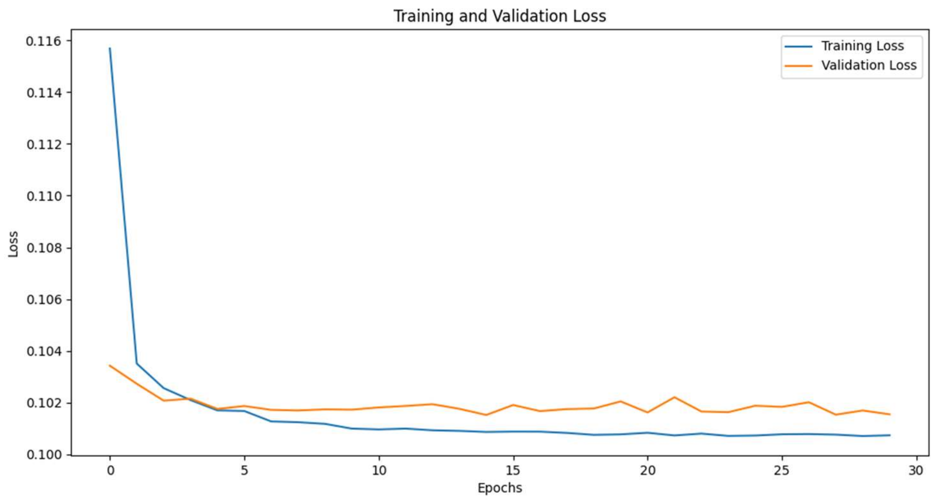


Figure 13: Training and validation loss of the Customer Churn Model through the epochs

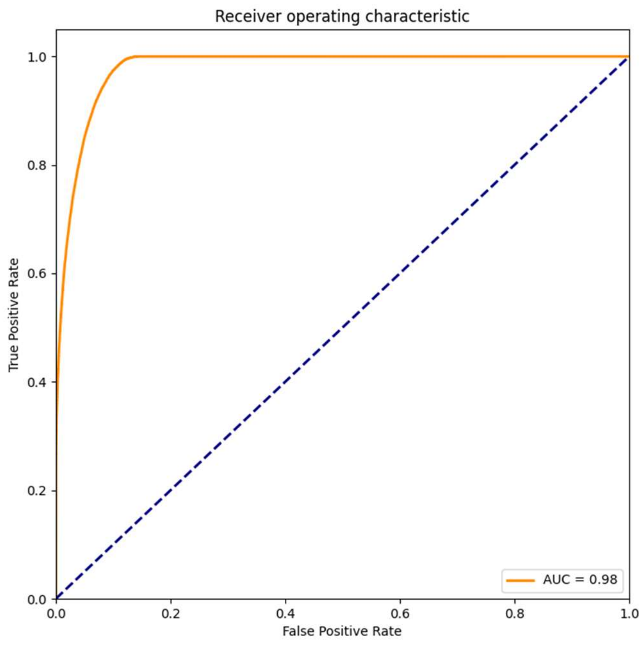


Figure 14: Receiver Operating Characteristic curve and Area under the curve for the customer churn model

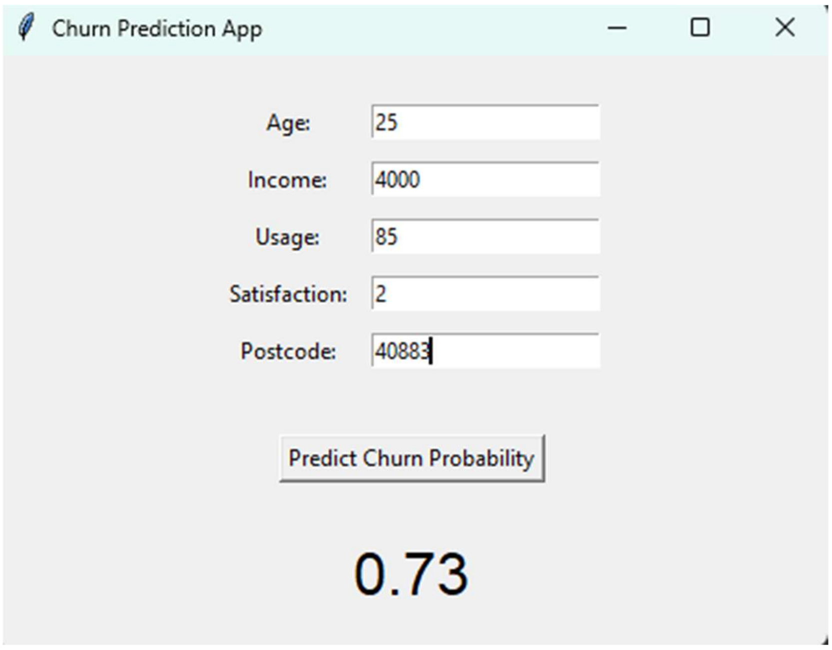
The Receiver Operating Characteristic (ROC) shows the trade-off between the true positive rate and the false positive rate across the various decision thresholds (Greiner et al., 2000, p. 27-28). It is plotted by using the validation datasets. The area under the curve (AUC) is equivalent to the probability that a randomly chosen individual from the positive class will have a higher test score than a randomly chosen individual from the negative class.

Our model shows an area under the curve value of 0.98, highlighting the high accuracy of the customer churn machine learning model (figure 14). An AUC value of 0.5 (dotted blue line) would indicate a model which is non-informative (Greiner et al., 2000, p. 28). Additionally, the ROC curve shows that the model has a high true positive rate (sensitivity) and low false positive rate (specificity), demonstrating outstanding binary classifier performance (figure 14).

2.7 Using the Model inside a Production Environment

After successfully running the machine learning model and saving it along with the scaler, it can now be used in day-to-day operations for predicting customer churns. The flexibility of Python allows for a multitude of options when building GUIs for a machine learning model.

The easiest and lowest maintenance method is to build a simple user interface using the tkinter Python library. First, the previously saved model and scaler are imported into a new Python file (figure 15). After building the user interface and assigning all the input fields with corresponding variable names, a function "predict_churn_probability" will take the user inputs and transform the data using the saved scaler. The function then employs TensorFlow's predict method to predict the churn probability for the provided customer data by loading them into the saved model (figure 15) [4].


Churn Prediction App

Age: 25

Income: 4000

Usage: 85

Satisfaction: 2

Postcode: 40883

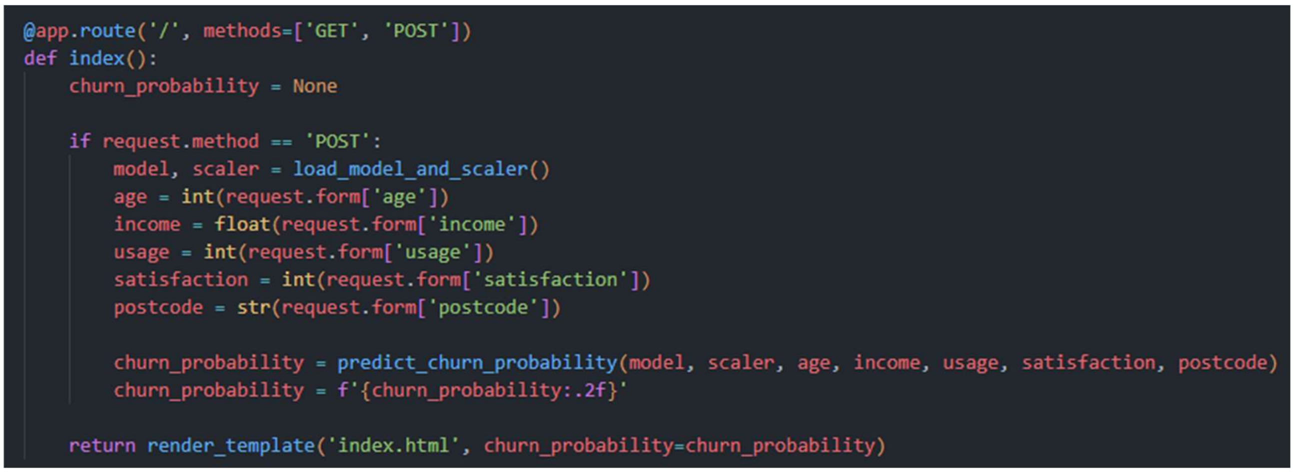
0.73

Figure 15: A simple churn prediction user interface in tkinter

[4] Source code for the tkinter user interface accessible at: https://github.com/leander-ms/ChurnModel/blob/main/use_model4.py

This user interface allows for querying the churn probabilities of multiple users at a time through the "window.mainloop()" functionality inside tkinter.

Another possibility is to build a simple web application using a lightweight framework like Flask. In the case of the home internet provider, this allows the model to run continuously on a production environment server. Employees of the company are then able to input the customer data through the web application, which subsequently loads the data into the model and scaler. The model will then return the customer churn probability. Furthermore, if a customer cancels their contract, the model can automatically be updated as soon as the cancellation is received by the telecommunications company, allowing for continuous fine tuning and adjustments of the model.



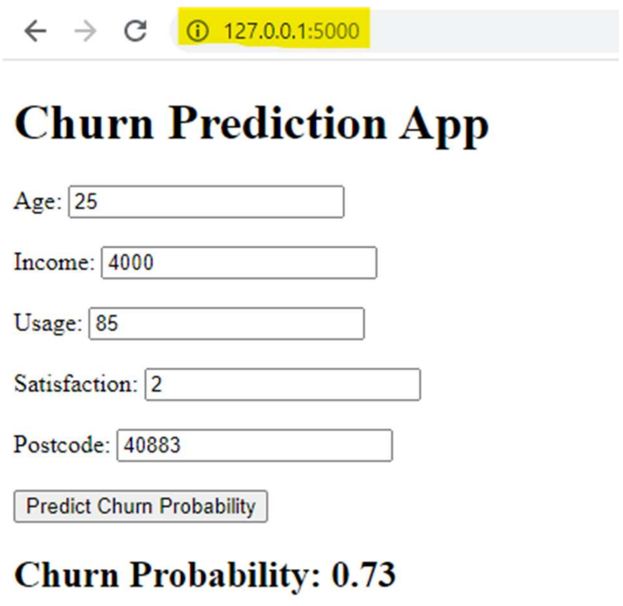
```
@app.route('/', methods=['GET', 'POST'])
def index():
    churn_probability = None

    if request.method == 'POST':
        model, scaler = load_model_and_scaler()
        age = int(request.form['age'])
        income = float(request.form['income'])
        usage = int(request.form['usage'])
        satisfaction = int(request.form['satisfaction'])
        postcode = str(request.form['postcode'])

        churn_probability = predict_churn_probability(model, scaler, age, income, usage, satisfaction, postcode)
        churn_probability = f'{churn_probability:.2f}'

    return render_template('index.html', churn_probability=churn_probability)
```

Figure 16: Posting the application using Flask[5]



127.0.0.1:5000

Churn Prediction App

Age: 25

Income: 4000

Usage: 85

Satisfaction: 2

Postcode: 40883

Predict Churn Probability

Churn Probability: 0.73

Figure 17: Flask web application running on localhost test server with port 5000

[5] Source code for the Python web application accessible at: https://github.com/leander-ms/ChurnModel/blob/main/app.py and https://github.com/leander-ms/ChurnModel/blob/main/templates/index.html

The flexibility of the TensorFlow model inside Python allows for the model to evolve over time, with each iteration sharpening the churn predictions. The implemented grid search also allows for better adaptability of the model, with possible automatic adjustments the learning rate and number of neurons within each layer.

3. Conclusion

In conclusion, home internet telecommunications providers can enhance their customer churn prediction capabilities by developing customized machine learning models. To achieve this, certain prerequisites must be met by the company. It is crucial to ensure that customer data is stored and easily accessible in a high-performance database, which may be hosted either on the cloud or on-premise. Moreover, the company should gather the maximum amount of data permissible within legal constraints. In a production environment, telecommunications companies must also implement sophisticated data preparation solutions, as not all data points will be available for every customer. By addressing these requirements, telecommunications companies can effectively improve the accuracy of their churn predictions, leading to better customer retention strategies and ultimately, business success.

After deploying the customer churn machine learning model in a production environment, minimal resources will be required for its upkeep. This is because the model can automatically adapt overnight to new situations, such as increased customer churn resulting from unforeseen events like the COVID-19 pandemic. By making the model more flexible and self-adjusting, telecommunications companies can ensure it remains effective and relevant in a rapidly changing business landscape.

The model presented in this written assignment also offers several areas for improvement, such as adding additional data points reflecting customer payment regularity. Factors like missed payments or instances where the home internet provider had to issue payment reminders should be integrated into the machine learning model to enhance its predictive capabilities. Additionally, a real-world customer database should also include contact information for each customer.

A fully developed machine learning model running on a production server allows the company to target individuals with a high probability of churning more effectively. For future research, it would be prudent to explore the development of a similar model using an alternative framework, such as PyTorch, enabling a direct performance comparison. Additionally, adding more alterable parameters to the grid search such as the number of hidden layers in the network should be explored.

4. Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., … Zheng, X. (2016). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems* (arXiv:1603.04467). arXiv. http://arxiv.org/abs/1603.04467

Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., & Lloyd, S. (2017). Quantum Machine Learning. *Nature*, *549*(7671), 195–202. https://doi.org/10.1038/nature23474

Capris, T., Melo, P., Garcia, N. M., Pires, I. M., & Zdravevski, E. (2022). Comparison of SQL and NoSQL databases with different workloads: MongoDB vs MySQL evaluation. *2022 International Conference on Data Analytics for Business and Industry (ICDABI), Data Analytics for Business and Industry (ICDABI), 2022 International Conference on*, 214–218. https://doi.org/10.1109/ICDABI56818.2022.10041513

Cardon, D. (2018). Database vs. Data Warehouse: A Comparative Review. *Health catalyst.*

Chaudhuri, S., & Dayal, U. (1997). An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, *26*(1), 65–74. https://doi.org/10.1145/248603.248616

Chauhan, A. (2019). A Review on Various Aspects of MongoDb Databases. *International Journal of Engineering Research*, *8*(05).

Chen, C. L. P. (1996). A rapid supervised learning neural network for function interpolation and approximation. *IEEE Transactions on Neural Networks*, *7*(5), 1220–1230. https://doi.org/10.1109/72.536316

Chollet, F. (2021). *Deep Learning with Python, Second Edition*. Simon and Schuster.

Daanouni, O., Cherradi, B., & Tmiri, A. (2019). *Predicting diabetes diseases using mixed data and supervised machine learning algorithms*. 1–6. https://doi.org/10.1145/3368756.3369072

DB-Engines Ranking. (o. J.). DB-Engines. Abgerufen 20. April 2023, von https://db-engines.com/en/ranking

Dietterich, T. (1995). Overfitting and undercomputing in machine learning. *ACM Computing Surveys*, *27*(3), 326–327. https://doi.org/10.1145/212094.212114

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM*, *55*(10), 78–87. https://doi.org/10.1145/2347736.2347755

Giese, T. G., Wende, M., Bulut, S., & Anderl, R. (2020). Introduction of Data Literacy in the Undergraduate Engineering Curriculum. *2020 IEEE Global Engineering Education Conference (EDUCON)*, 1237–1245. https://doi.org/10.1109/EDUCON45650.2020.9125212

Greiner, M., Pfeiffer, D., & Smith, R. D. (2000). Principles and practical application of the receiver-operating characteristic analysis for diagnostic tests. *Preventive Veterinary Medicine*, *45*(1), 23–41. https://doi.org/10.1016/S0167-5877(00)00115-X

Hackeling, G. (2014). *Mastering machine learning with scikit-learn: Apply effective learning algorithms to real-world problems using scikit-learn*. Packt Publ.

IT-Nutzung. (o. J.). Statistisches Bundesamt. Last updated 25th April 2023, https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Einkommen-Konsum-Lebensbedingungen/IT-Nutzung/_inhalt.html

Jiang, T., Gradus, J. L., & Rosellini, A. J. (2020). Supervised machine learning: A brief primer. *Behavior therapy*, *51*(5), 675–687. https://doi.org/10.1016/j.beth.2020.05.002

Kassambara, A. (2017). *Practical Guide to Cluster Analysis in R: Unsupervised Machine Learning*. STHDA.

Kingma, D. P., & Ba, J. (2017). *Adam: A Method for Stochastic Optimization* (arXiv:1412.6980). arXiv. http://arxiv.org/abs/1412.6980

Li, Y. (2018). *Deep Reinforcement Learning: An Overview* (arXiv:1701.07274). arXiv. https://doi.org/10.48550/arXiv.1701.07274

Moniruzzaman, A. B. M., & Hossain, S. A. (2013). *NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison* (arXiv:1307.0191). arXiv. https://doi.org/10.48550/arXiv.1307.0191

Munjal, R., Arif, S., Wendler, F., & Kanoun, O. (2022). Comparative Study of Machine-Learning Frameworks for the Elaboration of Feed-Forward Neural Networks by Varying the Complexity of Impedimetric Datasets Synthesized Using Eddy Current Sensors for the Characterization of Bi-Metallic Coins. *Sensors*, *22*(4), Article 4. https://doi.org/10.3390/s22041312

Novac, O.-C., Chirodea, M. C., Novac, C. M., Bizon, N., Oproescu, M., Stan, O. P., & Gordan, C. E. (2022). Analysis of the Application Efficiency of TensorFlow and PyTorch in Convolutional Neural Network. *Sensors (Basel, Switzerland)*, *22*(22). https://doi.org/10.3390/s22228872

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*, *32*. https://papers.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

Madhumala, R. B., & Tiwari, H. (2020). Analysis of virtual machine placement and optimization using swarm intelligence algorithms. *Business Intelligence for Enterprise Internet of Things*, 169-183.

Shukla, V., Prashar, S., & Pandiya, B. (2022). Is price a significant predictor of the churn behavior during the global pandemic? A predictive modeling on the telecom industry. *Journal of Revenue and Pricing Management*, *21*(4), 470–483. https://doi.org/10.1057/s41272-021-00367-2

Soriano-Valdez, D., Pelaez-Ballestas, I., Manrique de Lara, A., & Gastelum-Strozzi, A. (2021). The basics of data, big data, and machine learning in clinical practice. *Clinical Rheumatology*, *40*(1), 11–23. https://doi.org/10.1007/s10067-020-05196-z

Unpingco, J. (2016). *Python for Probability, Statistics, and Machine Learning*. Springer International Publishing AG. http://ebookcentral.proquest.com/lib/badhonnef/detail.action?docID=4453022

Verdienste nach Branchen und Berufen. (o. J.). Statistisches Bundesamt. Last updated 25th April 2023, https://www.destatis.de/DE/Themen/Arbeit/Verdienste/Verdienste-Branche-Berufe/_inhalt.html