ДОДАТОК А

Лістинг коду алгоритм

```
#Завантаження даних з Excel
excel_file = '/Users/pogremuhalike/PycharmProjects/pythonProject2/dataset.xlsx'
df = pd.read_excel(excel_file)

# Попередня обробка стовпця 'Приклад'
df['Приклад'] = df['Приклад'].str.replace(r'\d+\.', '') # видаляємо порядкові номери
df['Приклад'] = df['Приклад'].str.replace(r'«', '') # видаляємо двійні лапки
# Функція чищення тексту
def preprocess_text(document):
    document = re.sub(r'\W', ' ', str(document)) # видаляємо особливі символи
    document = re.sub(r'\s+[a-zA-Z]\s+', ' ', document) # видаляємо всі одинарні символи
    document = re.sub(r'\s+', ' ', document, flags=re.I) # замінюємо декілька пробілів одним пробілом
    document = document.lower() # переводимо рядок у нижній регістр
    tokens = document.split() # розділяємо рядок на токени
    return tokens

# Передобробка стовпця 'message'
df['Приклад'] = df['Приклад'].apply(preprocess_text)

# Розділення даних на навчальну та тестову вибірки
```

```
X_train, X_test, y_train, y_test = train_test_split(df['Приклад'], df['ТИП події'], test_size=0.2)

# Нам також потрібно перетворити списки токенів назад на рядки, щоб їх можна було ввести в CountVectorizer
X_train = [' '.join(x) for x in X_train]
X_test = [' '.join(x) for x in X_test]

# Векторизація тексту
vectorizer = CountVectorizer()
X_train_vect = vectorizer.fit_transform(X_train)

# Тренування моделі
clf = MultinomialNB()
clf.fit(X_train_vect, y_train)

# Перевірка моделі на тестовому наборі
X_test_vect = vectorizer.transform(X_test)
y_pred = clf.predict(X_test_vect)



loaded_clf = pickle.load(open(«naive_bayes_model.pkl», «rb»))
loaded_vectorizer = pickle.load(open(«count_vectorizer.pkl», «rb»))
```

Лістинг 1 – Лістинг коду класифікатора

```
def get_meeting_details(text):
    translator = Translator()
    nlp = spacy.load(«en_core_web_sm»)
    translation = translator.translate(text, src='uk', dest='en')
    translated_text = translation.text
    doc = nlp(translated_text)

    date = [ent.text for ent in doc.ents if ent.label_ == 'DATE']
    time = [ent.text for ent in doc.ents if ent.label_ == 'TIME']
    location = [ent.text for ent in doc.ents if ent.label_ in ['GPE', 'ORG']]

    return date, time, location

text = «Планується зустріч у парку Горького з друзями на 12:30. « \
      «Цу буде субота 20 травня «

meeting_date, meeting_time, meeting_location = get_meeting_details(text)

print(f'Date: {meeting_date}')
print(f'Time: {meeting_time}')
print(f'Location: {meeting_location}')

def predict_meeting_type(text):
    model = pickle.load(open(«naive_bayes_model.pkl», «rb»))
    vectorizer = pickle.load(open(«count_vectorizer.pkl», «rb»))

    text_vect = vectorizer.transform([text])

    predicted_type = model.predict(text_vect)
```

```
    return predicted_type[0]
print(predict_meeting_type(text))
```

Лістинг 2 – Лістинг коду обробки повідомлень

```
def eval_schedule(individual):
    f = user_criteria
cheduling = can_meetings_be_scheduled(individual)
    if not scheduling:
        return -10
    num_meetings = sum(scheduling)
    total_len = sum((meetings[i]['duration'] if 'duration' in meetings[i] else
0) for i in range(len(scheduling)) if scheduling[i] == 1)
    meeting_type_count = {type: 0 for type in priorities.keys()}
    priority_score = 0
    user_score = 0
    break_len = break_time(scheduling)
    for i in range(len(scheduling)):
        if scheduling[i] == 1:
            meeting_type_count[meetings[i]['type']] += 1
    priority_score = sum(meeting_type_count[_type] * priorities[_type] for
_type in meeting_type_count)
    user_score += f[«user_score»]

    for i in range(len(scheduling)):
        if scheduling[i] == 0 and meetings[i]['type'] == 'work' and
meetings[i]['mandatory']:
            user_score -= 100
    fitness = f[«priority_score»] * priority_score \
            + user_score \
```

```
                + f[«break_len»] * break_len \
                + f[«num_meetings»] * num_meetings \
                + f[«total_len»] * total_len
        return fitness,
    creator.create(«FitnessMax», base.Fitness, weights=(1.0,))
    creator.create(«Individual», list, fitness=creator.FitnessMax)
    toolbox = base.Toolbox()
    toolbox.register(«attr_bool», random.randint, 0, 1)
    toolbox.register(«individual», tools.initRepeat, creator.Individual,
toolbox.attr_bool, n=len(meetings))
    toolbox.register(«population», tools.initRepeat, list, toolbox.individual)

    def break_time(individual):
        total_break_time = 0
        last_end_time = 0
        meeting_list = [meetings[i] for i, x in enumerate(individual) if x == 1]
        meeting_list.sort(key=lambda x: x['time']
        if 'time' in x
        else 0)
        for meeting in meeting_list:
            if 'time' in meeting:
                if last_end_time > 0:
                    total_break_time += meeting['time'] - last_end_time
                    if meeting['type'] != «work»:
                        total_break_time += travel_time
                last_end_time = meeting['time'] + meeting['duration']
                if meeting['type'] != «work»:
                    last_end_time += travel_time
        return total_break_time
    def can_meetings_be_scheduled(individual):
```

```
        #individual = individual1.copy()
        if sum(individual) == 0:
            return []
        schedule = [meetings[i] for i in range(len(individual)) if individual[i] ==
1]
        schedule.sort(key=lambda x: x['time'])

        for i in range(len(schedule) - 1):
            cur_meeting = schedule[i]
            next_meeting = schedule[i + 1]
            if cur_meeting['date'] == next_meeting['date'] and cur_meeting['time']
+ cur_meeting['duration'] > next_meeting['time']:
                if cur_meeting['type'] == next_meeting['type'] == 'work':
                    if cur_meeting.get('mandatory', True):
                        individual[meetings.index(next_meeting)] = 0
                    elif next_meeting.get('mandatory', True):
                        individual[meetings.index(cur_meeting)] = 0
                    else:
                        return []
                elif cur_meeting['type'] != next_meeting['type']:

                    if                    priorities[cur_meeting['type']]                    >
priorities[next_meeting['type']]:
                        individual[meetings.index(next_meeting)] = 0
                    else:
                        individ

    ual[meetings.index(cur_meeting)] = 0
    )] = 0
        return individual
```

```
toolbox.register(«evaluate», eval_schedule)
toolbox.register(«mate», tools.cxTwoPoint)
toolbox.register(«mutate», tools.mutFlipBit, indpb=0.2)
toolbox.register(«select», tools.selTournament, tournsize=4)


def main():
    pop = toolbox.population(n=10000 )
    hof = tools.HallOfFame(1)
    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register(«avg», numpy.mean)
    stats.register(«min», numpy.min)
    stats.register(«max», numpy.max)

    if len(pop) == 0:
        print(«Популяція порожня. Перевірте функцію генерації популяції.»)
        return None
    fitnesses = list(map(toolbox.evaluate, pop))
    for ind, fit in zip(pop, fitnesses):
        ind.fitness.values = fit

    pop, log = algorithms.eaSimple(pop, toolbox, cxpb=0.7, mutpb=0.3, ngen=100, stats=stats, halloffame=hof, verbose=True)

    if len(hof) == 0:
        print(«Hall of Fame порожній після виконання генетичного алгоритму. Перевірте параметри алгоритму та/або функцію оцінювання.»)
        return None
```

```
    best_individual = hof[0]
    best_fitness = eval_schedule(best_individual)[0]
    if best_individual is not None and best_fitness > 0.5:
        display_schedule(best_individual)
print_meeting_schedule(best_individual)
    else:
        print(«No optimal schedule found»)
    gen, avg, min_, max_ = log.select(«gen», «avg», «min», «max»)
    plt.figure(figsize=(10, 7))
    plt.plot(gen, avg, label=«average»)
    plt.plot(gen, min_, label=«minimum»)
    plt.plot(gen, max_, label=«maximum»)
    plt.xlabel(«Generation»)
    plt.ylabel(«Fitness»)
    plt.legend(loc=«lower right»)
    plt.show()

    return best_individual

best_individual = main()
```

Лістинг 3 – Лістинг коду генетичного алгоритму

ДОДАТОК Б

Відомість кваліфікаційної роботи

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	77 с.
	Інші документи	
2.	Презентаційні матеріали	20 плакатів

.	.	Прізвище та ініціали.	Підп.	Дата			
Розробив		Бурцева А.Д.			Дослідження еволюційних методів для персоналізованого календарного планування на основі аналізу повідомлень про зустрічі в месенджері	Шифр групи	Код напр./спец.
Перевірив		Чалий С.Ф.				СШІм-22-3	122
Н.контр.		Малєєва І.А.				ХНУРЕ кафедра ШІ	
Затв.		Філатов В.О.					