Спочатку я зчитую дані, видаляю з них колонки id та замінюю в таргеті слова negative та positive на 0 та 1.

```
df_labels = pd.read_csv("./data/labels.csv")
df_reviews = pd.read_csv("./data/reviews.csv")
df_labels, df_reviews
```

```
(      id sentiment
 0    168  Positive
 1     26  Positive
 2     64  Positive
 3     10  Positive
 4    111  Negative
 ..   ...       ...
 249  189  Positive
 250   72  Positive
 251  107  Positive
 252  271  Positive
 253  103  Positive

 [254 rows x 2 columns],
       id                                               text
 0    168  Hi Leah, I ordered this early to give to a co-...
 1     26  Good evening, \n  I just received my order! I ...
 2     64  I couldn’t find where I could write a review b...
 3     10  I received my order today and gave it to my si...
 4    111  Max,\n\nWe received the heart but sadly are d...
 ..   ...                                                ...
 249  189    Thank you, this is beautiful and they loved it.
 250   72                    Thanks so much. They lookgreat!
 251  107  Emily, \n  THANK YOU so much for the new “bric...
 252  271  Jacqueline,  \n  I just received the replaceme...
 253  103  Order #(857)982-509708\nI just received my ord...

 [254 rows x 2 columns])
```

```
df_labels.drop("id", axis=1, inplace=True)
df_reviews.drop("id", axis=1, inplace=True)
df = pd.concat([df_labels, df_reviews], axis=1)
sentiment_mapping = {'Negative': 0, 'Positive': 1}
df["sentiment"] = df["sentiment"].map(sentiment_mapping)
df
```

	sentiment	text
0	1	Hi Leah, I ordered this early to give to a co-...
1	1	Good evening, \n I just received my order! I ...
2	1	I couldn’t find where I could write a review b...
3	1	I received my order today and gave it to my si...
4	0	Max,\n\nWe received the heart but sadly are d...

Далі я застосовую препроцесінг до тексту, а саме привожу усі символи до нижнього регістру, видаляю посилання, видаляю всі юзернейми, які починаються на @, також видаляю хештеги, видаляю усі символи які не є буквами або пробілами, заміняю усі повтори однакової літери підряд від 3 та більше на 2 літери, роблю токенізацію усіх слів та видаляю стоп слова і знову зн’єдную все у речення.

```
df['text'] = df['text'].apply(preprocess_text)
df["text"]
```

```
# Text preprocessing function
def preprocess_text(text):
    # Lowercasing
    text = text.lower()

    # Remove URLs (web links) from the text using regular expressions
    text = re.sub(r'http\S+', '', text)

    # Remove mentions (usernames) from the text starting with '@' using regular expressions
    text = re.sub(r'@[A-Za-z0-9]+', '', text)

    # Remove hashtags
    text = re.sub(r'#\S+|\(\([A-Za-z0-9]+\)\)', '', text)

    # Remove any characters that are not alphabetic letters or whitespace using regular expressions
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    # Replace three or more consecutive same letters with two instances
    text = re.sub(r'(\w)\1{2,}', r'\1\1', text)

    # Remove special characters and punctuation
    text = ''.join([char for char in text if char.isalnum() or char.isspace()])

    # Tokenization
    tokens = word_tokenize(text)

    # Remove stop words
    stop_words = set(stopwords.words('english'))
    tokens = [word for word in tokens if word not in stop_words]

    # Join tokens back into a string
    text = ' '.join(tokens)

    return text
```

Далі я розбиваю дані на тестову та навчальну вибірки та векторизую текст використовуючи tf-idf векторайзер.

```
X_train, X_test, y_train, y_test = train_test_split(df.drop("sentiment", axis=1, inplace=False),
                                                    df["sentiment"], test_size=0.5, random_state=42)
```

```
# Create a TF-IDF vectorizer with specified parameters
tfidf_vectorizer = TfidfVectorizer(max_features=10000, stop_words='english')

X_train_tfidf = tfidf_vectorizer.fit_transform(X_train['text'])
X_test_tfidf = tfidf_vectorizer.transform(X_test['text'])
#smote = SMOTE(random_state=42)
#X_train_resampled, y_train_resampled = smote.fit_resample(X_train_tfidf, y_train)
```

Далі я вже порівнюю результати трьох методів реалізованого мною MLPClassifier, вбудованого MLPClassifier та xgboostclassifier.

```
model = CustomMLPClassifier(learning_rate=0.1, num_epochs=10000, hidden_layer_sizes=(50, 80, 100, 70, 40, 1))
model.fit(X_train_tfidf, y_train)
y_pred = model.predict(X_test_tfidf)
```

```
f1 = custom_f1_score(np.array(y_test).reshape(-1, 1), y_pred)
f1
```

```
0.9512195121951219
```

```
from sklearn.neural_network import MLPClassifier
mlp_classifier = MLPClassifier(hidden_layer_sizes=(50, 80, 100, 70, 40),
                               activation='relu',
                               solver='adam',
                               alpha=0.0001,
                               learning_rate_init=0.1,
                               max_iter=10000,
                               random_state=42)
mlp_classifier.fit(X_train_tfidf, y_train)
y_pred = mlp_classifier.predict(X_test_tfidf)
```

```
f1 = custom_f1_score(np.array(y_test).reshape(-1, 1), (y_pred > 0.5).astype(int).reshape(-1, 1))
f1
```

```
0.9285714285714286
```

```
import xgboost as xgb
dtrain = xgb.DMatrix(X_train_tfidf, label=y_train)
dtest = xgb.DMatrix(X_test_tfidf)
params = {
    'objective': 'binary:logistic',
    'eta': 0.1,
    'max_depth': 6,
    'eval_metric': 'logloss'
}
num_round = 100
model = xgb.train(params, dtrain, num_round)
y_pred = model.predict(dtest)
```

```
f1 = custom_f1_score(np.array(y_test).reshape(-1, 1), (y_pred > 0.5).astype(int).reshape(-1, 1))
f1
```

```
0.9629629629629629
```

Тепер розберу як я реалізував кастомний млп.

```python
class CustomMLPClassifier:
    def __init__(self, learning_rate=0.01, num_epochs=100, hidden_layer_sizes=(100,)):
        self.learning_rate = learning_rate
        self.num_epochs = num_epochs
        self.hidden_layer_sizes = hidden_layer_sizes

    def fit(self, X, y):
        # Convert input data and labels to appropriate formats
        X = X.toarray()
        y = np.array(y)
        m = X.shape[0] # Number of training examples
        self._w = []
        self._b = []

        # Initialize weights and biases for each hidden layer
        for layer_idx in range(len(self.hidden_layer_sizes)):
            if layer_idx == 0:
                n_in = X.shape[1]
            else:
                n_in = self.hidden_layer_sizes[layer_idx-1]
            n_out = self.hidden_layer_sizes[layer_idx]
            limit = np.sqrt(6 / (n_in + n_out))
            self._w.append(np.random.uniform(-limit, limit, size=(n_in, n_out)))
            self._b.append(0.0)

        # Training loop
        for epoch in range(self.num_epochs):
            # Forward propagation
            cache = []
            A = X
            Z = []
            for l in range(len(self._w)):
                A_prev = A

                # Compute the linear transformation
                Z.append(np.dot(A_prev, self._w[l]) + self._b[l])
                linear_cache = (A_prev, self._w[l], self._b[l])

                if l != len(self._w) - 1:
                    # Apply ReLU activation function for hidden layers
                    A = np.maximum(0, Z[l])
                else:
                    # Apply sigmoid activation function for the output layer
                    A = 1 / (1 + np.exp(-Z[l]))
                activation_cache = Z[l]
                cache.append((linear_cache, activation_cache))

            # Compute the BCEWithLogitsCost
            cost = (-1 / m) * np.sum(y * np.log(A).T + (1 - y) * np.log(1 - A).T)

            # Backpropagation
            y = y.reshape(A.shape)
```

```
            m = A.shape[0]
            parameter_w = []
            parameter_b = []

            dA = - (np.divide(y, A) - np.divide(1 - y, 1 - A))

            current_cache = cache[-1]
            linear_cache, activation_cache = current_cache
            Z = activation_cache
            A_prev, W, b = linear_cache

            dZ = dA * (np.exp(-Z) / (1 + np.exp(-Z)) ** 2)
            dA_prev = np.dot(W, dZ.T).T
            dW = (1 / m) * np.dot(dZ.T, A_prev)
            db = (1 / m) * np.sum(dZ, axis=0)
            parameter_w.append(dW)
            parameter_b.append(db)

            # Backpropagate through hidden layers
            for l in reversed(range(len(self._w) - 1)):
                current_cache = cache[l]
                linear_cache, activation_cache = current_cache
                Z = activation_cache
                A_prev, W, b = linear_cache
                m = A_prev.shape[0]

                dZ = dA_prev * np.int64(Z > 0) # Backpropagate through ReLU activation
                dA_prev = np.dot(W, dZ.T).T
                dW = (1 / m) * np.dot(dZ.T, A_prev)
                db = (1 / m) * np.sum(dZ, axis=0)
                parameter_w.append(dW)
                parameter_b.append(db)

            # Update parameters using gradient descent
            parameter_w = copy.deepcopy(parameter_w)
            parameter_b = copy.deepcopy(parameter_b)
            for l in range(len(self._w)):
                self._w[l] = self._w[l] - self.learning_rate * parameter_w[-(l+1)].T
                self._b[l] = self._b[l] - self.learning_rate * parameter_b[-(l+1)].T


    def predict(self, X):
        # Convert input data to the appropriate format
        X = X.toarray()
        m = X.shape[0]
        y_prediction = np.zeros((m, 1))
        A = X
        A_prev = A
```

```python
# Forward propagation for prediction
for l in range(len(self._w)):
    if l != len(self._w) - 1:
        # Apply ReLU activation for hidden layers
        A = np.maximum(0, np.dot(A_prev, self._w[l]) + self._b[l])
    else:
        # Apply sigmoid activation for the output layer
        A = 1 / (1 + np.exp(-(np.dot(A_prev, self._w[l]) + self._b[l])))
    A_prev = A

# Convert probabilities to binary predictions (0 or 1)
for i in range(A.shape[0]):
    if A[i, 0] > 0.5 :
        y_prediction[i, 0] = 1
    else:
        y_prediction[i, 0] = 0

return y_prediction
```

Спочатку я ініціалізую ваги методом Xavier, далі я запускаю цикл по кількості епох та вбудований цикл ще по різним леєрам та для кожного лежру розраховую функцію активацію та зберігаю кеші, після цього обраховую кост функцію. Після цього робимо зворотнє поширення та апдейтимо ваги. Ну і в методі предікт просто застосовуємо натреновані ваги та зміщення.