**Wine Quality Classification Using Deep Learning**

**Alinur AITEY1**

1Information Technology, School of Computing and Technologies, Eastern Mediterranean University Famagusta, North Cyprus via Mersin 10 Turkey

ORCIS ID: https://orcid.org/0009-0000-6507-6872, aiteyev.alnur@gmail.com

**1 - INTRODUCTION**

This article is devoted to the selection of optimal parameters for learning of Multi-layer perceptron Neural Network. Wine dataset contains fixed\_acidity, volatile\_acidity, citric\_acid, residual\_sugar, chlorides, free\_sulfur\_dioxide, total\_sulfur\_dioxide, density, pH, sulphates, alcohol, quality, color.

**2 – MODEL DESIGN**

This libraries were used:

import pandas as pd  
import numpy as np  
from sklearn.preprocessing import MinMaxScaler  
from sklearn.model\_selection import train\_test\_split  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense  
from tensorflow.keras.optimizers import Adam, SGD, RMSprop, Nadam, Adagrad, Adadelta  
from tensorflow.keras.utils import to\_categorical  
from tensorflow.keras.callbacks import EarlyStopping  
from tqdm.keras import TqdmCallback  
from imblearn.over\_sampling import SMOTE

Model structure

*# A function for creating a neural network model*  
def create\_model(layers, neurons, activation, optimizer):  
 model = Sequential()  
 *# Adding the first hidden layer*  
model.add(Dense(neurons, input\_dim=X\_train.shape[1], activation=activation))  
 *# Adding the remaining hidden layers*  
for \_ in range(layers - 1):  
 model.add(Dense(neurons, activation=activation))  
 *# Output layer*  
model.add(Dense(6, activation='softmax')) *# 6 classes (one-hot encoding)*  
 *# Compiling the model*  
model.compile(optimizer=optimizer, loss='categorical\_crossentropy', metrics=['accuracy'])  
 return model

Feature extraction

X = df.drop('quality', axis=1).values  
y = df['quality'].values

Normalization

scaler = MinMaxScaler()  
X\_scaled = scaler.fit\_transform(X)

Using SMOTE (Synthetic Minority Oversampling Technique) to balance the dataset by generating synthetic samples for minority classes in the target variable.

os=SMOTE()  
x\_res, y\_res = os.fit\_resample(X\_scaled, y\_encoded)

Number of Layers, Neurons, Activation functions, optimizers

activations = ['relu']optimizers = [Nadam, RMSprop, Adam]

Test and Train dataset split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(x\_res, y\_res, test\_size=0.2, random\_state=42)

We choice this parameters because they were best in compare to each other

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Hidden** **Neurons** | **Layers** | **Neurons**  **Per layer** | **Activation** **function** | **Optimizer** | **Epochs** | **Accuracy** | **Loss** | **Val** **Accuracy** | **Val** **Loss** |
| 512 | 2 | 256 | relu | RMSprop | 300 | 0.984 | 0.0398 | 0.867 | 0.991 |
| 512 | 4 | 128 | relu | Nadam | 203 | 0.998 | 0.00991 | 0.863 | 0.998 |
| 2048 | 4 | 512 | relu | Nadam | 119 | 0.972 | 0.0778 | 0.863 | 0.949 |
| 1024 | 8 | 128 | relu | Adam | 152 | 0.966 | 0.109 | 0.863 | 0.96 |
| 1024 | 2 | 512 | relu | Nadam | 235 | 0.993 | 0.0277 | 0.858 | 0.919 |
| 512 | 2 | 256 | relu | Adam | 285 | 0.988 | 0.0387 | 0.856 | 0.749 |
| 1024 | 2 | 512 | relu | RMSprop | 203 | 0.988 | 0.0402 | 0.856 | 0.928 |
| 1024 | 4 | 256 | relu | Adam | 172 | 0.992 | 0.0293 | 0.855 | 1.01 |
| 2048 | 8 | 256 | relu | Nadam | 202 | 0.991 | 0.0292 | 0.853 | 1.21 |
| 4096 | 8 | 512 | relu | RMSprop | 181 | 0.943 | 0.4 | 0.852 | 3.11 |
| 1024 | 16 | 64 | relu | Nadam | 165 | 0.959 | 0.119 | 0.851 | 0.666 |
| 256 | 2 | 128 | relu | RMSprop | 289 | 0.951 | 0.126 | 0.85 | 0.598 |
| 512 | 4 | 128 | relu | RMSprop | 260 | 0.986 | 0.058 | 0.85 | 2.25 |
| 256 | 2 | 128 | relu | Nadam | 214 | 0.942 | 0.163 | 0.848 | 0.505 |
| 1024 | 2 | 512 | relu | Adam | 260 | 0.975 | 0.08 | 0.848 | 0.925 |
| 256 | 4 | 64 | relu | RMSprop | 257 | 0.979 | 0.0663 | 0.848 | 1.18 |
| 512 | 8 | 64 | relu | Adam | 288 | 0.985 | 0.0482 | 0.848 | 0.879 |
| 4096 | 8 | 512 | relu | Adam | 154 | 0.985 | 0.0398 | 0.846 | 1.07 |
| 2048 | 4 | 512 | relu | RMSprop | 150 | 0.979 | 0.0876 | 0.845 | 2 |
| 2048 | 8 | 256 | relu | Adam | 170 | 0.991 | 0.0255 | 0.845 | 1.24 |
| 512 | 4 | 128 | relu | Adam | 184 | 0.994 | 0.0193 | 0.844 | 1.07 |
| 2048 | 4 | 512 | relu | Adam | 131 | 0.987 | 0.0456 | 0.844 | 1.14 |
| 256 | 4 | 64 | relu | Adam | 185 | 0.965 | 0.0921 | 0.841 | 0.761 |
| 1024 | 8 | 128 | relu | RMSprop | 163 | 0.964 | 0.166 | 0.841 | 3.61 |
| 2048 | 16 | 128 | relu | Nadam | 164 | 0.973 | 0.089 | 0.841 | 0.712 |

**3 - IMPLEMENTATION**

To model the system, we used the python programming language and the ready-made tensorflow library. For the test, 20% of the main set were selected in a random set. The learning algorithm is represented by the following code:

history = []  
for i, (model, layers, neurons, activation, optimizer) in enumerate(models):  
 early\_stopping = EarlyStopping(monitor='val\_accuracy', patience=50, restore\_best\_weights=True)  
 history.append(model.fit(X\_train, y\_train, epochs=300, batch\_size=16, validation\_data=(X\_test, y\_test), callbacks=[early\_stopping, TqdmCallback()], verbose=0))

We were interested in achieving the highest accuracy of var\_accuracy, so after 50 epochs of no improvement in accuracy, we stop training the model.

**4 – TRAINING PROCESS**

For training, we used the parameters relative to the compiled table. 16 data samples will be processed in one iteration. Validation data will be taken from the test sample. The progress will be displayed via TqdmCallback(). learning\_rate=0.001

models = []  
model = create\_model(layers=2,  
 neurons=128,  
 activation='relu',  
 optimizer=Nadam(learning\_rate=0.001)  
 )  
models.append((model, 2, 128, 'relu', Nadam))

**5 – EVALUATION**

For example we consider layers=2,neurons=128,activation='relu',optimizer=Nadam(learning\_rate=0.001) for red wine

precision recall f1-score support

0 0.9778 1.0000 0.9888 132

1 0.9143 0.9771 0.9446 131

2 0.7708 0.7400 0.7551 150

3 0.7463 0.6452 0.6920 155

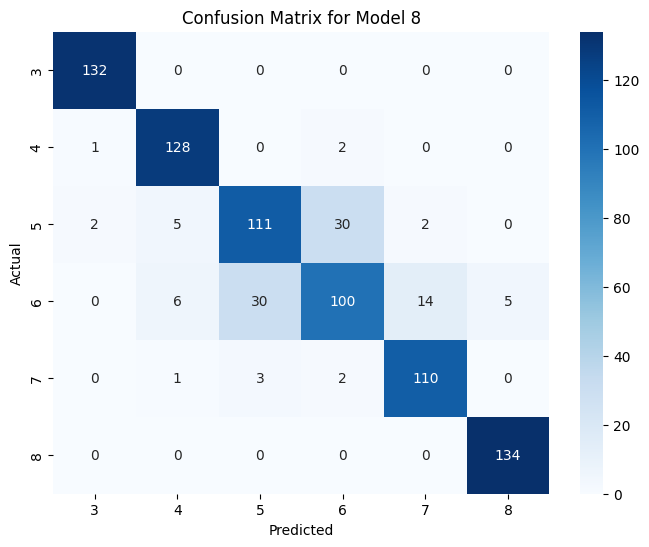
4 0.8730 0.9483 0.9091 116

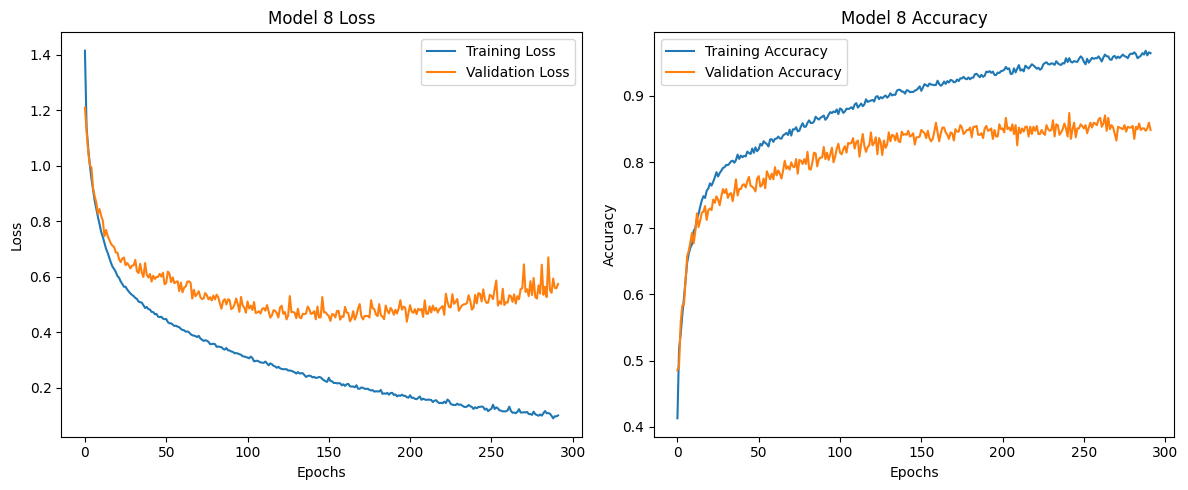
5 0.9640 1.0000 0.9817 134

accuracy 0.8741 818

macro avg 0.8744 0.8851 0.8786 818

weighted avg 0.8687 0.8741 0.8702 818





And for white wine

precision recall f1-score support

0 0.9930 1.0000 0.9965 428

1 0.9577 0.9908 0.9740 434

2 0.7817 0.7633 0.7724 469

3 0.6630 0.6749 0.6689 446

4 0.8658 0.7917 0.8271 432

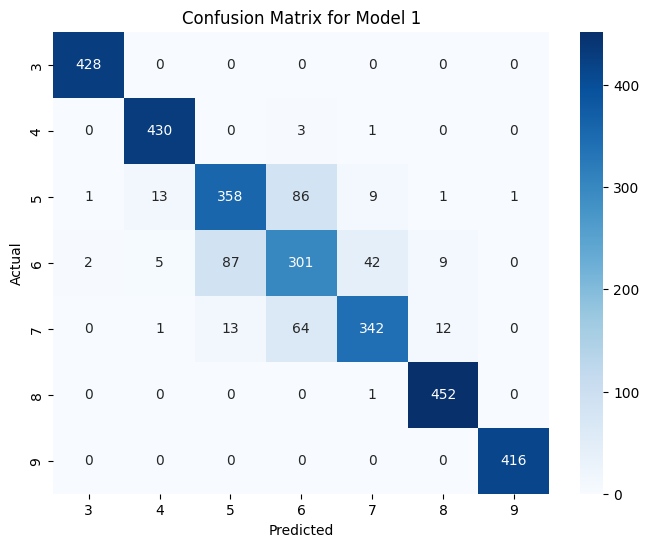
5 0.9536 0.9978 0.9752 453

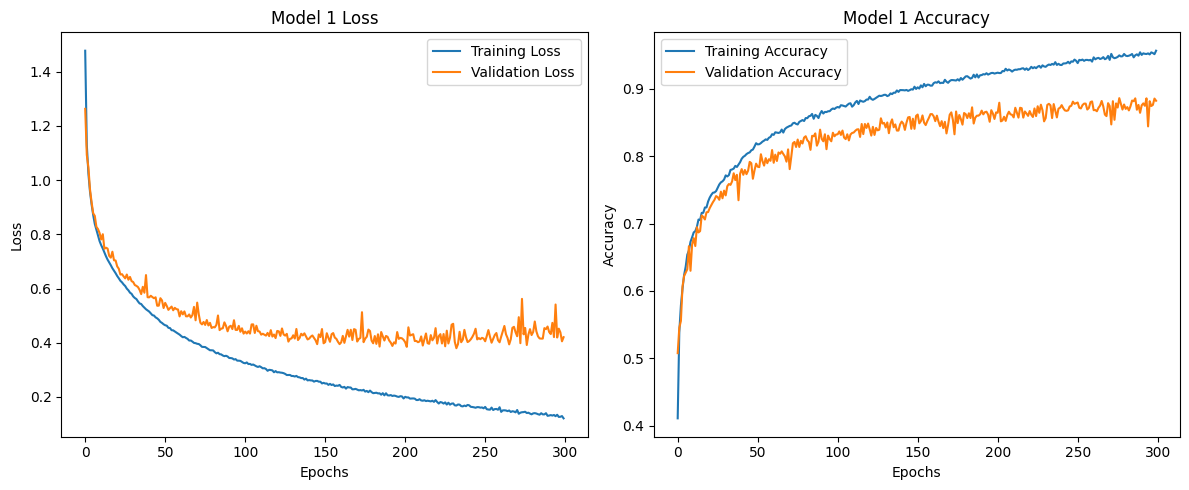
6 0.9976 1.0000 0.9988 416

accuracy 0.8860 3078

macro avg 0.8875 0.8884 0.8875 3078

weighted avg 0.8850 0.8860 0.8851 3078





**6 - CONCLUSION**

With SMOTE and fit\_transform we achieved the highest possible efficiency of the dataset (about 87%) for white and red wines. With other technologies, the accuracy of data can be increased to 90 percent or more. Special attention should be paid to wines 5,6,7 category for white wines and 5,6 for red wines. Since they have the highest number of errors. We also found the optimal neural network structure for this task in terms of ratio to number of neurons and accuracy. It is possible to use cross\_validation for more objective val\_accuracy monitoring.