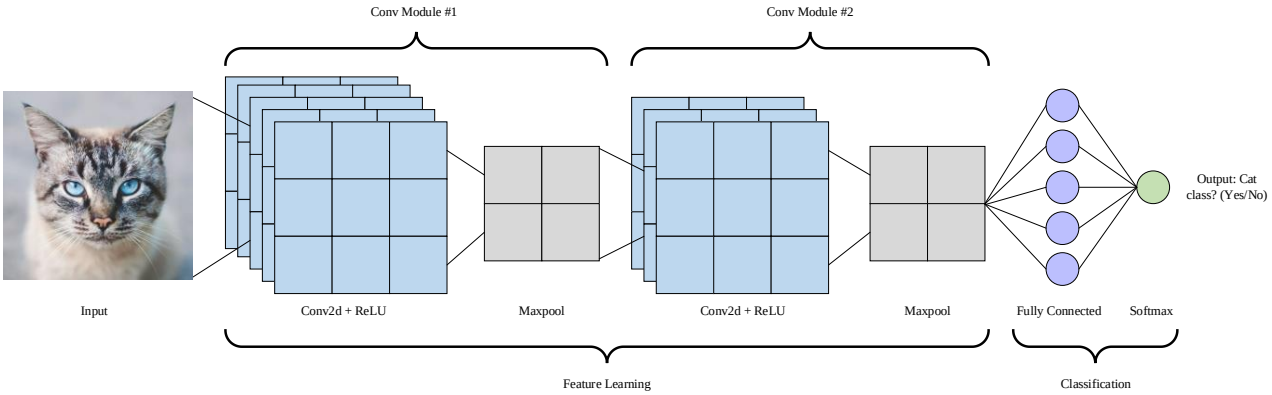
ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

ГЮІК. 50 2840.006

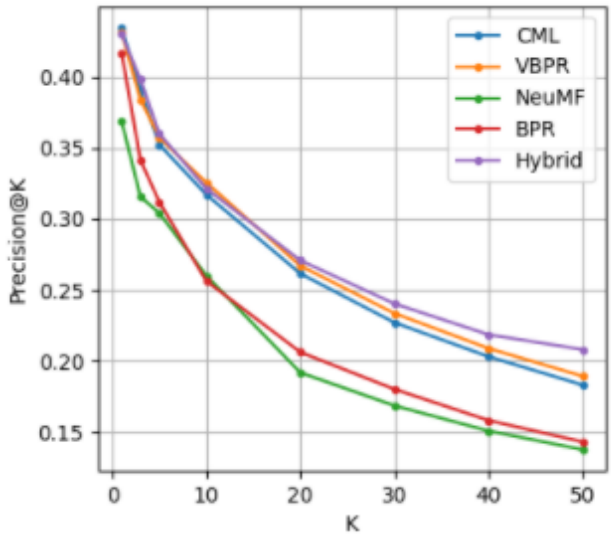
(позначення документу)

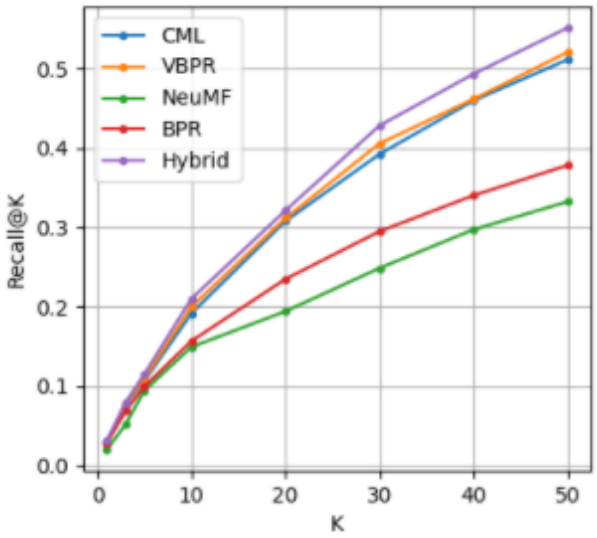
ЗАГАЛЬНА СХЕМА ЗГОРТКОВОЇ НЕЙРОННОЇ МЕРЕЖІ



Розробила	*Міронова К. В.*			*Дослідження та використання методів розпізнавання зображень для рекомендаційних систем*	
Перевірила	*Колесник Л.В.*				
Н. контр.	*Колесник Л.В.*			*ІТПм-20-1*	*Аркуш 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Аркушів 1*

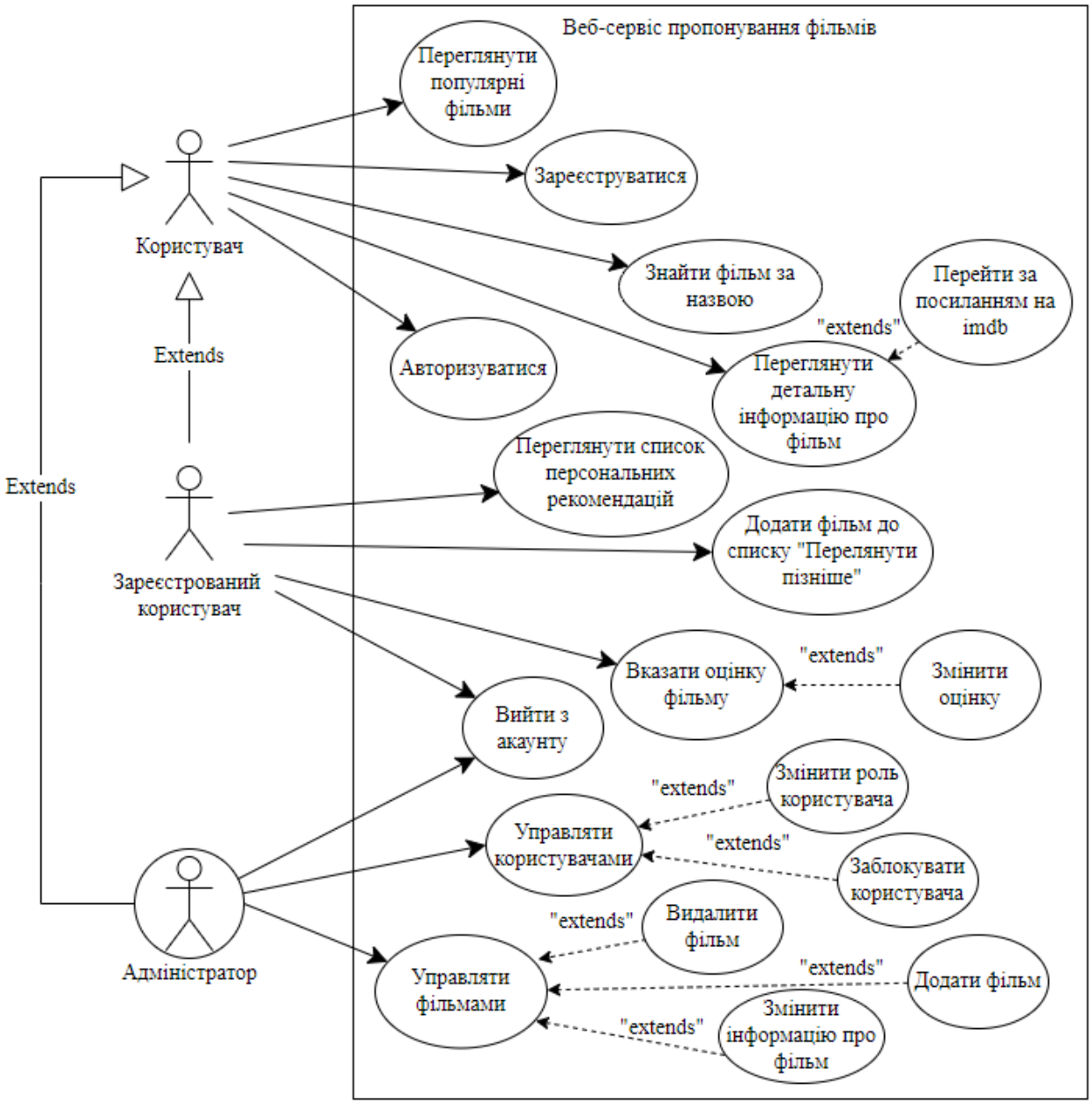
РЕЗУЛЬТАТИ ДОСЛІДЖЕНЬ





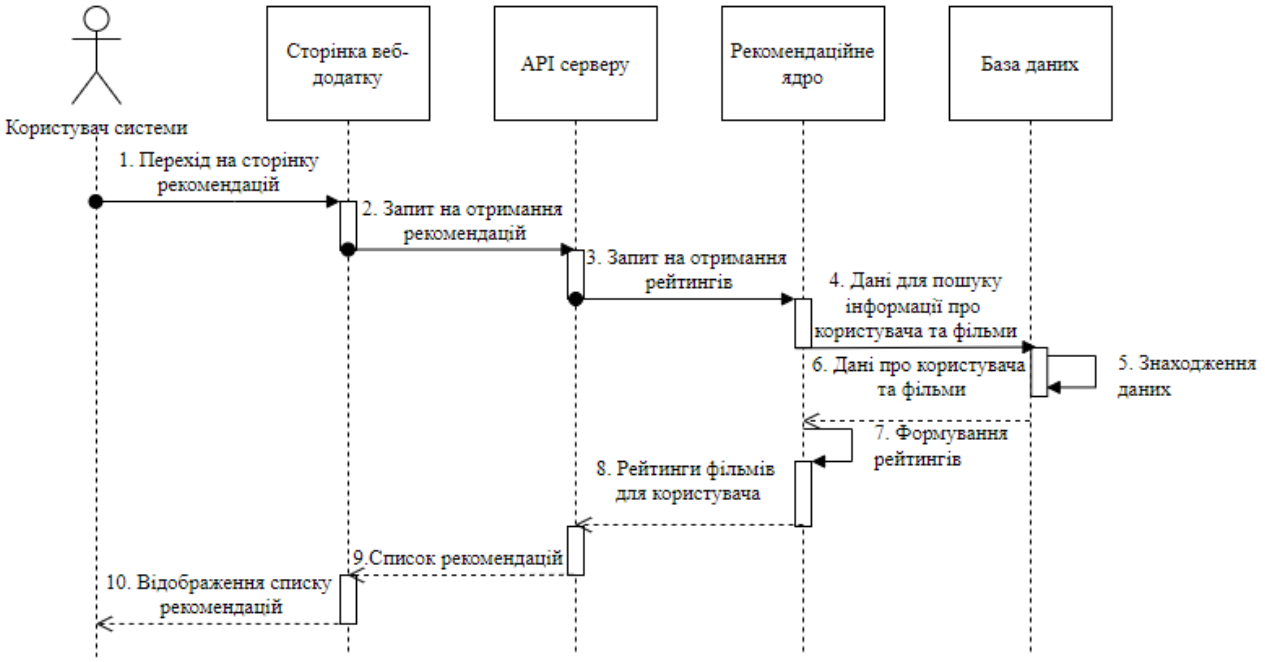
Розробила	*Міронова К. В.*			*Дослідження та використання методів розпізнавання зображень для рекомендаційних систем*	
Перевірила	*Колесник Л.В.*				
Н. контр.	*Колесник Л.В.*			*ІТПм-20-1*	*Аркуш 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Аркушів 1*

ДІАГРАМА ПРЕЦЕДЕНТІВ



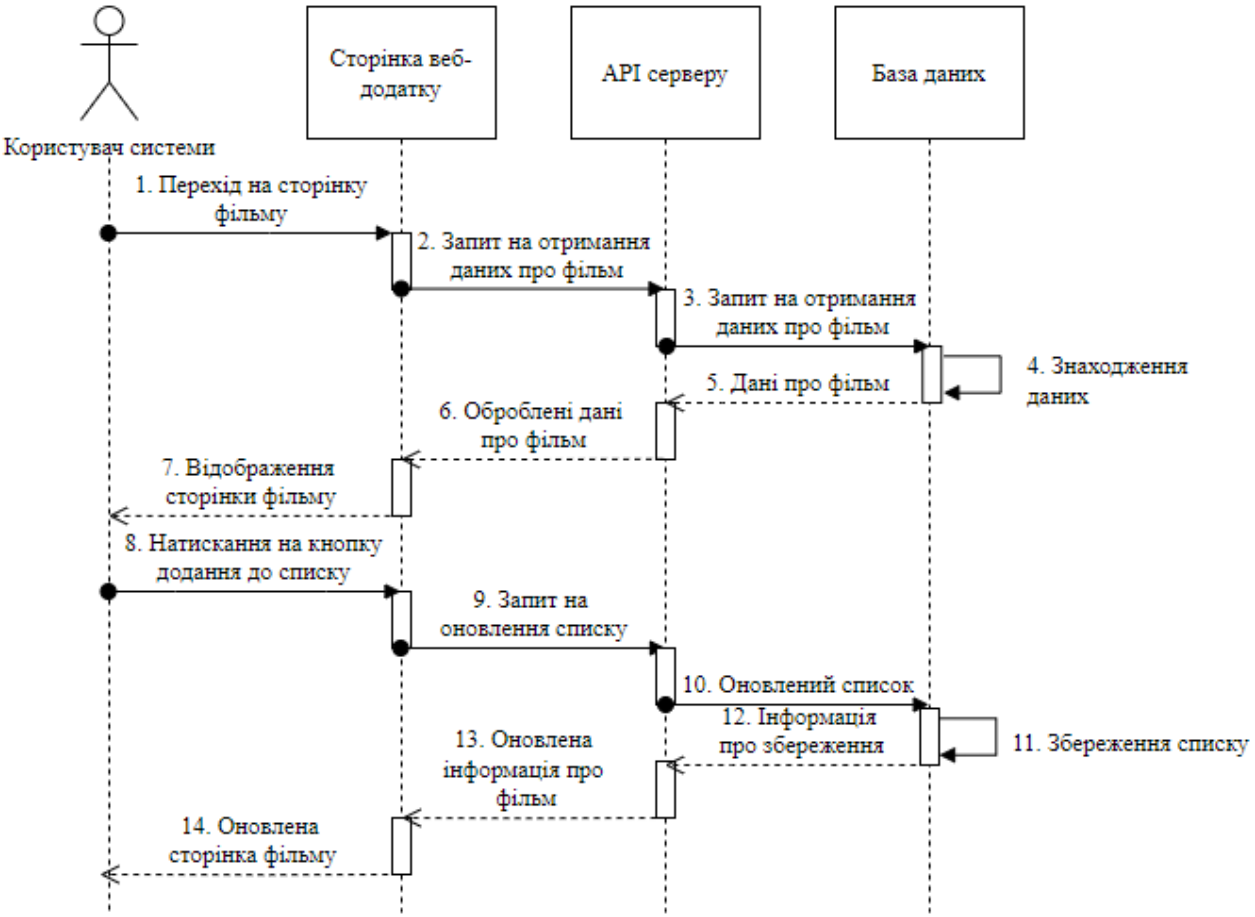
Розробила	*Міронова К. В.*			*Дослідження та використання методів розпізнавання зображень для рекомендаційних систем*	
Перевірила	*Колесник Л.В.*				
Н. контр.	*Колесник Л.В.*			*ІТПм-20-1*	*Аркуш 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Аркушів 1*

ДІАГРАМА ПОСЛІДОВНОСТЕЙ ОТРИМАННЯ РЕКОМЕНДАЦІЙ



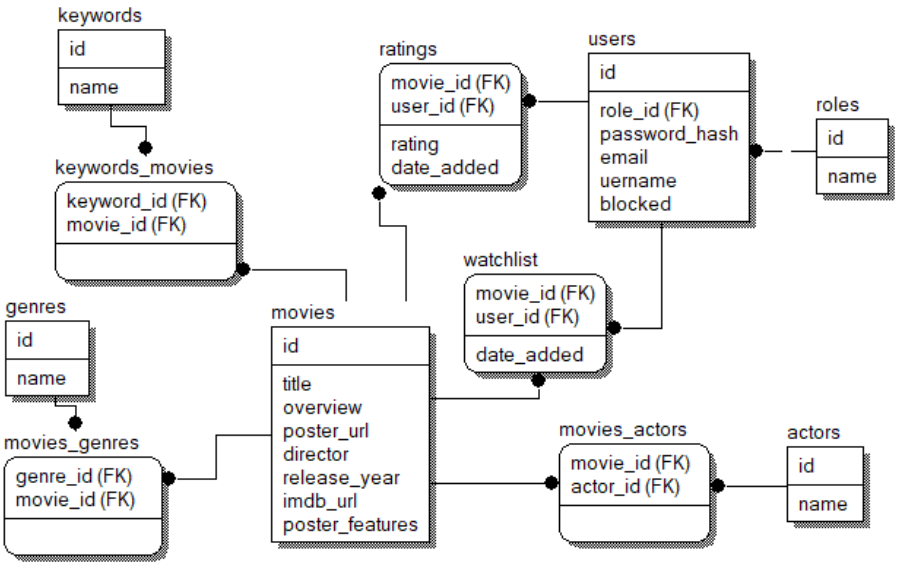
Розробила	*Міронова К. В.*			*Дослідження та використання методів розпізнавання зображень для рекомендаційних систем*	
Перевірила	*Колесник Л.В.*				
Н. контр.	*Колесник Л.В.*			*ІТПм-20-1*	*Аркуш 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Аркушів 1*

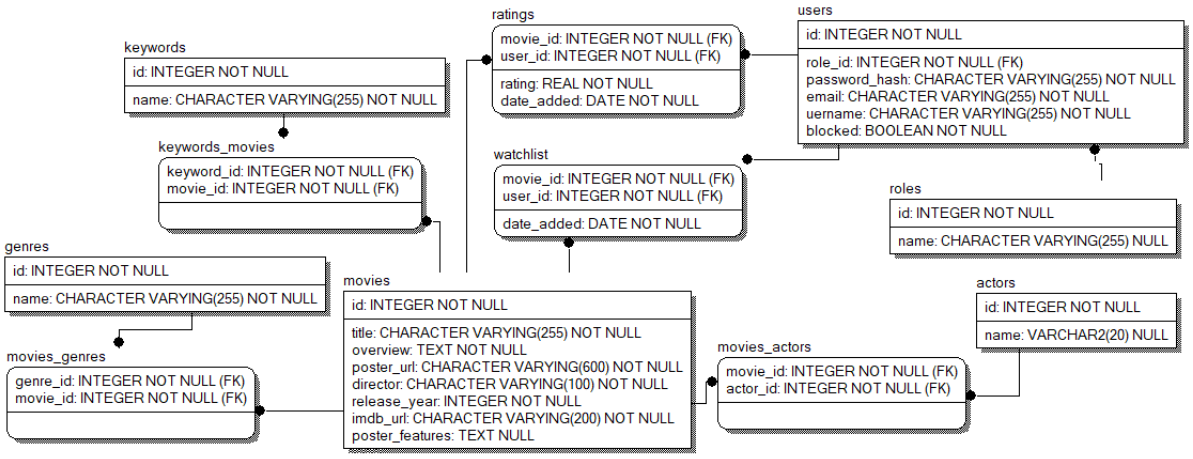
ДІАГРАМА ПОСЛІДОВНОСТЕЙ ДОДАВАННЯ ФІЛЬМУ В «ПЕРЕГЛЯНУТИ ПІЗНІШЕ»



Розробила	*Міронова К. В.*			*Дослідження та використання методів розпізнавання зображень для рекомендаційних систем*	
Перевірила	*Колесник Л.В.*				
Н. контр.	*Колесник Л.В.*			*ІТПм-20-1*	*Аркуш 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Аркушів 1*

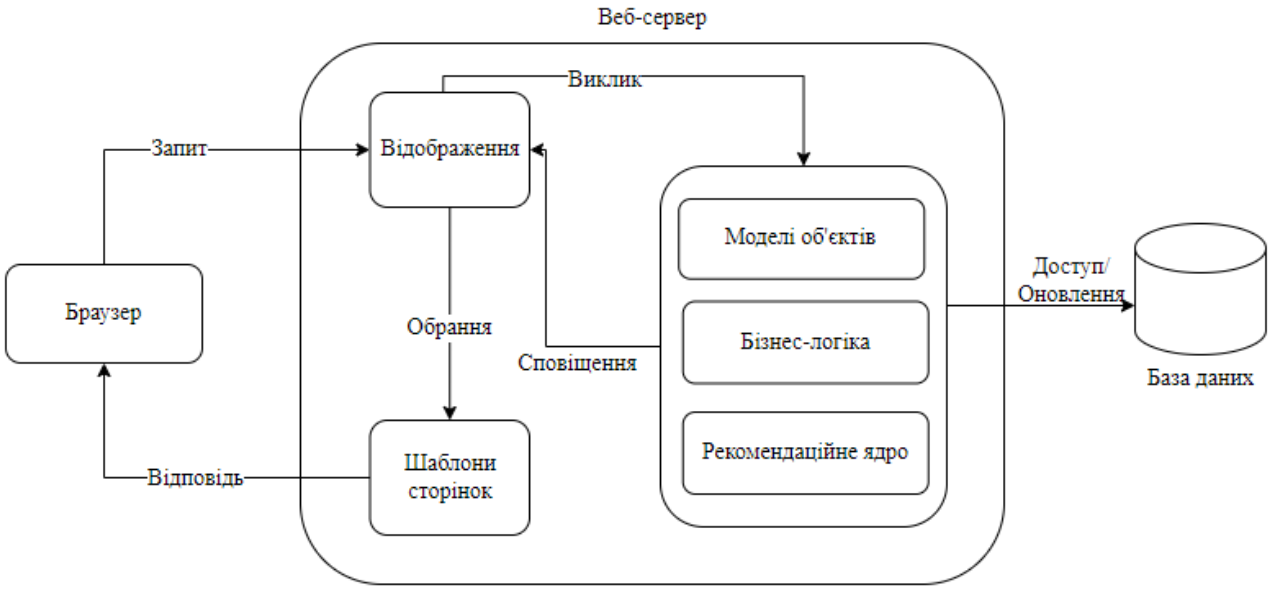
ПРОЕКТУВАННЯ БАЗИ ДАНИХ





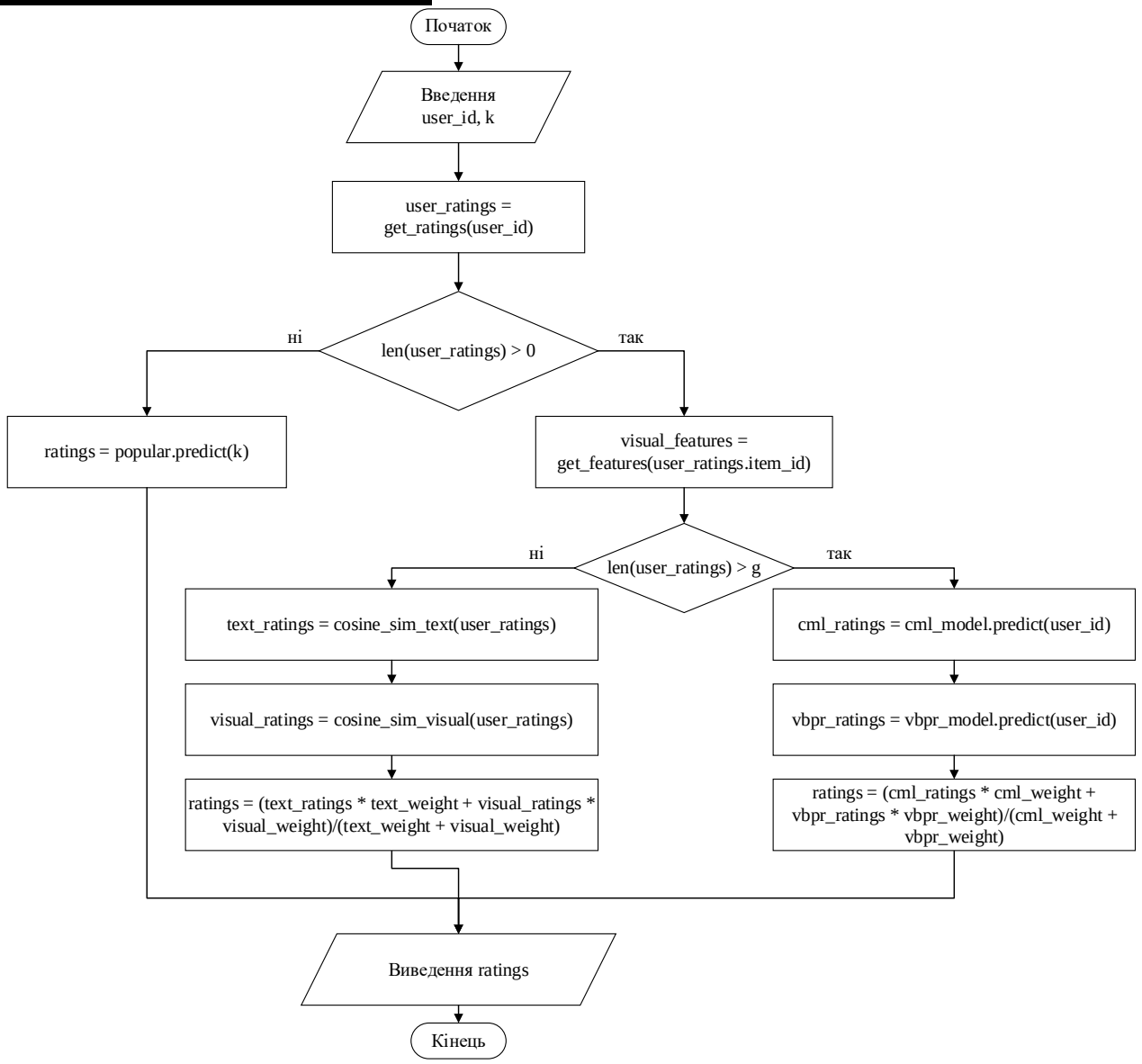
Розробила	*Міронова К. В.*			*Дослідження та використання методів розпізнавання зображень для рекомендаційних систем*	
Перевірила	*Колесник Л.В.*				
Н. контр.	*Колесник Л.В.*			*ІТПм-20-1*	*Аркуш 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Аркушів 1*

АРХІТЕКТУРА ДОДАТКУ



Розробила	*Міронова К. В.*			*Дослідження та використання методів розпізнавання зображень для рекомендаційних систем*	
Перевірила	*Колесник Л.В.*				
Н. контр.	*Колесник Л.В.*			*ІТПм-20-1*	*Аркуш 1*
Затвердив	*Гребеннік І.В.*			*СТ*	*Аркушів 1*

ГЮІК. 50 2840.006



					ГЮІК. 50 2840.006 С10
Зм.	*Лист*	*№ докум*	*Підпис*	*Дата*	
Розробила		*Міронова К.В.*			*Алгоритм запропонованої рекомендаційної системи*
Перевірила		*Колесник Л.В.*			
Т. контр.					
Реценз.					
Н. контр.		*Колесник Л.В.*			
Затвердив		*Гребеннік І.В.*			

Літ.	*Маса*	*Маштаб*
Аркуш 1	*Аркушів 1*	
ХНУРЕ Кафедра СТ		

ДОДАТОК Б

Текст програми

ГЮІК. 50 2840.006 – 01 12 01

(позначення документу)

Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

«ЗАТВЕРДЖУЮ»
Керівник кваліфікаційної роботи
_____ проф. Колесник Л. В.
(підпис)
«_____»_____2021 р.

ДОСЛІДЖЕННЯ ТА ВИКОРИСТАННЯ МЕТОДІВ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ ДЛЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

Текст програми
ЛИСТ ЗАТВЕРДЖЕННЯ
ГЮІК.50 2840.006– 01 12 01

Виконавець:
студент групи ІТПм-20-1
_____ Міронова К. В.
(підпис)
«____»_____2021 р.

2021

```
class CML(object):
    def __init__(self, sess, num_user, num_item, features=None,
learning_rate=0.1, reg_rate=0.1, epoch=500,
                 batch_size=500, verbose=True, t=5,
display_step=100, feature_l2_reg=0.1,
                 feature_projection_scaling_factor=0.5,
hidden_layer_dim=128):
        self.sess = sess
        self.num_user = num_user
        self.num_item = num_item
        self.learning_rate = learning_rate
        self.reg_rate = reg_rate
        self.hidden_layer_dim = hidden_layer_dim
        self.epochs = epoch
        self.batch_size = batch_size
        self.verbose = verbose
        self.T = t
        self.display_step = display_step
        self.feature_l2_reg = feature_l2_reg
        self.feature_projection_scaling_factor =
feature_projection_scaling_factor

        self.user_id = None
        self.item_id = None
        self.neg_item_id = None
        self.keep_rate = None
        self.pred_distance = None
        self.pred_distance_neg = None
        self.embedding_loss = None
        self.feature_loss = None
        self.loss = None
        self.optimizer = None
        self._P = None
        self._Q = None
        self.clip_P = None
        self.clip_Q = None

        self.user = None
        self.item = None
        self.num_training = None
        self.test_data = None
        self.total_batch = None
        self.neg_items = None
        self.test_users = None

        self.features = tf.constant(features, dtype=tf.float32)
        self.feature_projection = None

        print("You are running CML.")

def build_network(self, num_factor=100, margin=0.5,
norm_clip_value=1):
```

```
#tf.train.Saver().restore(self.sess, 'cml.ckpt')

self.user_id = tf.placeholder(dtype=tf.int32, shape=[None],
name='user_id')
        self.item_id = tf.placeholder(dtype=tf.int32, shape=[None],
name='item_id')
        self.neg_item_id = tf.placeholder(dtype=tf.int32,
shape=[None], name='neg_item_id')
        self.keep_rate = tf.placeholder(tf.float32)

        self._P = tf.Variable(
            tf.random_normal([self.num_user, num_factor], stddev=1 /
(num_factor ** 0.5)), dtype=tf.float32)
        self._Q = tf.Variable(
            tf.random_normal([self.num_item, num_factor], stddev=1 /
(num_factor ** 0.5)), dtype=tf.float32)

        user_embedding = tf.nn.embedding_lookup(self._P,
self.user_id)
        item_embedding = tf.nn.embedding_lookup(self._Q,
self.item_id)
        neg_item_embedding = tf.nn.embedding_lookup(self._Q,
self.neg_item_id)

        self.pred_distance = tf.reduce_sum(
            tf.nn.dropout(tf.squared_difference(user_embedding,
item_embedding), self.keep_rate), 1)
        self.pred_distance_neg = tf.reduce_sum(
            tf.nn.dropout(tf.squared_difference(user_embedding,
neg_item_embedding), self.keep_rate), 1)

        mlp_layer_1 = tf.layers.dense(inputs=self.features,
units=self.hidden_layer_dim, activation=tf.nn.relu)

        mlp_layer_2 =
tf.layers.dense(inputs=tf.layers.dropout(inputs=mlp_layer_1,
rate=self.keep_rate),
                                        units=num_factor)

        self.feature_projection = tf.clip_by_norm(mlp_layer_2 *
self.feature_projection_scaling_factor,
                                                  norm_clip_value,
axes=[1])
self.embedding_loss = tf.reduce_sum(tf.maximum(self.pred_distance -
self.pred_distance_neg + margin, 0))
self.feature_loss =
tf.reduce_sum(tf.reduce_sum(tf.squared_difference(
            self._Q, self.feature_projection), 1)) *
self.feature_l2_reg

        self.loss = self.embedding_loss + self.feature_loss
```

```
        self.clip_P = tf.assign(self._P, tf.clip_by_norm(self._P,
norm_clip_value, axes=[1]))
        self.clip_Q = tf.assign(self._Q, tf.clip_by_norm(self._Q,
norm_clip_value, axes=[1]))

        gds =
[tf.train.AdagradOptimizer(self.learning_rate).minimize(self.loss,
var_list=[self._P, self._Q]),

tf.train.AdagradOptimizer(self.learning_rate).minimize(self.feature_
loss / self.num_item)]
        self.optimizer = gds + [self.clip_P, self.clip_Q]

        return self

def prepare_data(self, train_data, test_data):
        t = train_data.tocoo()
        self.user = t.row.reshape(-1)
        self.item = t.col.reshape(-1)
        self.num_training = len(self.item)
        self.test_data = test_data
        self.total_batch = int(self.num_training / self.batch_size)
        self.neg_items = self._get_neg_items(train_data.tocsr())
        self.test_users = set([u for u in self.test_data.keys() if
len(self.test_data[u]) > 0])
        print("model: CML; data preparation finished.")

    def train(self):
        self.sess.run(tf.assign(self._Q, self.feature_projection))
        idxs = np.random.permutation(self.num_training)  # shuffled
ordering
        user_random = list(self.user[idxs])
        item_random = list(self.item[idxs])
        item_random_neg = []
        for u in user_random:
            neg_i = self.neg_items[u]
            s = np.random.randint(len(neg_i))
            item_random_neg.append(neg_i[s])

        for i in range(self.total_batch):
            start_time = time.time()
            batch_user = user_random[i * self.batch_size:(i + 1) *
self.batch_size]
            batch_item = item_random[i * self.batch_size:(i + 1) *
self.batch_size]
            batch_item_neg = item_random_neg[i * self.batch_size:(i
+ 1) * self.batch_size]

            _, loss, _, _ = self.sess.run((self.optimizer,
self.loss, self.clip_P, self.clip_Q),
                                          feed_dict={self.user_id:
batch_user,
```

```
                                                        self.item_id:
batch_item,

self.neg_item_id: batch_item_neg,
                                                        self.keep_rate:
0.98})

if i % self.display_step == 0:
            if self.verbose:
                print("model: CML; Index: %04d; cost= %.9f" % (i +
1, loss))
                print("model: CML; one iteration: %s seconds." %
(time.time() - start_time))

def test(self):
    return evaluate(self)

def execute(self, train_data, test_data):

    self.prepare_data(train_data, test_data)

    init = tf.global_variables_initializer()
    self.sess.run(init)

    for epoch in range(self.epochs):
        self.train()
        if epoch % self.T == 0:
            print("Epoch: %04d; " % epoch, end='')
            self.test()
    self.save('/checkpoints/cml.ckpt')

def save(self, path):
    saver = tf.train.Saver()
    saver.save(self.sess, path)

def predict(self, user_id, item_id):
    return -self.sess.run([self.pred_distance],
                          feed_dict={self.user_id: user_id,
self.item_id: item_id, self.keep_rate: 1})[0]

def _get_neg_items(self, data):
    all_items = set(np.arange(self.num_item))
    neg_items = {}
    for u in range(self.num_user):
        neg_items[u] = list(all_items -
set(data.getrow(u).nonzero()[1]))

    return neg_items
```

```
class VBPR(object):
    def __init__(self, sess, num_user, num_item, features,
learning_rate=0.001, epoch=150,
                 batch_size=500, verbose=True, num_factor=30,
num_feature_factor=4096, t=5, display_step=1000,
                 le=0, lu=2.5e-3, li=2.5e-3, li_neg=2.5e-4, lb=0):
        self.sess = sess
        self.num_user = num_user
        self.num_item = num_item
        self.num_factor = num_factor
        self.num_feature_factor = num_feature_factor
        self.learning_rate = learning_rate
        self.epochs = epoch
        self.batch_size = batch_size
        self.verbose = verbose
        self.T = t
        self.display_step = display_step

        self.user_id = None
        self.item_id = None
        self.neg_item_id = None
        self.item_content = None
        self.neg_item_content = None
        self.user_rating_embed = None
        self.user_content_embed = None
        self.item_rating_embed = None
        self.item_rating_bias = None
        self.item_content_bias = None
        self.content_embed_matrix = None
        self.pred_y = None
        self.pred_y_neg = None
        self.pred_y_all = None
        self.loss = None
        self.optimizer = None

        self.le = le
        self.lu = lu
        self.li = li
        self.li_neg = li_neg
        self.lb = lb

        self.test_data = None
        self.user = None
        self.item = None
        self.neg_items = None
        self.test_users = None
        self.features = features

        self.num_training = None
        self.total_batch = None
        print("You are running VBPR.")
```

```
    def build_network(self):

        self.user_id = tf.placeholder(dtype=tf.int32, shape=[None],
name='user_id')
        self.item_id = tf.placeholder(dtype=tf.int32, shape=[None],
name='item_id')
        self.neg_item_id = tf.placeholder(dtype=tf.int32,
shape=[None], name='neg_item_id')
        self.item_content = tf.placeholder(dtype=tf.float32,
shape=[None, self.num_feature_factor],
                                           name='item_content')
        self.neg_item_content = tf.placeholder(dtype=tf.float32,
shape=[None, self.num_feature_factor],

name='neg_item_content')

        self.user_rating_embed =
tf.Variable(tf.random_normal([self.num_user, self.num_factor],
stddev=0.01))
        self.user_content_embed =
tf.Variable(tf.random_normal([self.num_user, self.num_factor],
stddev=0.01))
        self.item_rating_embed =
tf.Variable(tf.random_normal([self.num_item, self.num_factor],
stddev=0.01))

        self.item_rating_bias = tf.Variable(tf.constant(0.0,
shape=[self.num_item, 1], dtype=tf.float32))
        self.item_content_bias = tf.Variable(tf.constant(0.0,
shape=[self.num_feature_factor, 1], dtype=tf.float32))
        self.content_embed_matrix = tf.constant(1 /
(self.num_feature_factor * self.num_factor),

shape=[self.num_feature_factor, self.num_factor], dtype=tf.float32)

        user_rating_embedding =
tf.nn.embedding_lookup(self.user_rating_embed, self.user_id)
        user_content_embedding =
tf.nn.embedding_lookup(self.user_content_embed, self.user_id)
        item_rating_embedding =
tf.nn.embedding_lookup(self.item_rating_embed, self.item_id)
        neg_item_rating_embedding =
tf.nn.embedding_lookup(self.item_rating_embed, self.neg_item_id)
        item_rating_bias_f =
tf.nn.embedding_lookup(self.item_rating_bias, self.item_id)
        neg_item_rating_bias_f =
tf.nn.embedding_lookup(self.item_rating_bias, self.neg_item_id)
        item_content_embedding = tf.matmul(self.item_content,
self.content_embed_matrix)
        neg_item_content_embedding =
tf.matmul(self.neg_item_content, self.content_embed_matrix)
```

```
        self.pred_y =
tf.reduce_sum(tf.multiply(user_rating_embedding,
item_rating_embedding) +

tf.multiply(user_content_embedding, item_content_embedding), 1)
        self.pred_y_neg =
tf.reduce_sum(tf.multiply(user_rating_embedding,
neg_item_rating_embedding) +

tf.multiply(user_content_embedding, neg_item_content_embedding), 1)

        self.pred_y_all = item_rating_bias_f -
neg_item_rating_bias_f + self.pred_y - self.pred_y_neg + \
                          tf.matmul(self.item_content -
self.neg_item_content, self.item_content_bias)

        self.loss = tf.reduce_sum(tf.log(1 + tf.exp(-
self.pred_y_all))) + \
                    0.5 * tf.reduce_sum(self.content_embed_matrix **
2) * self.le + \
                    0.5 * tf.reduce_sum(user_rating_embedding ** 2 +
(user_content_embedding ** 2) * self.lu
                                        + item_rating_embedding ** 2
* self.li
                                        + neg_item_rating_embedding
** 2 * self.li_neg) + \
                    0.5 * (tf.reduce_sum(item_rating_bias_f ** 2 +
neg_item_rating_bias_f ** 2)
                           + tf.reduce_sum(self.item_content_bias **
2)) * self.lb

        self.optimizer =
tf.train.RMSPropOptimizer(learning_rate=self.learning_rate).minimize
(self.loss)

    def prepare_data(self, train_data, test_data):
        t = train_data.tocoo()
        self.user = t.row.reshape(-1)
        self.item = t.col.reshape(-1)
        self.num_training = len(self.item)
        self.test_data = test_data
        self.total_batch = int(self.num_training / self.batch_size)
        self.neg_items = self._get_neg_items(train_data.tocsr())
        self.test_users = set([u for u in self.test_data.keys() if
len(self.test_data[u]) > 0])
        print("model: VBPR; data preparation finished.")

    def train(self):
        idxs = np.random.permutation(self.num_training)  # shuffled
ordering
        user_random = list(self.user[idxs])
        item_random = list(self.item[idxs])
```

```
        item_random_neg = []
        for u in user_random:
            neg_i = self.neg_items[u]
            s = np.random.randint(len(neg_i))
            item_random_neg.append(neg_i[s])

        for i in range(self.total_batch):
            start_time = time.time()
            batch_user = user_random[i * self.batch_size:(i + 1) *
self.batch_size]
            batch_item = item_random[i * self.batch_size:(i + 1) *
self.batch_size]
            batch_item_neg = item_random_neg[i * self.batch_size:(i
+ 1) * self.batch_size]
            batch_item_content = self.get_feature(batch_item)
            batch_item_content_neg =
self.get_feature(batch_item_neg)

            _, loss = self.sess.run((self.optimizer, self.loss),
feed_dict={self.user_id: batch_user,

self.item_id: batch_item,

self.neg_item_id: batch_item_neg,

self.item_content: batch_item_content,

self.neg_item_content: batch_item_content_neg})

            if i % self.display_step == 0:
                if self.verbose:
                    print("model: VBPR; Index: %04d; cost= %.9f" %
(i + 1, loss))
                    print("model: VBPR; one iteration: %s seconds."
% (time.time() - start_time))

    def get_feature(self, batch_item):
        df = pd.DataFrame({'item_id': [x + 1 for x in batch_item]})
        feature =
df['item_id'].map(self.features.set_index('id')['feature'])
        return feature.str.split(pat=",",
expand=True).to_numpy(dtype=np.float)

    def test(self):
        evaluate(self)

    def execute(self, train_data, test_data):
        self.prepare_data(train_data, test_data)

        init = tf.global_variables_initializer()
        self.sess.run(init)
```

```
        for epoch in range(self.epochs):
            self.train()
            if epoch % self.T == 0:
                print("Epoch: %04d; " % epoch, end='')
                self.test()

    def save(self, path):
        saver = tf.train.Saver()
        saver.save(self.sess, path)

    def predict(self, user_id, item_id, item_content):
        return self.sess.run([self.pred_y], feed_dict={self.user_id:
user_id,
                                                       self.item_id:
item_id,

self.item_content: item_content})[0]

    def _get_neg_items(self, data):
        all_items = set(np.arange(self.num_item))
        neg_items = {}
        for u in range(self.num_user):
            neg_items[u] = list(all_items -
set(data.getrow(u).nonzero()[1]))

        return neg_items

class Hybrid(object):
    def __init__(self, sess, cml_weight=0.5, vbpr_weight=0.5,
epochs=100, t=5, batch_size=1024):
        self.sess = sess
        self.cml_model = None
        self.vbpr_model = None
        self.cosine_similarity = None
        self.popular = None

        self.cml_weight = cml_weight / (cml_weight + vbpr_weight)
        self.vbpr_weight = vbpr_weight / (cml_weight + vbpr_weight)

        self.n_user = None
        self.n_item = None
        self.train_data = None
        self.test_data = None

        self.T = t
        self.epochs = epochs
        self.batch_size = batch_size

        self.scores = None
        self.pred_ratings_5 = None
        self.pred_ratings_10 = None
```

```
        self.user = None
        self.item = None
        self.num_training = None
        self.test_data = None
        self.total_batch = None
        self.neg_items = None
        self.test_users = None

        self.items_info = load_item_info()
        self.ratings_df = load_ratings_df()
        self.features =
load_features_df(path="../../data/ml100k/features.csv")

        print("You are running hybrid recommender.")

    def execute(self):
        self.train_data, self.test_data, self.n_user, self.n_item =
load_data_neg(
            path="../../data/ml100k/movielens_100k.dat",
test_size=0.2, sep="\t")


        self.cml_model = CML(self.sess, self.n_user, self.n_item,

features=load_data_features(path="../../data/ml100k/features.csv"))
        self.vbpr_model = VBPR(self.sess, self.n_user, self.n_item,
self.features)

        self.cml_model = tf.train.Saver().restore(self.sess,
'cml.ckpt')
        self.vbpr_model = tf.train.Saver().restore(self.sess,
'vbpr.ckpt')
        self.popular = Popular(self.sess)
        self.cosine_similarity = CosineSimilarity(self.items_info,
self.features)

        self.cml_model.build_network()
        self.vbpr_model.build_network()
        self.cosine_similarity.prepare_cosine()

        self.prepare_data(self.train_data, self.test_data)

        init = tf.global_variables_initializer()
        self.sess.run(init)

        for epoch in range(self.epochs):
            self.cml_model.train()
            self.vbpr_model.train()
            if epoch % self.T == 0:
                print("Epoch: %04d; " % epoch, end='')
                self.test()
```

```
    def prepare_data(self, train_data, test_data):
        t = train_data.tocoo()
        self.user = t.row.reshape(-1)
        self.item = t.col.reshape(-1)
        self.num_training = len(self.item)
        self.test_data = test_data
        self.total_batch = int(self.num_training / self.batch_size)
        self.neg_items = self._get_neg_items(train_data.tocsr())
        self.test_users = set([u for u in self.test_data.keys() if
len(self.test_data[u]) > 0])

        for model in [self.cml_model, self.vbpr_model]:
            model.user = self.user
            model.item = self.item
            model.num_training = self.num_training
            model.test_data = self.test_data
            model.total_batch = self.total_batch
            model.neg_items = self.neg_items
            model.test_users = self.test_users

    def test(self):
        return evaluate(self)

    def predict(self, user_id):
        user_ratings = self.ratings_df[self.ratings_df == user_id]
        num_ratings = len(user_ratings.item_id.unique())
        if num_ratings == 0:
            self.popular.predict()
        elif num_ratings < 5:
            self.predict_cosine(user_ratings)
        else:
            self.predict_cf(user_id)

        print('Recommendation list (5 items)')
        for idx, item in enumerate(self.pred_ratings_5):
            item_info = self.items_info[self.items_info.id ==
item[0]]
            print("%01d. item = %s (rating = %.5f)" % (idx,
item_info.title, item[1]))

        print('------------------------------')

        print('Recommendation list (10 items)')
        for idx, item in enumerate(self.pred_ratings_10):
            item_info = self.items_info[self.items_info.id ==
item[0]]
            print("%02d. item = %s (rating = %.5f)" % (idx,
item_info.title, item[1]))

        return self.scores

    def predict_popular(self, ratings_df):
```

```
        self.popular.predict(ratings_df)
        self.pred_ratings_5 = self.popular.pred_ratings_5
        self.pred_ratings_10 = self.popular.pred_ratings_10

    def predict_cosine(self, user_ratings):
        return self.cosine_similarity.predict(user_ratings)

    def predict_cf(self, user_id):
        all_items = set(np.arange(self.n_item))
        neg_items = list(all_items -
set(self.train_data.tocsr().getrow(user_id).nonzero()[1]))
        item_ids = [neg_item for neg_item in neg_items]
        user_ids = [user_id] * len(neg_items)
        item_contents = self.vbpr_model.get_feature(item_ids)
        cml_scores = self.cml_model.predict(user_ids, item_ids)
        vbpr_scores = self.vbpr_model.predict(user_ids, item_ids,
item_contents)
        self.scores = cml_scores * self.cml_weight + vbpr_scores *
self.vbpr_weight
        neg_item_index = list(zip(item_ids, self.scores))
        ranked_list = sorted(neg_item_index, key=lambda tup: tup[1],
reverse=True)
        pred_ratings = [r[0] for r in ranked_list]
        self.pred_ratings_5 = pred_ratings[:5]
        self.pred_ratings_10 = pred_ratings[:10]

    def predict(self, user_id, item_id, item_content):
        user_ratings = self.ratings_df[self.ratings_df.user_id ==
user_id[0]+1]
        num_ratings = len(user_ratings.item_id.unique())
        self.predict_cosine(user_ratings)
        if num_ratings == 0:
            self.popular.predict()
        elif num_ratings < 5:
            return self.predict_cosine(user_ratings)
        else:
            cml_scores = self.cml_model.predict(user_id, item_id)
            vbpr_scores = self.vbpr_model.predict(user_id, item_id,
item_content)

            cml_scores = [(score[0], score[1] * self.cml_weight) for
score in cml_scores]
            vbpr_scores = [(score[0], score[1] * self.vbpr_weight)
for score in vbpr_scores]

            self.scores = cml_scores * self.cml_weight + vbpr_scores
* self.vbpr_weight
        return self.scores

    def _get_neg_items(self, data):
        all_items = set(np.arange(self.n_item))
        neg_items = {}
```

```
        for u in range(self.n_user):
            neg_items[u] = list(all_items -
set(data.getrow(u).nonzero()[1]))

        return neg_items


if __name__ == '__main__':
    config = tf.ConfigProto(device_count={'GPU': 0})
    sess = tf.Session(config=config)

    recommender = Hybrid(sess)
    recommender.execute()
    recommender.predict(12)


class CosineSimilarity(object):
    def __init__(self, items_info, features, feature_weight=0.4,
text_weight=0.6):
        self.feature_weight = feature_weight / (feature_weight +
text_weight)
        self.text_weight = text_weight / (feature_weight +
text_weight)
        self.items_info = items_info
        self.features = features
        self.cosine_sim_features = None
        self.cosine_sim_text = None
        print("You are running cosine similarity.")

    def prepare_cosine(self):
        self.prepare_cosine_text()
        self.prepare_cosine_features()

    def predict(self, item_id):
        cosine_sim_text = self.cosine_sim_text
        cosine_sim_features = self.cosine_sim_features

        sim_scores_text = list(enumerate(cosine_sim_text[item_id]))
        sim_scores_img =
list(enumerate(cosine_sim_features[item_id]))

        sim_scores_text = [(score[0], score[1] * self.text_weight)
for score in sim_scores_text]
        sim_scores_img = [(score[0], score[1] * self.feature_weight)
for score in sim_scores_img]

        sim_scores = sim_scores_text + sim_scores_img

        sim_scores = [(k, sum(map(itemgetter(1), g)))
                      for k, g in groupby(sorted(sim_scores,
key=itemgetter(0)), key=itemgetter(0))]
```

```
        return sorted(sim_scores, key=lambda x: x[1], reverse=True)

    def prepare_cosine_text(self):
        items_data = self.items_info
        columns_genres = items_data[["unknown", "action",
"adventure", "animation", "children\'s", "comedy",
                                     "crime", "documentary",
"drama", "film-noir", "horror", "musical", "mystery",
                                     "romance", "sci-fi",
"thriller", "war", "western"]]

        items_data['genres'] =
columns_genres.gt(0).dot(columns_genres.columns +
',').str.rstrip(',').str.split(',')

        text_features = ['director', 'keywords', 'actors', 'genres']

        for feature in text_features:
            items_data[feature] =
items_data[feature].apply(clean_data)

        items_data['soup'] = items_data.apply(create_soup, axis=1)

        count = CountVectorizer(stop_words='english')
        count_matrix_text = count.fit_transform(items_data['soup'])
        cosine_sim_text = cosine_similarity(count_matrix_text,
count_matrix_text)

pd.DataFrame(cosine_sim_text).to_csv('../../data/ml100k/cosine_text.
csv', sep=',', index=False, header=False)
        self.cosine_sim_text = cosine_sim_text
        return self.cosine_sim_text

    def prepare_cosine_features(self):
        img_features =
np.array(list(self.features['feature'].str.split(",")), np.float32)
        cosine_sim_features = cosine_similarity(img_features)

pd.DataFrame(cosine_sim_features).to_csv('../../data/ml100k/cosine_f
eatures.csv', sep=',', index=False,
                                                header=False)
        self.cosine_sim_features = cosine_sim_features
        return self.cosine_sim_features


def create_soup(item_info):
    return ' '.join(item_info['director']) + ' ' + '
'.join(item_info['keywords']) + ' ' \
           + item_info['actors'] + ' ' + '
'.join(item_info['genres'])
```

```
def clean_data(column):
    if isinstance(column, list):
        return [str.lower(i.replace(" ", "")) for i in column]
    else:
        if isinstance(column, str):
            return str.lower(column.replace(" ", ""))
        else:
            return ''
```

ДОДАТОК В

Відомість кваліфікаційної роботи

ГЮІК.502840.006 ДЗ

(позначення документу)

	Позначення	Найменування	Додаткові відомості
		Текстові документи	
1.	ГЮІК. 502840.006 ПЗ	Пояснювальна записка	79 с.
2.	ГЮІК. 502840.006 – 01 12 01	Додаток Б. Текст програми	15 с., вкл. до ПЗ
		Графічні документи	
3.		Загальна схема згорткової нейронної	Включено до ПЗ
		мережі	
4.		Результати досліджень	Включено до ПЗ
5.		Діаграма прецедентів	Включено до ПЗ
6.		Діаграма послідовностей отримання	Включено до ПЗ
		рекомендацій	
7.		Діаграма послідовностей додавання	Включено до ПЗ
		фільму в «Переглянути пізніше»	
8.		Проектування бази даних	Включено до ПЗ
9.		Архітектура додатку	Включено до ПЗ
7.	ГЮІК. 502840.006 С10	Схема алгоритму запропонованої	Включено до ПЗ
		рекомендаційної системи	

					ГЮІК. 502840.006 ДЗ		
Зм.	*Лист*	*№ докум*	*Під-пис*	*Дата*			
Розробила		*Міронова К.В.*			Дослідження та використання методів розпізнавання зображень для рекомендаційних систем	Лист	Листів
Перевірила		*Колесник Л.В.*				*1*	*1*
Н. контр.		*Колесник Л.В.*				*ХНУРЕ Кафедра СТ*	
Затвердив		*Гребеннік І.В.*					