ДОДАТОК А

Функція для очистки тексту

Лістинг А.1 – Функція для очистки тексту

```
    def clean_text(text):
        text = text.lower().strip()
        text = ''.join(char for char in text if not
emoji.is_emoji(char))
        text = contractions.fix(text)
        try:
            lang = detect(text)
            if lang != 'en':
                return ''
        except LangDetectException:
            return ''
        text = re.sub(r'\r|\n', ' ', text)
        text = re.sub(r"(?:\@|https?\://)\S+", "", text)
        text = re.sub(r'[^\x00-\x7f]', '', text)
        banned_list = string.punctuation
        table = str.maketrans('', '', banned_list)
        text = text.translate(table)
        text = ' '.join('' if ('$' in word) or ('&' in word)
else word
                        for word in text.split())
        text = re.sub(r'(\s+#[\w-]+)+\s*$', '', text).strip()
        text = re.sub(r'#([\w-]+)', r'\1', text).strip()
        text = ' '.join(
            word for word in text.split()
            if ('$' not in word and '&' not in word and word
not in stop_words))
        text = re.sub(r"\s\s+", " ", text)
        text = re.sub(r'\d+', '', text)
        words = word_tokenize(text)
        lemmatized_words = [lemmatizer.lemmatize(word) for word
in words]
```

```
        text = ' '.join(lemmatized_words)
        words = text.split()
        long_words = [word for word in words if len(word) >= 3]
        text = ' '.join(long_words)
        text      =      re.sub(r'\b(\w+)((\w)\3{2,})(\w*)\b',
r'\1\3\4', text)
        text = re.sub(r'[\?\.\!]+(?=[\?\.\!])', '', text)
        text = re.sub(
    r'(?:http[s]?://)?(?:www\.)?(?:bit\.ly|goo\.gl|t\.co|tinyu
rl\.com|tr\.im|is\.gd|cli\.gs|u\.nu|url\.ie|tiny\.cc|alturl\.c
om|ow\.ly|bit\.do|adoro\.to)\S+',
            '', text)
        text = text.strip()
        text = text if len(words) >= 2 else ""
        return ' '.join(text.split())
```

ДОДАТОК Б

Створення та навчання багатошарової лінійної мережі

Лістинг Б.1 – Програмний код створення та навчання багатошарової лінійної мережі

```
linear_model = LinearClassifier(vocab_size=VOCAB_SIZE,

embedding_dim=EMBEDDING_DIM,
                                    sequence_length=max_len,
                                    num_classes=5)
linear_model.embedding.weight.data.copy_(torch.from_numpy(
embedding_matrix))
linear_model.embedding.weight.requires_grad = True
EPOCHS = 20
early_stopping_patience = 4
early_stopping_counter = 0
valid_acc_max = 0
criterion = nn.CrossEntropyLoss()
optimizer  =  torch.optim.Adam(linear_model.parameters(),
lr=1e-3)
linear_model.to('cpu')
train_loss_history = []
valid_loss_history = []
train_acc_history = []
valid_acc_history = []
total_step = len(train_loader)
total_step_val = len(valid_loader)
for e in range(EPOCHS):
    train_loss, valid_loss = [], []
    train_acc, valid_acc = [], []
    y_train_list, y_val_list = [], []
    correct, correct_val = 0, 0
    total, total_val = 0, 0
    running_loss, running_loss_val = 0, 0
    linear_model.train()
```

```
        for inputs, labels in train_loader:
            inputs,    labels    =    inputs.long().to("cpu"),
labels.to("cpu")
            optimizer.zero_grad()
            output = linear_model(inputs)
            loss = criterion(output, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            y_pred_train = torch.argmax(output, dim=1)
            y_train_list.extend(y_pred_train.cpu().tolist())
            correct += (y_pred_train == labels).sum().item()
            total += labels.size(0)
        train_loss.append(running_loss / total_step)
        train_acc.append(100 * correct / total)
        model.eval()
        with torch.no_grad():
            for inputs, labels in valid_loader:
                inputs,   labels   =   inputs.long().to("cpu"),
labels.to("cpu")
                output = linear_model(inputs)
                val_loss = criterion(output, labels)
                running_loss_val += val_loss.item()
                y_pred_val = torch.argmax(output, dim=1)
                y_val_list.extend(y_pred_val.cpu().tolist())
                correct_val      +=      (y_pred_val      ==
labels).sum().item()
                total_val += labels.size(0)
        valid_loss.append(running_loss_val / total_step_val)
        valid_acc.append(100 * correct_val / total_val)
        train_loss_history.append(np.mean(train_loss))
        valid_loss_history.append(np.mean(valid_loss))
        train_acc_history.append(np.mean(train_acc))
        valid_acc_history.append(np.mean(valid_acc))
        if valid_acc_history[-1] >= valid_acc_max:
```

```
            torch.save(linear_model.state_dict(),
'models/linear_model.pt')
            print(f"Epoch {e+1}: Validation accuracy increased
({valid_acc_max:.4f}  →  {valid_acc_history[-1]:.4f}).  Saving
model.")
            valid_acc_max = valid_acc_history[-1]
            early_stopping_counter = 0
        else:
            print(f"Epoch {e+1}: Validation accuracy did not
increase.")
            early_stopping_counter += 1
        if early_stopping_counter > early_stopping_patience:
            print(f"Early stopped at epoch {e+1}")
            break
        print(f"\tTrain_loss:     {train_loss_history[-1]:.4f}
Val_loss: {valid_loss_history[-1]:.4f}")
        print(f"\tTrain_acc:      {train_acc_history[-1]:.2f}%
Val_acc: {valid_acc_history[-1]:.2f}%")
```

ДОДАТОК В

Архітектура LSTM мережі з механізмом уваги

Лістинг В.1 – Програмний код архітектури LSTM мережі з механізмом уваги

```
class AttentionLayer(nn.Module):
    def __init__(self, hidden_dim):
        super(AttentionLayer, self).__init__()
        self.attn = nn.Linear(hidden_dim * 2, hidden_dim)
        self.v = nn.Linear(hidden_dim, 1, bias=False)
    def forward(self, hidden, encoder_outputs):
        seq_len = encoder_outputs.size(1)
        hidden = hidden[-1]
        hidden_repeated = hidden.unsqueeze(1).repeat(1,
seq_len, 1)
        attn_weights =
torch.tanh(self.attn(torch.cat((hidden_repeated,
encoder_outputs), dim=2)))
        attn_weights = self.v(attn_weights).squeeze(2)
        return nn.functional.softmax(attn_weights, dim=1)

class LSTMClassifier(nn.Module):
    def __init__(self, vocab_size, embedding_dim,
hidden_dim, num_classes, dropout):
        super(LSTMClassifier, self).__init__()
        self.hidden_dim = hidden_dim
        self.embedding = nn.Embedding(vocab_size,
embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim,
batch_first=True)
        self.attention = AttentionLayer(hidden_dim)
        self.fc = nn.Linear(hidden_dim, num_classes)
        self.softmax = nn.LogSoftmax(dim=1)
        self.dropout = nn.Dropout(dropout)
    def forward(self, x, hidden):
```

```
            embedded = self.embedding(x)
            out, hidden = self.lstm(embedded, hidden)
            attn_weights = self.attention(hidden[0], out)
            context =
attn_weights.unsqueeze(1).bmm(out).squeeze(1)
            out = self.softmax(self.fc(context))
            return out, hidden
        def init_hidden(self, batch_size):
            factor = 1
            h0 = torch.zeros(factor, batch_size,
self.hidden_dim).to(DEVICE)
            c0 = torch.zeros(factor, batch_size,
self.hidden_dim).to(DEVICE)
            return (h0, c0)
```

ДОДАТОК Г

Створення та навчання LSTM мережі з механізмом уваги

Лістинг Г.1 – Програмний код створення та навчання LSTM мережі з механізмом уваги

```
NUM_CLASSES = 5
HIDDEN_DIM = 100
LR = 1e-3
DROPOUT = 0.5
EPOCHS = 10
lstm_model = LSTMClassifier(VOCAB_SIZE, EMBEDDING_DIM, HIDDEN_DIM, NUM_CLASSES, DROPOUT)
lstm_model = lstm_model.to('cpu')
lstm_model.embedding.weight.data.copy_(torch.from_numpy(embedding_matrix))
lstm_model.embedding.weight.requires_grad = True
criterion = nn.NLLLoss()
optimizer = torch.optim.AdamW(lstm_model.parameters(), lr=LR)
total_step = len(train_loader)
total_step_val = len(valid_loader)
train_loss_history = []
valid_loss_history = []
train_acc_history = []
valid_acc_history = []
early_stopping_patience = 4
early_stopping_counter = 0
valid_acc_max = 0
for e in range(EPOCHS):
    train_loss, valid_loss = [], []
    train_acc, valid_acc = [], []
    y_train_list, y_val_list = [], []
    correct, correct_val = 0, 0
    total, total_val = 0, 0
    running_loss, running_loss_val = 0, 0
```

```
        lstm_model.train()
        for inputs, labels in train_loader:
            inputs, labels = inputs.to('cpu'), labels.to('cpu')
            h = lstm_model.init_hidden(labels.size(0))
            lstm_model.zero_grad()
            output, h = lstm_model(inputs,h)
            loss = criterion(output, labels)
            loss.backward()
            running_loss += loss.item()
            optimizer.step()
            y_pred_train = torch.argmax(output, dim=1)

y_train_list.extend(y_pred_train.squeeze().tolist())
            correct += torch.sum(y_pred_train==labels).item()
            total += labels.size(0)
        train_loss.append(running_loss / total_step)
        train_acc.append(100 * correct / total)
        with torch.no_grad():
            lstm_model.eval()
            for inputs, labels in valid_loader:
                inputs,     labels     =     inputs.to('cpu'),
labels.to('cpu')
                val_h = lstm_model.init_hidden(labels.size(0))
                output, val_h = lstm_model(inputs, val_h)
                val_loss = criterion(output, labels)
                running_loss_val += val_loss.item()
                y_pred_val = torch.argmax(output, dim=1)

y_val_list.extend(y_pred_val.squeeze().tolist())
                correct_val                                    +=
torch.sum(y_pred_val==labels).item()
                total_val += labels.size(0)
            valid_loss.append(running_loss_val                 /
total_step_val)
            valid_acc.append(100 * correct_val / total_val)
        train_loss_history.append(np.mean(train_loss))
```

```
        valid_loss_history.append(np.mean(valid_loss))
        train_acc_history.append(np.mean(train_acc))
        valid_acc_history.append(np.mean(valid_acc))
        if np.mean(valid_acc) >= valid_acc_max:
            torch.save(lstm_model.state_dict(),
'models/lstm_model.pt')
            print(f'Epoch {e+1}:Validation accuracy increased
({valid_acc_max:.6f}  -->  {np.mean(valid_acc):.6f}).    Saving
model ...')
            valid_acc_max = np.mean(valid_acc)
            early_stopping_counter=0
        else:
            print(f'Epoch  {e+1}:Validation  accuracy  did  not
increase')
            early_stopping_counter+=1
        if early_stopping_counter > early_stopping_patience:
            print('Early stopped at epoch :', e+1)
            break
        print(f'\tTrain_loss    :    {np.mean(train_loss):.4f}
Val_loss : {np.mean(valid_loss):.4f}')
        print(f'\tTrain_acc    :    {np.mean(train_acc):.3f}%
Val_acc : {np.mean(valid_acc):.3f}%')
```

ДОДАТОК Д

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	99 с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

.	.	Прізвище та ініціали.	Підп.	Дата	
Розробив		Слюсаренко Т.А.			Розробка системи виявлення кібербулінгу на основі методів обробки природної мови
Перевірив		Погурська М.М.			
Н.контр.		Малєєва І.А.			
Затв.		Золотухін О.В.			

Шифр групи	Код напр./спец.
ІТШІ-21-1	122
ХНУРЕ кафедра ШІ	