**4. Neural Netwrok**

**4.1 Neural Network Description**

For our recommendation system, we need to create a Neural Network. This neural network model should be able to analyze the data from the users’ Questionnaires and recommend a News Category to a user.

We solve this as a Classification problem. The **FEATURES** will be the following 5 values from the questionnaire:

1. **Extrovert or Introvert**
2. **Age**
3. **Educational Level**
4. **Sports Interest**
5. **Arts Interest**

The **LABEL** will be the user **Favorite Category**.

Our model will be trained with data from existing users with filled questionnaires and recommend a category to a user who should have also filled his own questionnaire.

**4.2 Neural Network Creation**

The neural network is created by the system’s administrator using our own command line utility. The administrator can create multiple neural network models, but only 1 can be active at each time.

The administration runs the command below in order to create an active neural network model, with name “default-nn”.

poetry run python news\_recommender\_backend/main\_cli.py nn create --name default-nn --active

We implement our model as a Tensorflow Sequential Model as shown in the code below:

def create\_model() -> tf.keras.Sequential:

metrics, \_ = create\_metrics()

learning\_rate: float = 0.01

model = tf.keras.Sequential([

tf.keras.layers.Flatten(input\_shape=(5,)),

tf.keras.layers.Dense(64, activation='relu'),

tf.keras.layers.Dense(128, activation='relu'),

tf.keras.layers.Dense(len(NewsCategoriesEnum), activation='softmax'),

])

model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=learning\_rate),

loss=tf.keras.losses.CategoricalCrossentropy(from\_logits=False),

metrics=metrics)

return model

Our model will have 2 middle layers with 64 and 128 neurons each. The final layer will have neurons equal to the length of our news categories. We use the **softmax** activation function so that the results are probabilities for which is the best suited News Category. The available News Categories in our used api are 7 and will be encoded as following:

1. **business**→ [1 0 0 0 0 0 0]
2. **entertainment**→ [0 1 0 0 0 0 0]
3. **general**→ [0 0 1 0 0 0 0]
4. **health**→ [0 0 0 1 0 0 0]
5. **science**→ [0 0 0 0 1 0 0]
6. **sports**→ [0 0 0 0 0 1 0]
7. **technology**→ [0 0 0 0 0 0 1]

Our model will be trained using the CategoricalCrossentropy loss function.

**4.3 Neural Network Training**

We fetch all the users with filled Questionnaires from our db. We then make 2 sets a TRAIN one (90% of data) and a TEST one (10%) of data. We then create the sets X\_train containing the FEATURES data as described above and Y\_train containing the LABLES data of the TRAIN set. Similarly we create the X\_test and Y\_test data from our TEST set.

We show below the basic code that performs the training:

def user\_to\_feature\_list(user: User) -> list[float]:

assert user.questionnaire is not None

return [1 if user.questionnaire.is\_extrovert else 0, user.questionnaire.age, user.questionnaire.educational\_level, user.questionnaire.sports\_interest, user.questionnaire.arts\_interest]

def user\_to\_label\_one\_hot\_vector(user: User) -> NDArray[float64]:

assert user.questionnaire is not None

one\_hot\_vector = np.zeros(len(NewsCategoriesEnum))

for index, category in enumerate(NewsCategoriesEnum):

logging.info(f"index,category={index},{category}")

if category == user.questionnaire.favorite\_category:

one\_hot\_vector[index] = 1

return one\_hot\_vector

def create\_train\_test\_sets(users: list[User]) -> tuple[NDArray, NDArray, NDArray, NDArray]:

X\_list = list(map(user\_to\_feature\_list, users))

X = np.array(X\_list, dtype=float64)

X = preprocessing.normalize(X, axis=0)

Y\_list = list(map(user\_to\_label\_one\_hot\_vector, users))

Y = np.array(Y\_list, dtype=int64)

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.10, random\_state=42)

return X\_train, X\_test, Y\_train, Y\_test

def train\_model(mymodel: tf.keras.Sequential, metrics\_names: list[str], epochs: int, batch\_size: int, train\_features: NDArray, train\_labels: NDArray, plot\_file: Path):

history = mymodel.fit(train\_features, train\_labels,

epochs=epochs, batch\_size=batch\_size)

# The list of epochs is stored separately from the rest of history.

epochs = history.epoch

# To track the progression of training, gather a snapshot

# of the model's mean squared error at each epoch.

hist = pd.DataFrame(history.history)

plot\_curves(epochs, hist, metrics\_names, plot\_file)

def evaluate\_model(mymodel: tf.keras.Sequential, batch\_size: int, test\_features: NDArray, test\_labels: NDArray):

evaluation = mymodel.evaluate(test\_features, test\_labels, batch\_size=batch\_size)

logging.info('evaluation')

return evaluation

def train\_evaluate\_model(mymodel: tf.keras.Sequential, users: list[User], plot\_file: Path) -> tf.keras.Sequential:

epochs = 80

batch\_size = 10

\_, metrics\_names = create\_metrics()

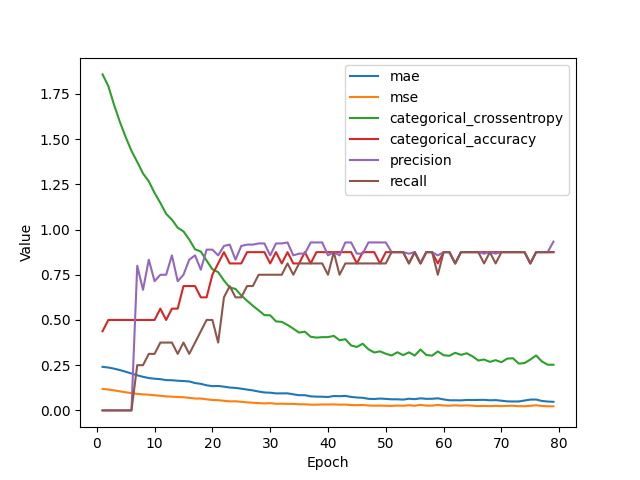
X\_train, X\_test, Y\_train, Y\_test = create\_train\_test\_sets(users)

train\_model(mymodel, metrics\_names, epochs, batch\_size, X\_train, Y\_train, plot\_file)

evaluation = evaluate\_model(mymodel, batch\_size, X\_test, Y\_test)

return mymodel

We show below the metrics values for each epoch during our training:



In order to avoid overfitting, we use only 80 epochs when the CategoricalCrossentropy graph line begins to flatten.

We evaluate our model against our TEST sets:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **mae** | **mse** | **categorical\_crossentropy** | **categorical\_accuracy** | **precision** | **recall** |
| 3.7786965370178223 | 0.2776724696159363 | 0.2316981852054596 | 3.7786965370178223 | 0.0 | 0.0 |

**4.4 Neural Network Predictions**

When the user clicks the “Recommend a Category” button, our backend first selects the active Neural Network from our db. It then uses the user’s questionnaire as input to the predictions of the model. Because, as we have explained earlier, we use the **softmax** actiovation function, the result will be an array of 7 probabilities (each for every category). The one with the highest probability, is the recommended Category we show to the user!

We show the basic code of the prediction functionality below:

def predict\_model(mymodel: tf.keras.Sequential, batch\_size: int, features: list[list[float]]):

logging.debug("start predict\_model")

predictions = mymodel.predict(features, batch\_size=batch\_size)

return predictions

def predict\_model\_for\_user(mymodel: tf.keras.Sequential, user: User) -> NewsCategoriesEnum:

logging.debug("start predict\_model")

batch\_size: int = 10

user\_features = user\_to\_feature\_list(user)

predictions = predict\_model(mymodel, batch\_size, [user\_features])

predicted\_classes = np.argmax(predictions, axis = 1)

predicted\_class = predicted\_classes[0]

predicted\_category: NewsCategoriesEnum | None = None

for index, category in enumerate(NewsCategoriesEnum):

if index == predicted\_class:

predicted\_category = category

assert predicted\_category is not None

return predicted\_category

Our development system has limited number of users (about 20 to 30 users) so the results of the Neural Network are not very accurate. In a production system with thousands of users, the Neural Network will be trained much better and will be much more accurate.