Artificial Neural Network

Learning Portfolio 4

universität innsbruck

Data Set

The data set is a data set retrieved on kaggle. It contains data about Titanic passengers and whether they survived the crash.

The goal of the data set is to predict whether the passengers died or not.

```
[3] data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

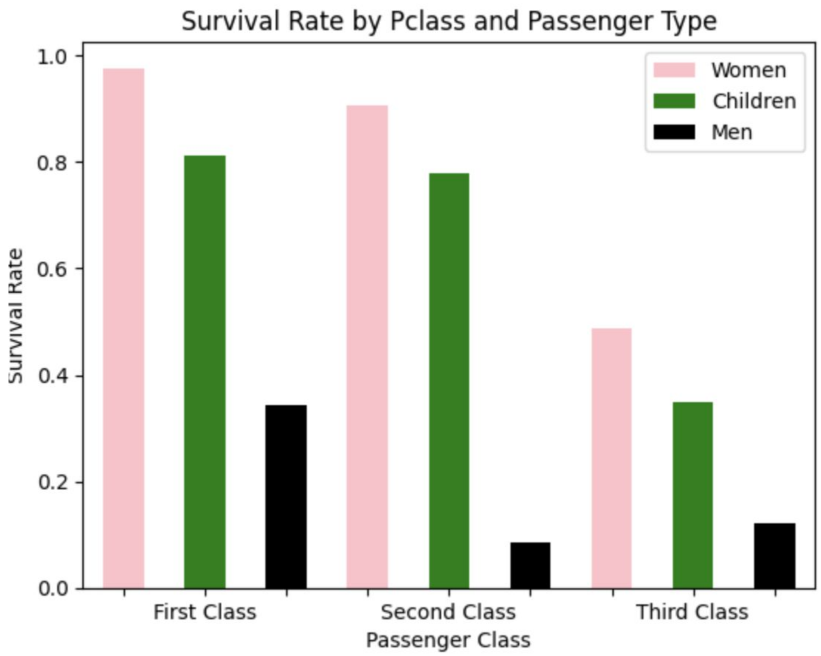
Data Preprocessing

- Missing values in “Age” -> Imputed by clustering
- Missing values in “Embarked” -> Imputed by Mode
- New Feature “Passenger Type” -> Child, Women or Men (ordinal)
- “SibSp”, “Parch” -> Dichotomous reduction
- Logarithmic transformation and scaling
- One-Hot-Encoding for categorical data

```python
# drop not needed data
data = data.drop("Cabin", axis=1)
data = data.drop("Ticket", axis=1)
data = data.drop("Name", axis=1)

# impute passenger age using k-nearest neighbors
from sklearn.impute import KNNImputer

imputer = KNNImputer()
data_num = data.select_dtypes(include=[np.number])
imputer.fit(data_num)
data_new = imputer.transform(data_num)
data["Age"] = data_new[:,3]

# impute embarkation point by mode
from sklearn.impute import SimpleImputer

s_imputer = SimpleImputer(strategy="most_frequent")
s_imputer.fit(data)
data_new = s_imputer.transform(data)
data["Embarked"] = data_new[:,3]

# reduce to boolean attribute
data['SibSp'] = data['SibSp'].apply(lambda x: 0 if x == 0 else 1)
data['Parch'] = data['Parch'].apply(lambda x: 0 if x == 0 else 1)
```

Data Visualization

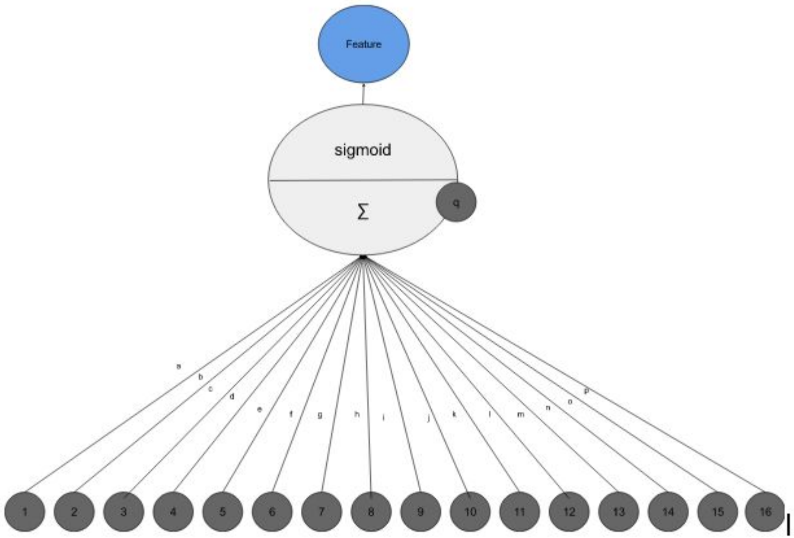
- Women were more likely to survive than children and men
- Thir passenger class was the least likely to survive



universität innsbruck

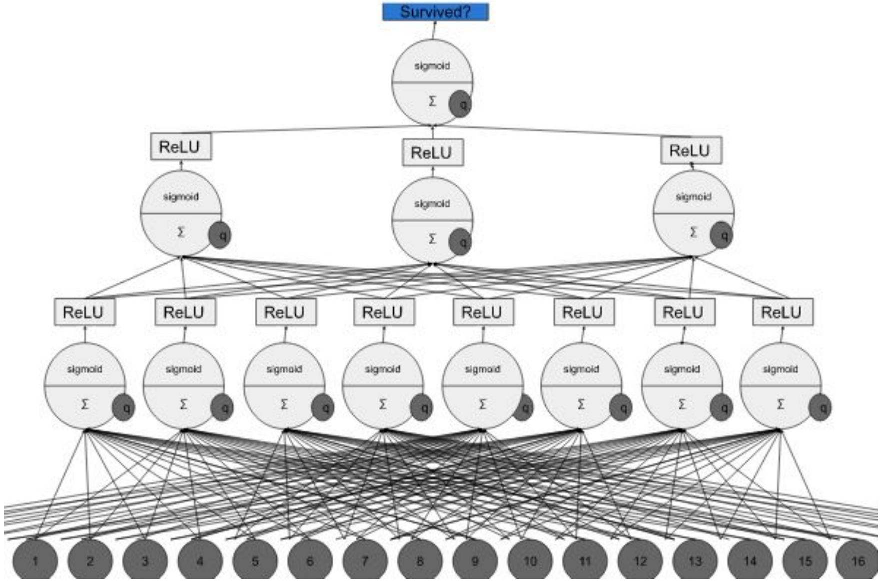
Training

- 8 first-layer perceptrons
- 15 weights (a to p)
- Bias q
- Activation function sigmoid



Training

- Three-Layer-Architecture
- 8 first-layer-perceptrons
- 3 second-layer-perceptrons
- output layer
- Rectified Linear Units as elements of Non-Linearity



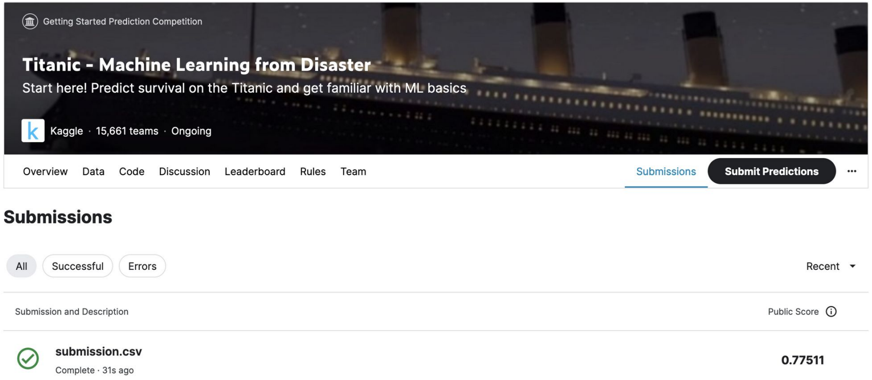
Training

- Loss function: RMSE
- Learning rate: 0.4
- Training for 200 epochs
- Batch size: 200
- Validation metric: accuracy
- 18.29 % wrong predictions on the validation data set
- 82 % accuracy on validation data set

```
[11] # Build the model: 16 -> 8 -> 3 -> Output
simple_net = nn.Sequential(
    nn.Linear(16,8),
    nn.ReLU(),
    nn.Linear(8,3),
    nn.ReLU(),
    nn.Linear(3,1),
)

# Accuracy function
def batch_accuracy(xb, yb):
    xb = xb.sigmoid()
    xb = xb.squeeze()
    correct = (xb>0.5) == yb
    return correct.float().mean()

# Loss function: Root Mean Squared Error
def titanic_loss(predictions, targets):
    predictions = predictions.sigmoid()
    predictions=predictions.squeeze()
    squared_diff = torch.pow(predictions - targets, 2)
    mean_squared_diff = torch.mean(squared_diff)
    rmse = torch.sqrt(mean_squared_diff)

    return rmse

tensor_data = torch.from_numpy(data_prepared)
tensor_label = torch.tensor(data_label.values).squeeze()

tensor_data = tensor_data.float()
tensor_label = tensor_label.float()

# Splitting into training validation data. TO-DO: Randomize
tensor_data, valid_data = torch.split(tensor_data, [tensor_data.shape[0]-100, 100])
tensor_label, valid_label = torch.split(tensor_label, [tensor_label.shape[0]-100, 100])


# The data sets and data loaders for both training and validation.
dl = DataLoader(list(zip(tensor_data,tensor_label)), batch_size=100)
valid_dl = DataLoader(list(zip(valid_data,valid_label)), batch_size=10)
dls = DataLoaders(dl, valid_dl)

# Learner takes test and validation dataset, the model, the SGD for optimization,
# the minst_loss function for calculating the loss, the batch_accuracy for validation
learn = Learner(dls, simple_net, opt_func=SGD,
                loss_func=titanic_loss, metrics=batch_accuracy)
learn.fit(200, 0.4)
plt.plot(L(learn.recorder.values).itemgot(2));
```

Testing

- 77.272 % accuracy
- A little worse than single layer network from Learning Portfolio 3 (77.511 %)
- Overfitting of data or chance? => Significant?
- Above average accuracy for models in kaggle



universität innsbruck

Kontakt

Fabian Leuk
12215478
fabian.leuk@student.uibk.ac.at