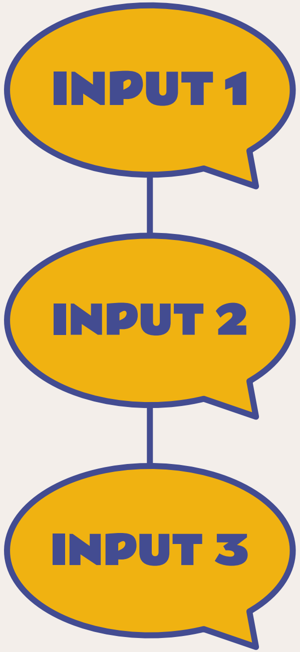
Transparent Hybrid User-adjustable Recipe Recommender System

THURSDAY

CS 4675/6675 Group 8
Shiyi W. Taichang Z. Shuangyue C.
Shuyan L. Haoran Z.

System Design

Hybrid - User Experience Flow



INPUT 1

I might want ingredients..

“Winter squash, mexican seasoning, mixed spice, honey butter”

INPUT 2

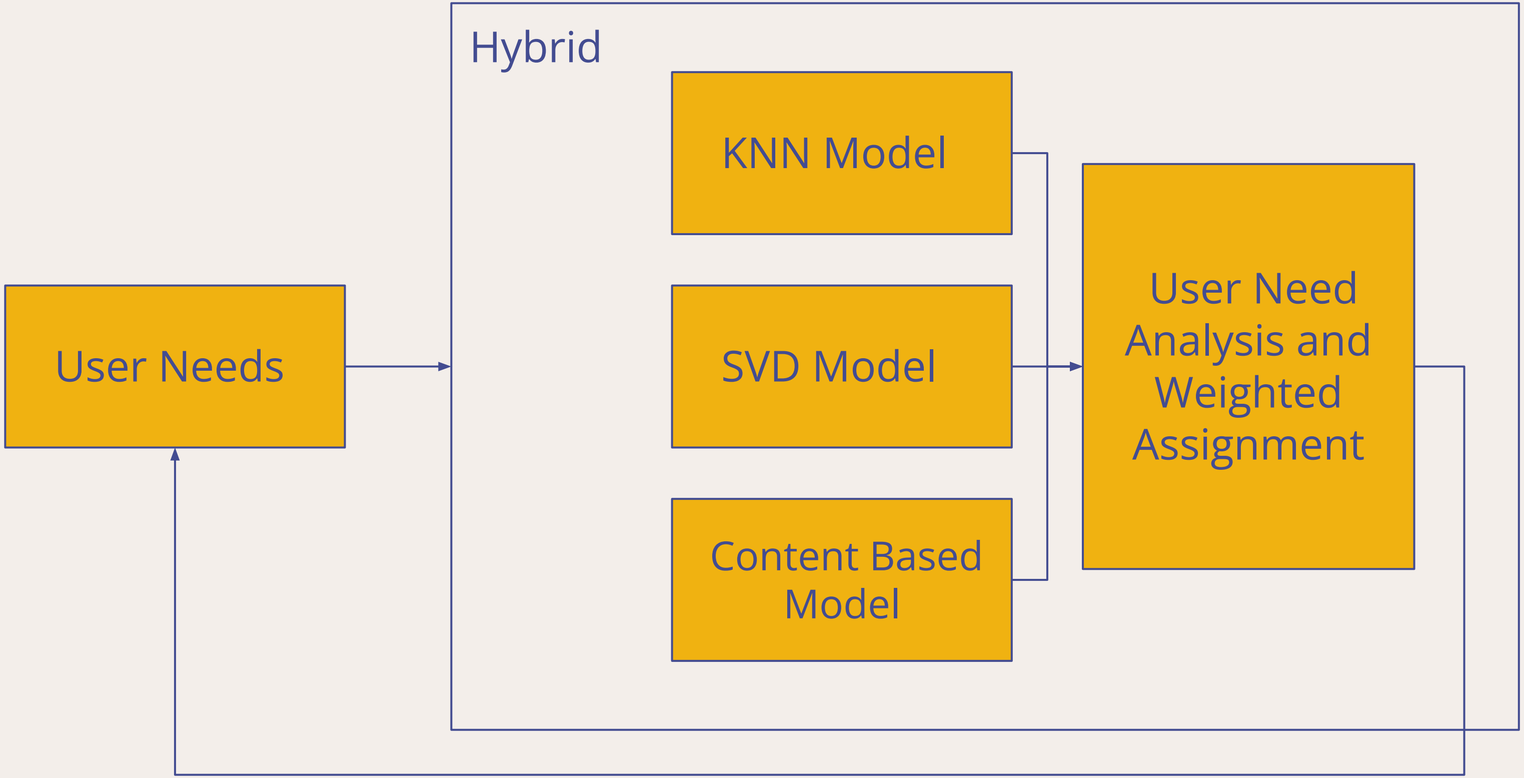
Ingredients that I absolutely don’t want..

“Onion, garlic, chicken”

INPUT 3

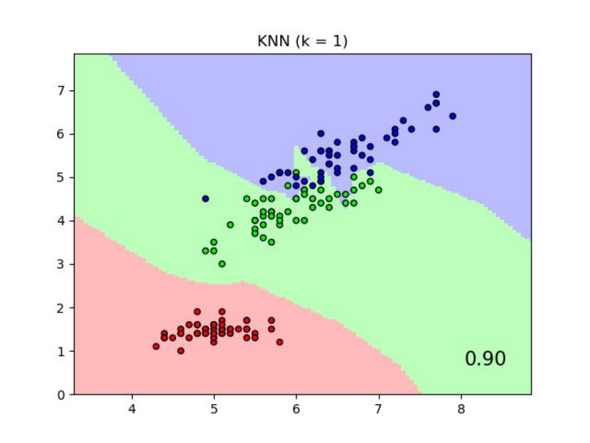
How likely I want to see other users’ recipe with a similar taste with me..

System Design Overview

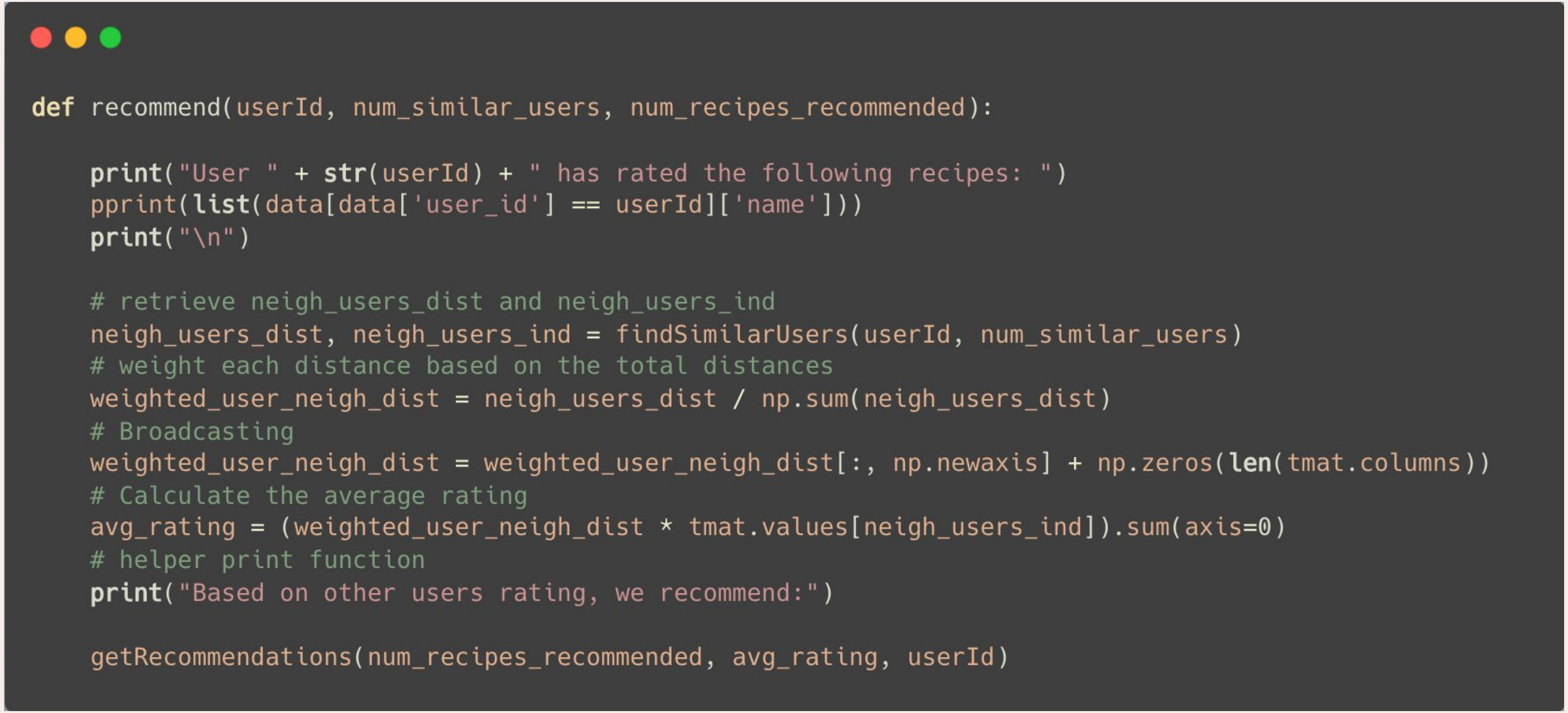


KNN

- K-Nearest Neighbors algorithm
- Collaborative Filtering
  - Birds of the same feather flock together.
- Advantages
  - Simplicity (Occam’s razor)
  - Non-parametric
- Disadvantages
  - Lazy Learning Algorithm
  - Efficiency performance drops as dataset size grows



KNN Code Walkthrough

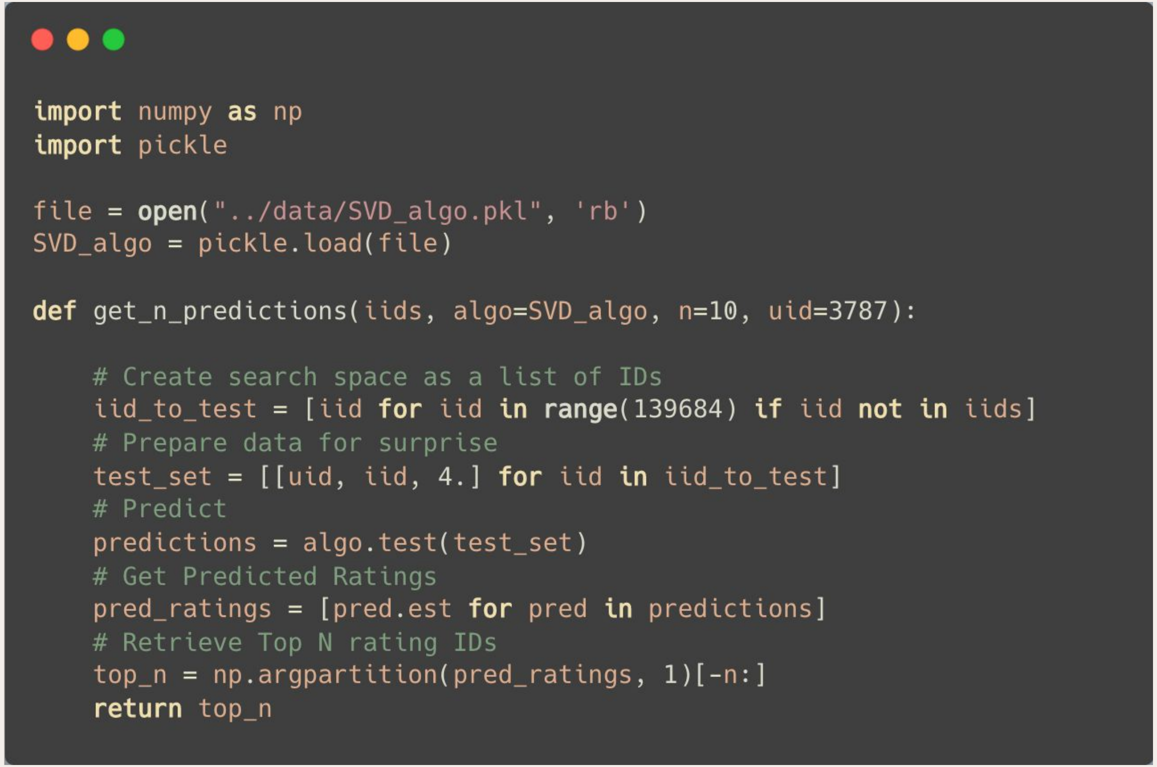


```
def recommend(userId, num_similar_users, num_recipes_recommended):

    print("User " + str(userId) + " has rated the following recipes: ")
    pprint(list(data[data['user_id'] == userId]['name']))
    print("\n")

    # retrieve neigh_users_dist and neigh_users_ind
    neigh_users_dist, neigh_users_ind = findSimilarUsers(userId, num_similar_users)
    # weight each distance based on the total distances
    weighted_user_neigh_dist = neigh_users_dist / np.sum(neigh_users_dist)
    # Broadcasting
    weighted_user_neigh_dist = weighted_user_neigh_dist[:, np.newaxis] + np.zeros(len(tmat.columns))
    # Calculate the average rating
    avg_rating = (weighted_user_neigh_dist * tmat.values[neigh_users_ind]).sum(axis=0)
    # helper print function
    print("Based on other users rating, we recommend:")

    getRecommendations(num_recipes_recommended, avg_rating, userId)
```

SVD

- Singular Value decomposition
- Collaborative Filtering
  - Matrix factorization
  - Reduce the # of features of by reducing space dimensions
- Advantages
  - Optimize model performance by minimizing RMSE error
- Disadvantages
  - Transformed data may be difficult to understand.

SVD Code Walkthrough



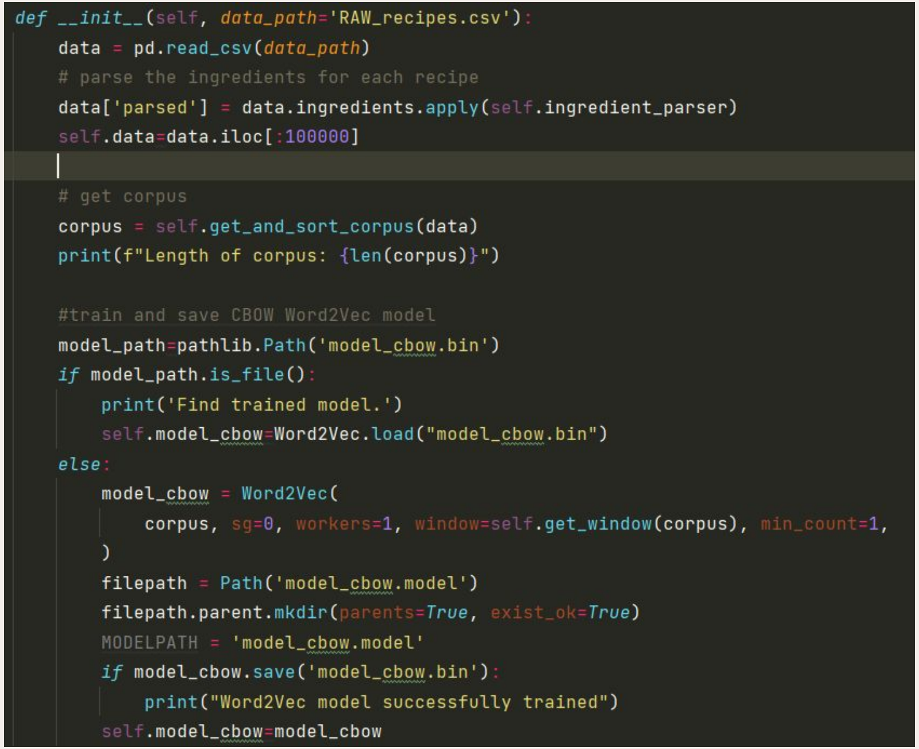
```python
import numpy as np
import pickle

file = open("../data/SVD_algo.pkl", 'rb')
SVD_algo = pickle.load(file)

def get_n_predictions(iids, algo=SVD_algo, n=10, uid=3787):

    # Create search space as a list of IDs
    iid_to_test = [iid for iid in range(139684) if iid not in iids]
    # Prepare data for surprise
    test_set = [[uid, iid, 4.] for iid in iid_to_test]
    # Predict
    predictions = algo.test(test_set)
    # Get Predicted Ratings
    pred_ratings = [pred.est for pred in predictions]
    # Retrieve Top N rating IDs
    top_n = np.argpartition(pred_ratings, 1)[-n:]
    return top_n
```

Content-Based

- Word2vec:
  - A model that converts words into vectors. Vectors with similar meanings are close to each other, and there is a well-known equation:

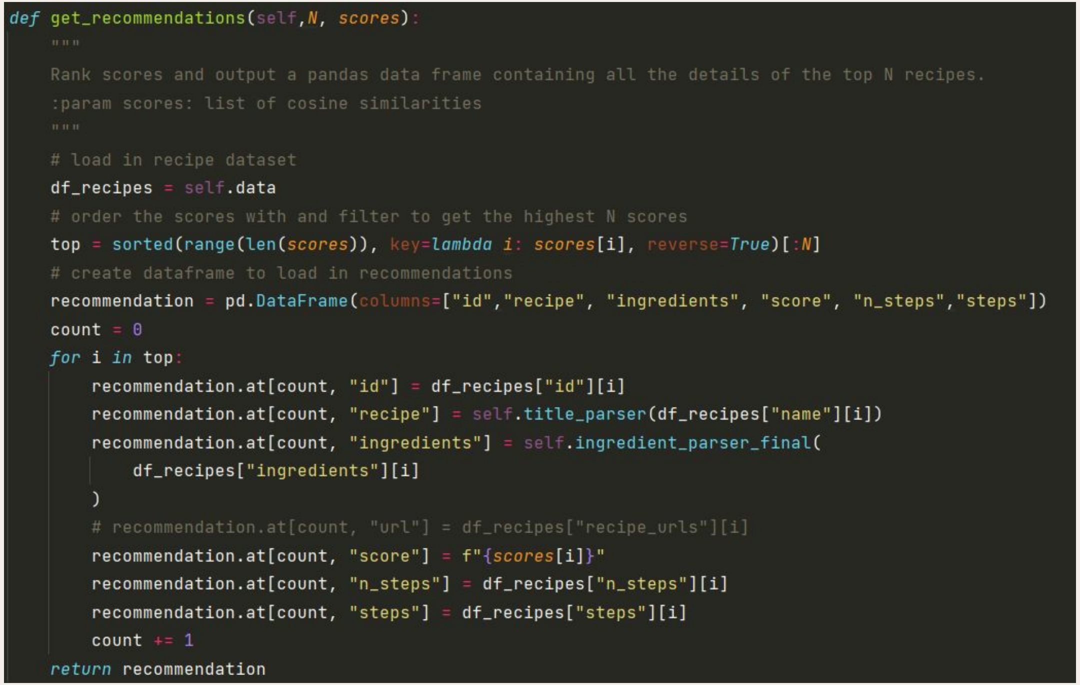
    ‘king’ - ‘man’ +’woman’ ≈’queen’
  - Comparing with one-hot vectorization, the word2vec occupies less space and it’s a dense vector.
- Remove stop words (non-alpha, measuring units and common words)
- Embedding the input words and dataset:
  - Average embedding: Simply average the word vectors
  - TF-IDF embedding (usually higher score): Use TF-IDF value of the word as the weight for weighted average
- Compute the cosine similarity of the embedded input and each record from the dataset
- List top N results

Content-based Code Walkthrough



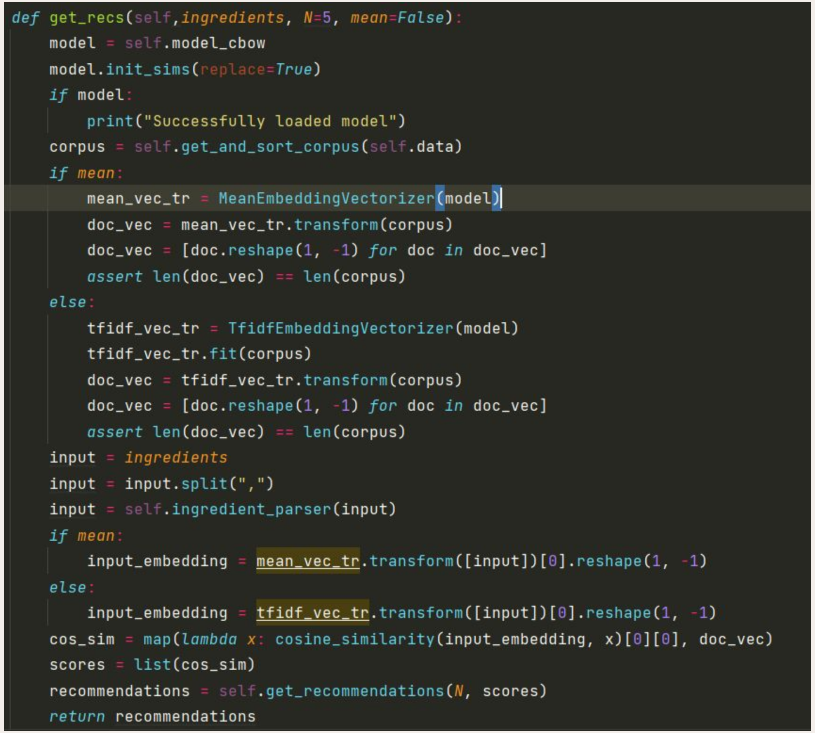
```python
def __init__(self, data_path='RAW_recipes.csv'):
    data = pd.read_csv(data_path)
    # parse the ingredients for each recipe
    data['parsed'] = data.ingredients.apply(self.ingredient_parser)
    self.data=data.iloc[:100000]

    # get corpus
    corpus = self.get_and_sort_corpus(data)
    print(f"Length of corpus: {len(corpus)}")

    #train and save CBOW Word2Vec model
    model_path=pathlib.Path('model_cbow.bin')
    if model_path.is_file():
        print('Find trained model.')
        self.model_cbow=Word2Vec.load("model_cbow.bin")
    else:
        model_cbow = Word2Vec(
            corpus, sg=0, workers=1, window=self.get_window(corpus), min_count=1,
        )
        filepath = Path('model_cbow.model')
        filepath.parent.mkdir(parents=True, exist_ok=True)
        MODELPATH = 'model_cbow.model'
        if model_cbow.save('model_cbow.bin'):
            print("Word2Vec model successfully trained")
        self.model_cbow=model_cbow
```

Content-based Code Walkthrough



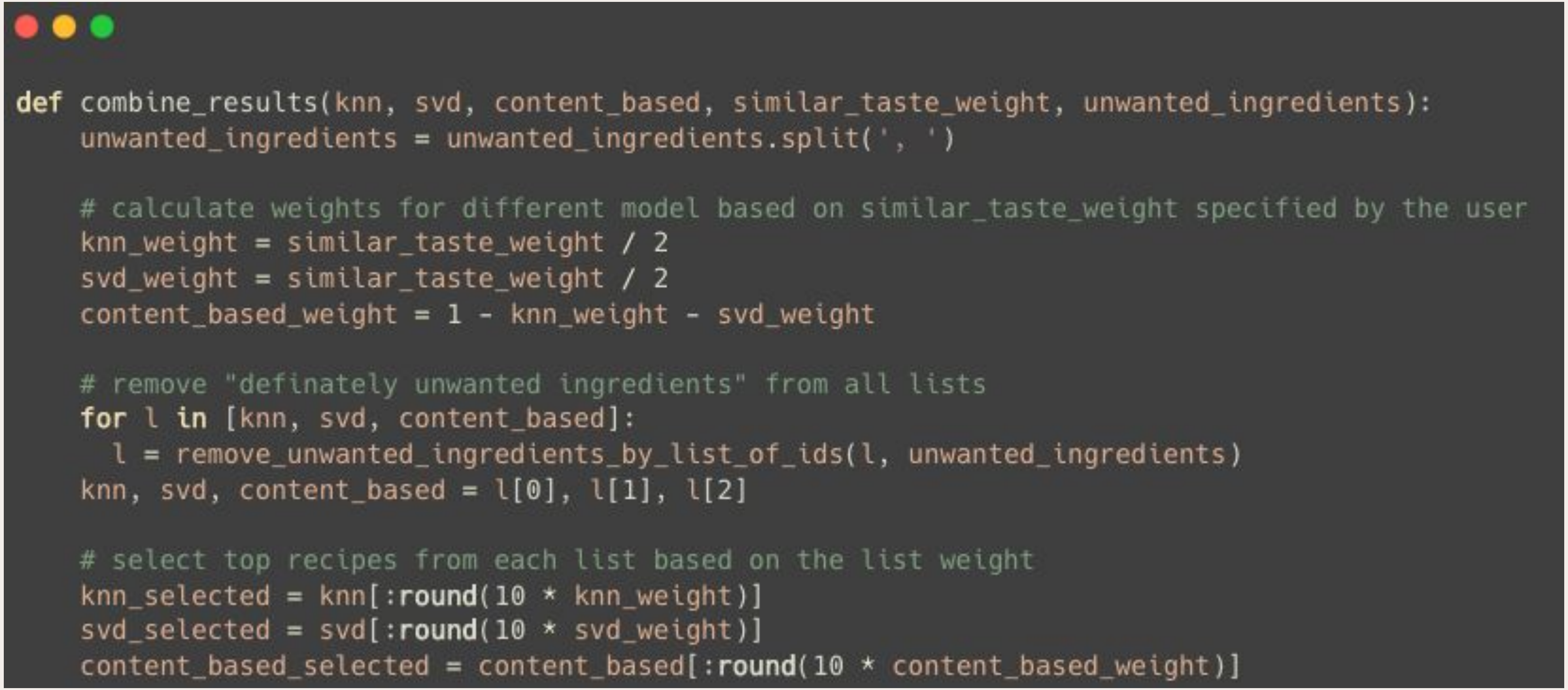
```
def get_recommendations(self,N, scores):
    """
    Rank scores and output a pandas data frame containing all the details of the top N recipes.
    :param scores: list of cosine similarities
    """
    # load in recipe dataset
    df_recipes = self.data
    # order the scores with and filter to get the highest N scores
    top = sorted(range(len(scores)), key=lambda i: scores[i], reverse=True)[:N]
    # create dataframe to load in recommendations
    recommendation = pd.DataFrame(columns=["id","recipe", "ingredients", "score", "n_steps","steps"])
    count = 0
    for i in top:
        recommendation.at[count, "id"] = df_recipes["id"][i]
        recommendation.at[count, "recipe"] = self.title_parser(df_recipes["name"][i])
        recommendation.at[count, "ingredients"] = self.ingredient_parser_final(
            df_recipes["ingredients"][i]
        )
        # recommendation.at[count, "url"] = df_recipes["recipe_urls"][i]
        recommendation.at[count, "score"] = f"{scores[i]}"
        recommendation.at[count, "n_steps"] = df_recipes["n_steps"][i]
        recommendation.at[count, "steps"] = df_recipes["steps"][i]
        count += 1
    return recommendation
```

Content-based Code Walkthrough



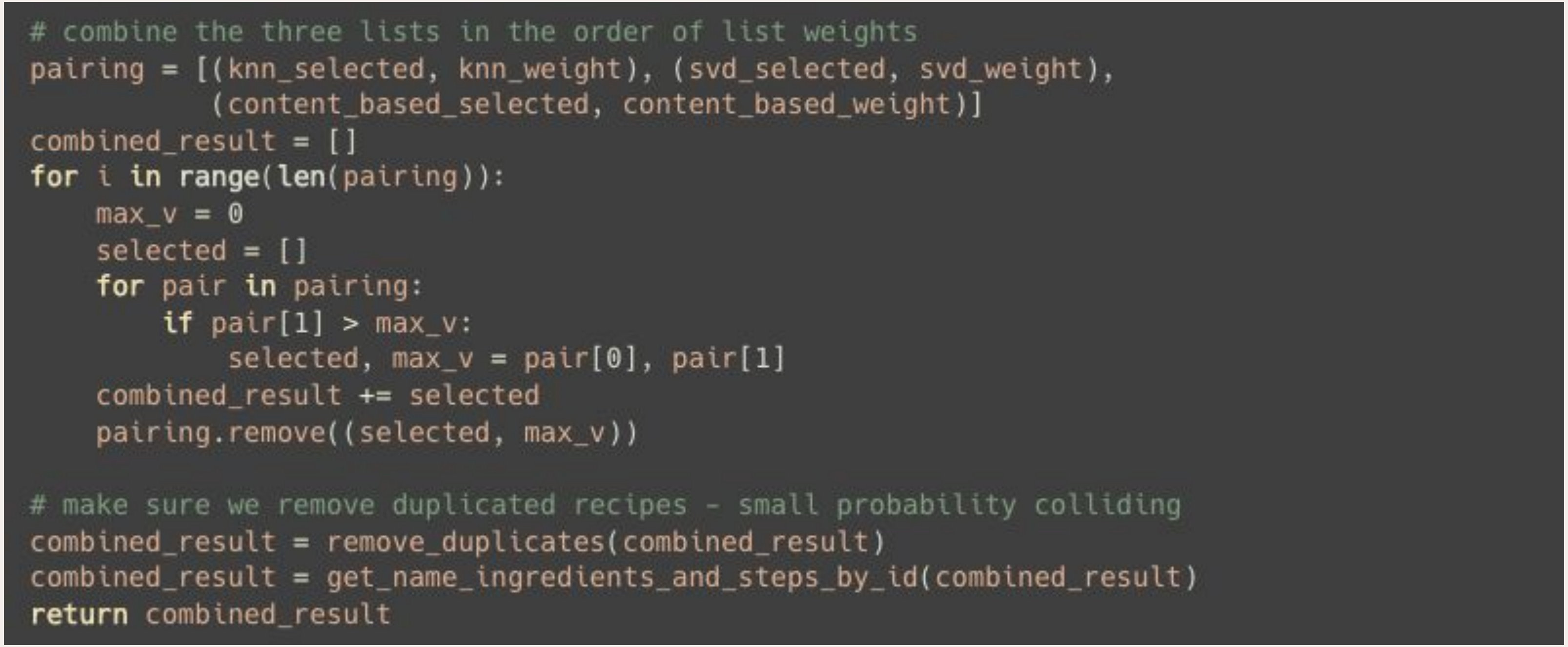
```
def get_recs(self,ingredients, N=5, mean=False):
    model = self.model_cbow
    model.init_sims(replace=True)
    if model:
        print("Successfully loaded model")
    corpus = self.get_and_sort_corpus(self.data)
    if mean:
        mean_vec_tr = MeanEmbeddingVectorizer(model)
        doc_vec = mean_vec_tr.transform(corpus)
        doc_vec = [doc.reshape(1, -1) for doc in doc_vec]
        assert len(doc_vec) == len(corpus)
    else:
        tfidf_vec_tr = TfidfEmbeddingVectorizer(model)
        tfidf_vec_tr.fit(corpus)
        doc_vec = tfidf_vec_tr.transform(corpus)
        doc_vec = [doc.reshape(1, -1) for doc in doc_vec]
        assert len(doc_vec) == len(corpus)
    input = ingredients
    input = input.split(",")
    input = self.ingredient_parser(input)
    if mean:
        input_embedding = mean_vec_tr.transform([input])[0].reshape(1, -1)
    else:
        input_embedding = tfidf_vec_tr.transform([input])[0].reshape(1, -1)
    cos_sim = map(lambda x: cosine_similarity(input_embedding, x)[0][0], doc_vec)
    scores = list(cos_sim)
    recommendations = self.get_recommendations(N, scores)
    return recommendations
```

Hybrid code walk-through



```
def combine_results(knn, svd, content_based, similar_taste_weight, unwanted_ingredients):
    unwanted_ingredients = unwanted_ingredients.split(', ')

    # calculate weights for different model based on similar_taste_weight specified by the user
    knn_weight = similar_taste_weight / 2
    svd_weight = similar_taste_weight / 2
    content_based_weight = 1 - knn_weight - svd_weight

    # remove "definately unwanted ingredients" from all lists
    for l in [knn, svd, content_based]:
      l = remove_unwanted_ingredients_by_list_of_ids(l, unwanted_ingredients)
    knn, svd, content_based = l[0], l[1], l[2]

    # select top recipes from each list based on the list weight
    knn_selected = knn[:round(10 * knn_weight)]
    svd_selected = svd[:round(10 * svd_weight)]
    content_based_selected = content_based[:round(10 * content_based_weight)]
```

Hybrid code walk-through

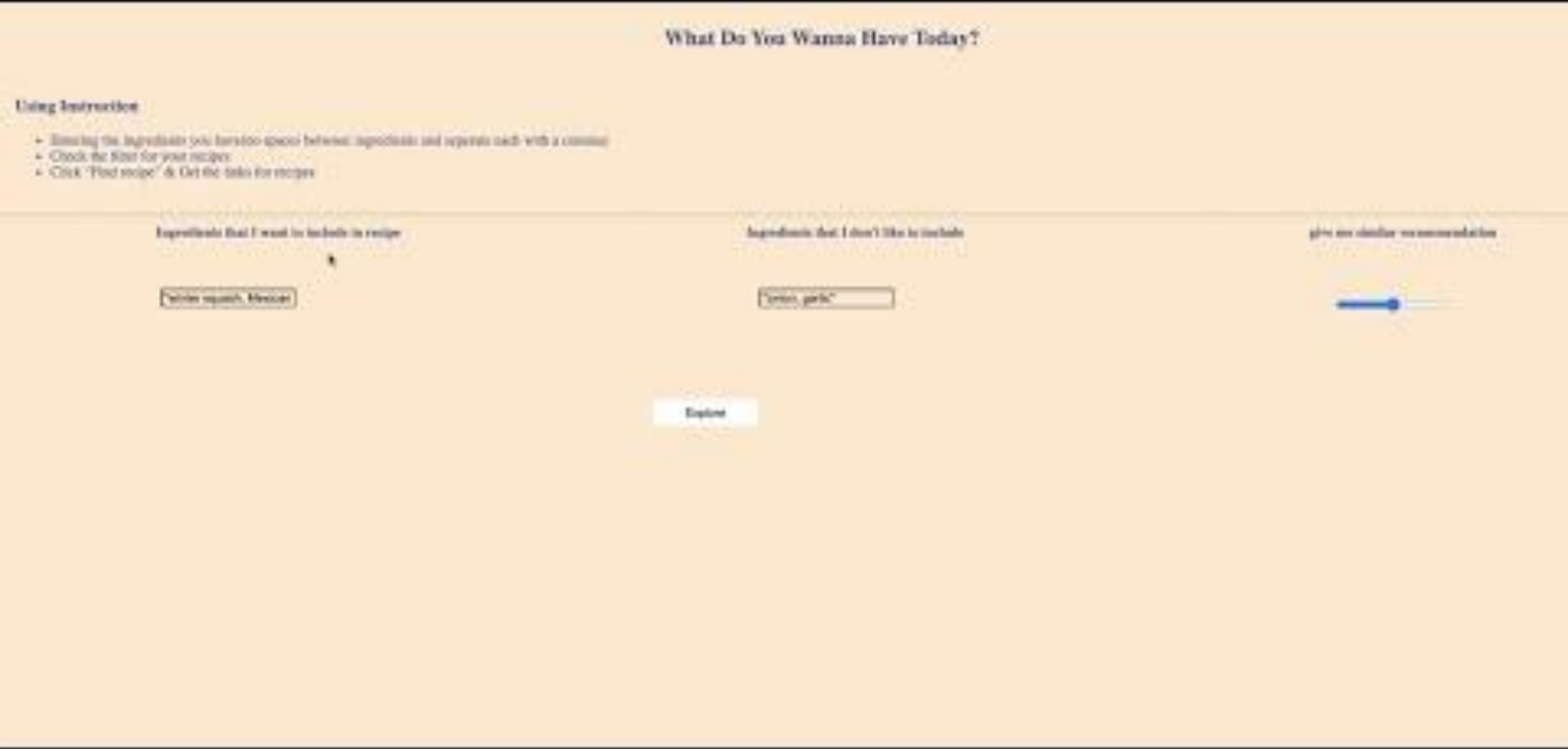


```python
# combine the three lists in the order of list weights
pairing = [(knn_selected, knn_weight), (svd_selected, svd_weight),
           (content_based_selected, content_based_weight)]
combined_result = []
for i in range(len(pairing)):
    max_v = 0
    selected = []
    for pair in pairing:
        if pair[1] > max_v:
            selected, max_v = pair[0], pair[1]
    combined_result += selected
    pairing.remove((selected, max_v))

# make sure we remove duplicated recipes - small probability colliding
combined_result = remove_duplicates(combined_result)
combined_result = get_name_ingredients_and_steps_by_id(combined_result)
return combined_result
```

API Demo

Demo 3 cases

Frontend Demo



What Do You Wanna Have Today?

Using Instruction

- Entering the ingredients you have(no space) between ingredients and separate each with a comma
- Check the filter for your recipes
- Click "Find recipe" & Get the links for recipes

Ingredients that I want to include in recipe: "winter squash, Mexican

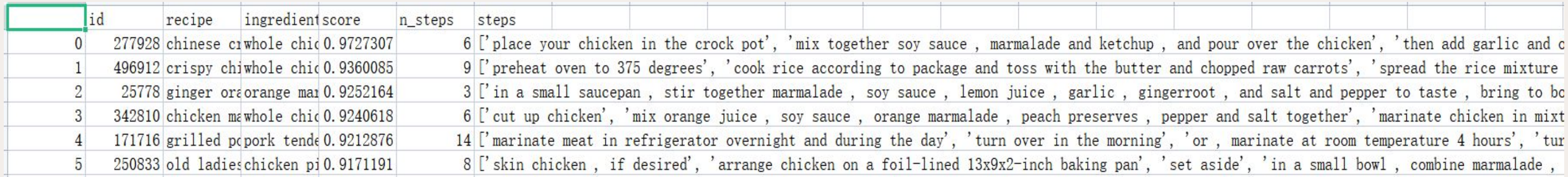
Ingredients that I don't like to include: "onion, garlic"

give me similar recommendation

Evaluation Metrics

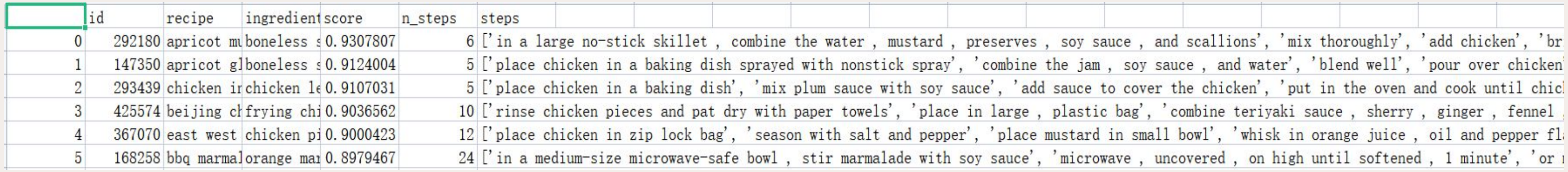
Content based Evaluation Metrics

- **Output from TF-IDF embedding**



	id	recipe	ingredient	score	n_steps	steps
0	277928	chinese c	whole chic	0.9727307	6	[’place your chicken in the crock pot’, ’mix together soy sauce , marmalade and ketchup , and pour over the chicken’, ’then add garlic and c
1	496912	crispy chi	whole chic	0.9360085	9	[’preheat oven to 375 degrees’, ’cook rice according to package and toss with the butter and chopped raw carrots’, ’spread the rice mixture
2	25778	ginger ora	orange ma	0.9252164	3	[’in a small saucepan , stir together marmalade , soy sauce , lemon juice , garlic , gingerroot , and salt and pepper to taste , bring to bo
3	342810	chicken ma	whole chic	0.9240618	6	[’cut up chicken’, ’mix orange juice , soy sauce , orange marmalade , peach preserves , pepper and salt together’, ’marinate chicken in mixt
4	171716	grilled p	pork tende	0.9212876	14	[’marinate meat in refrigerator overnight and during the day’, ’turn over in the morning’, ’or , marinate at room temperature 4 hours’, ’tur
5	250833	old ladies	chicken pi	0.9171191	8	[’skin chicken , if desired’, ’arrange chicken on a foil-lined 13x9x2-inch baking pan’, ’set aside’, ’in a small bowl , combine marmalade ,

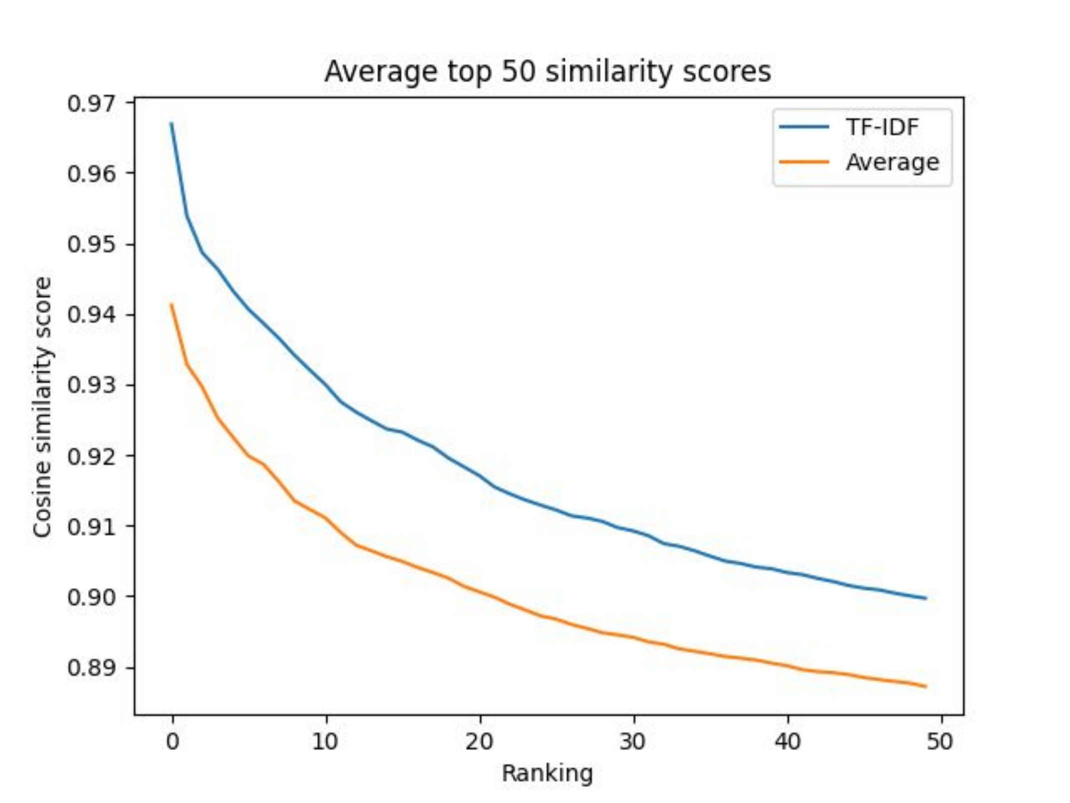
- **Output from Average embedding**



	id	recipe	ingredient	score	n_steps	steps
0	292180	apricot mu	boneless s	0.9307807	6	[’in a large no-stick skillet , combine the water , mustard , preserves , soy sauce , and scallions’, ’mix thoroughly’, ’add chicken’, ’br
1	147350	apricot gl	boneless s	0.9124004	5	[’place chicken in a baking dish sprayed with nonstick spray’, ’combine the jam , soy sauce , and water’, ’blend well’, ’pour over chicken’
2	293439	chicken i	chicken le	0.9107031	5	[’place chicken in a baking dish’, ’mix plum sauce with soy sauce’, ’add sauce to cover the chicken’, ’put in the oven and cook until chic
3	425574	beijing c	frying chi	0.9036562	10	[’rinse chicken pieces and pat dry with paper towels’, ’place in large , plastic bag’, ’combine teriyaki sauce , sherry , ginger , fennel ,
4	367070	east west	chicken pi	0.9000423	12	[’place chicken in zip lock bag’, ’season with salt and pepper’, ’place mustard in small bowl’, ’whisk in orange juice , oil and pepper fl
5	168258	bbq marmal	orange ma	0.8979467	24	[’in a medium-size microwave-safe bowl , stir marmalade with soy sauce’, ’microwave , uncovered , on high until softened , 1 minute’, ’or

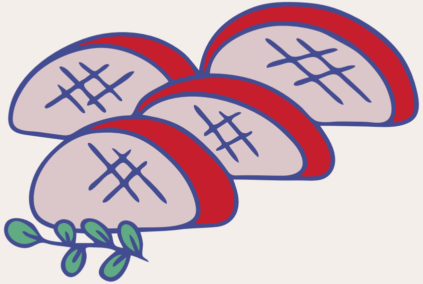
Content based Evaluation Metrics

Add observations

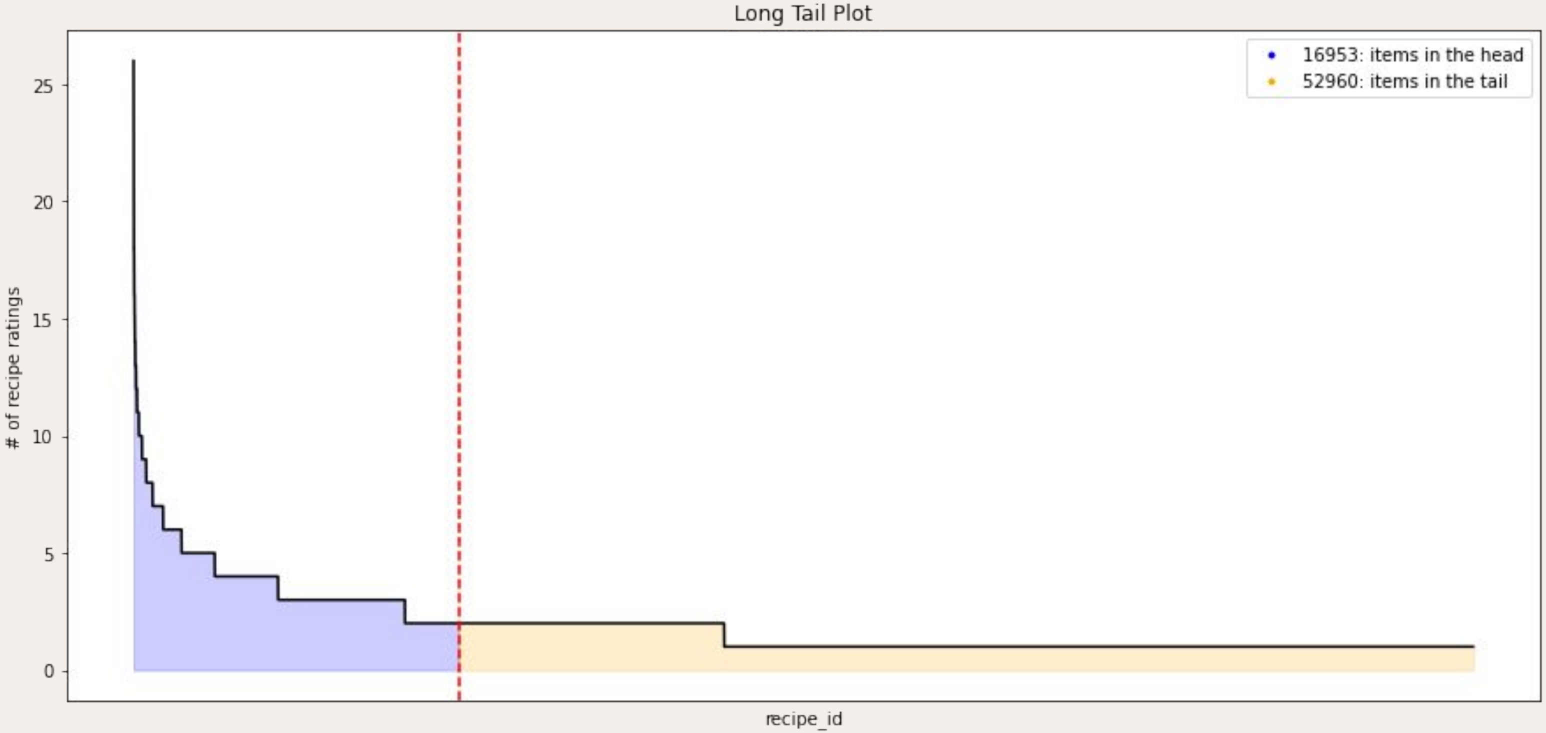


CF Evaluation Metrics

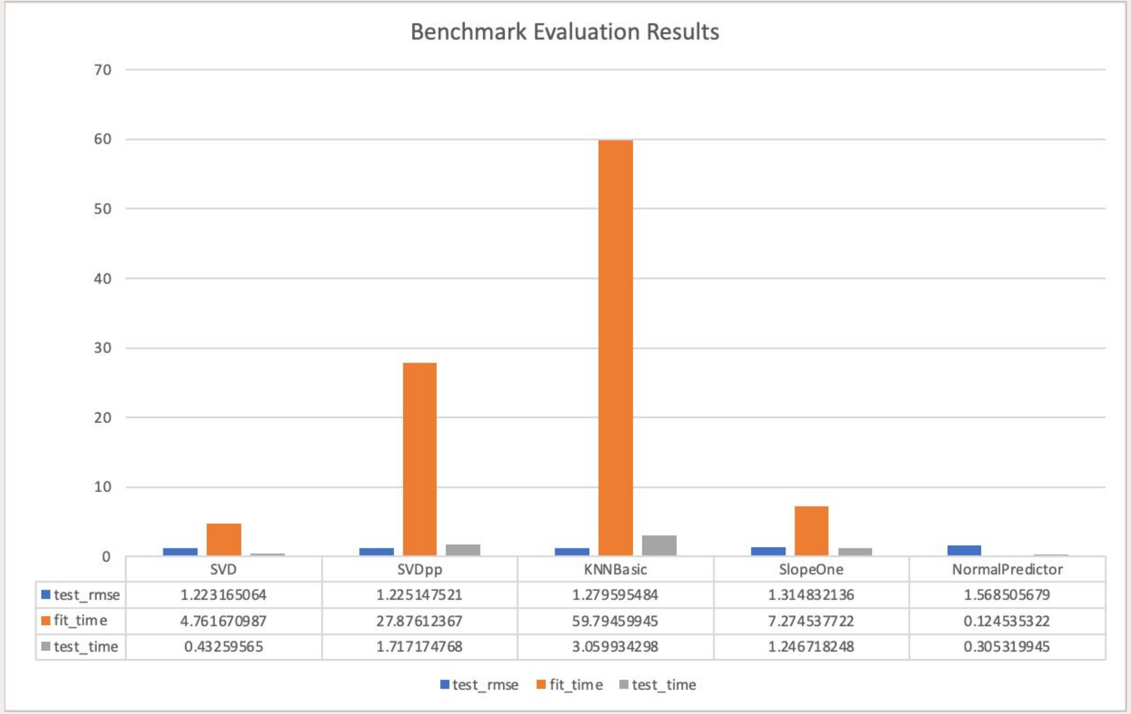
- **Preliminary Benchmark Evaluation Metrics**
  - RMSE, Fit time, Test time, Long Tail Plot
- **Model-specific Evaluation Metrics**
  - Precision & Recall Rates, Mar@k Plot, Precision Recall Curve
  - Coverage Plot, Classification Probability Plot
  - ROC AUC Plot
- **Cross-model Evaluation Metrics**
  - Coverage, Personalization, Intralist Similarity Score



Dataset Evaluation

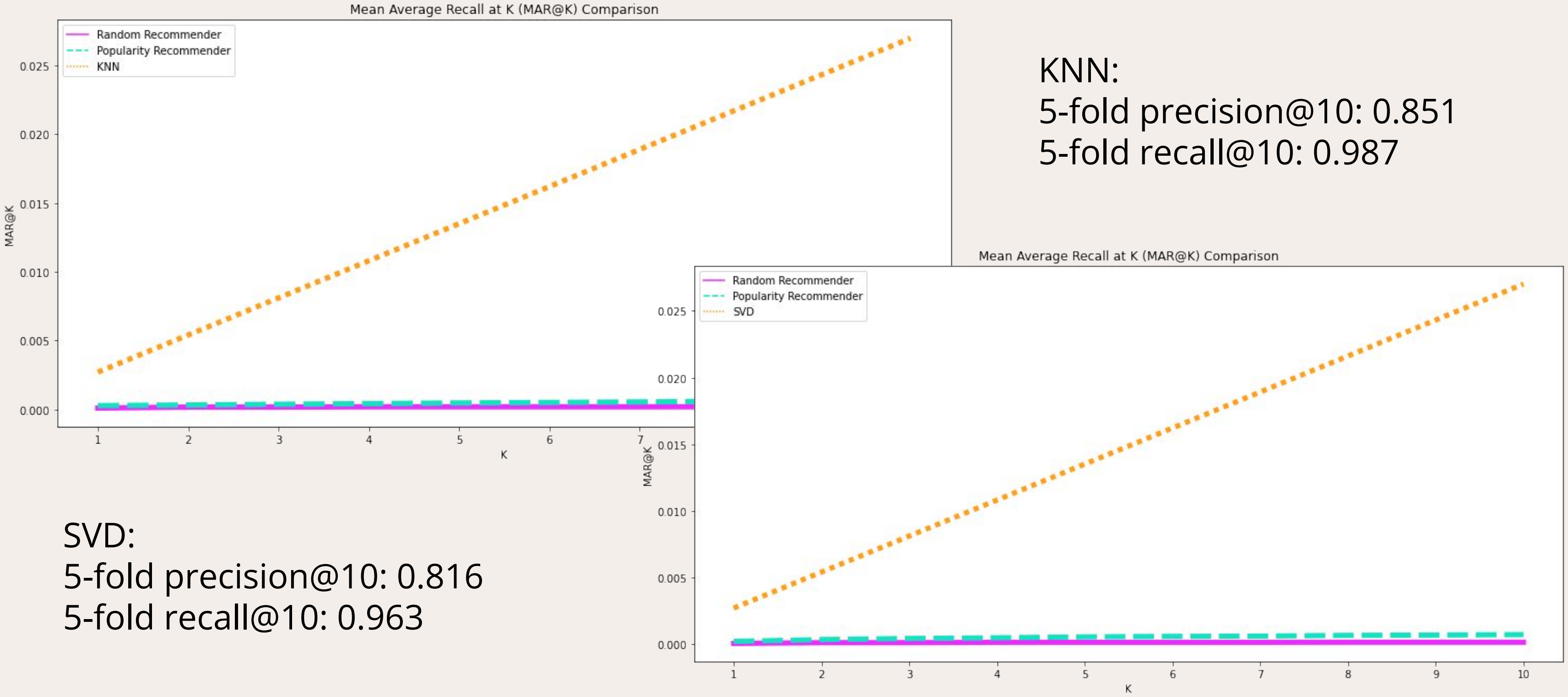


Benchmark Evaluation


Benchmark Evaluation Results

	SVD	SVDpp	KNNBasic	SlopeOne	NormalPredictor
■ test_rmse	1.223165064	1.225147521	1.279595484	1.314832136	1.568505679
■ fit_time	4.761670987	27.87612367	59.79459945	7.274537722	0.124535322
■ test_time	0.43259565	1.717174768	3.059934298	1.246718248	0.305319945

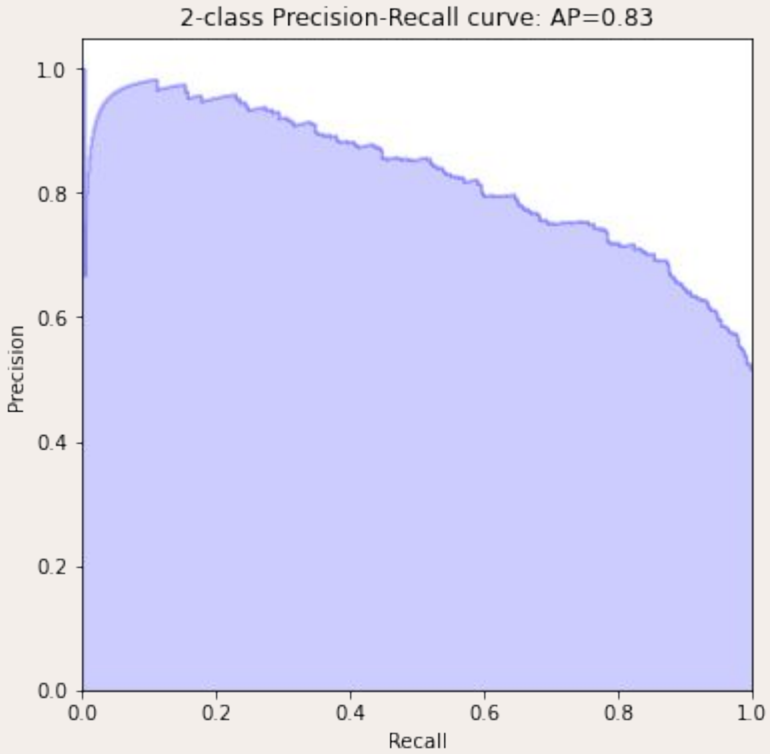
Precision Recall Rates Mar@k Plot



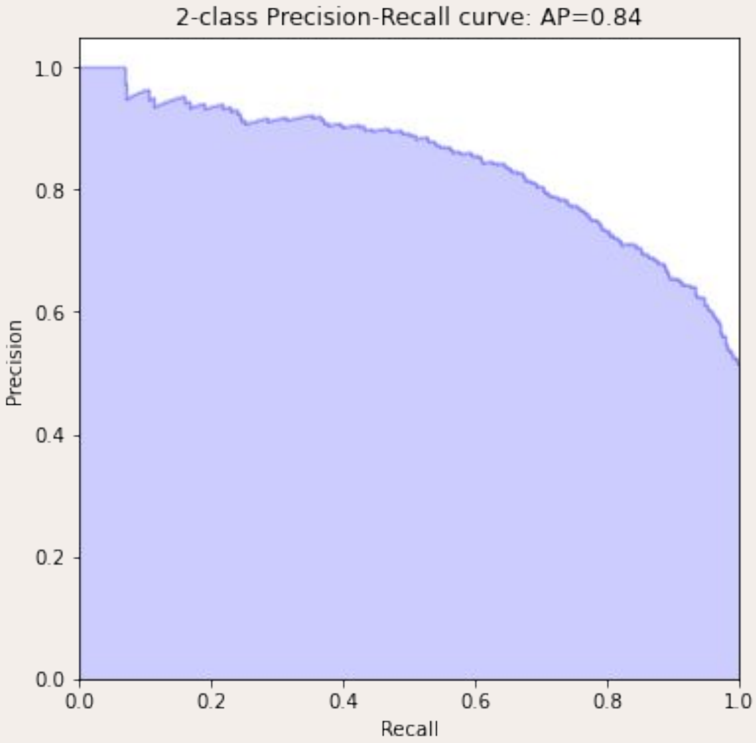
KNN:
5-fold precision@10: 0.851
5-fold recall@10: 0.987

SVD:
5-fold precision@10: 0.816
5-fold recall@10: 0.963

Precision Recall Curve

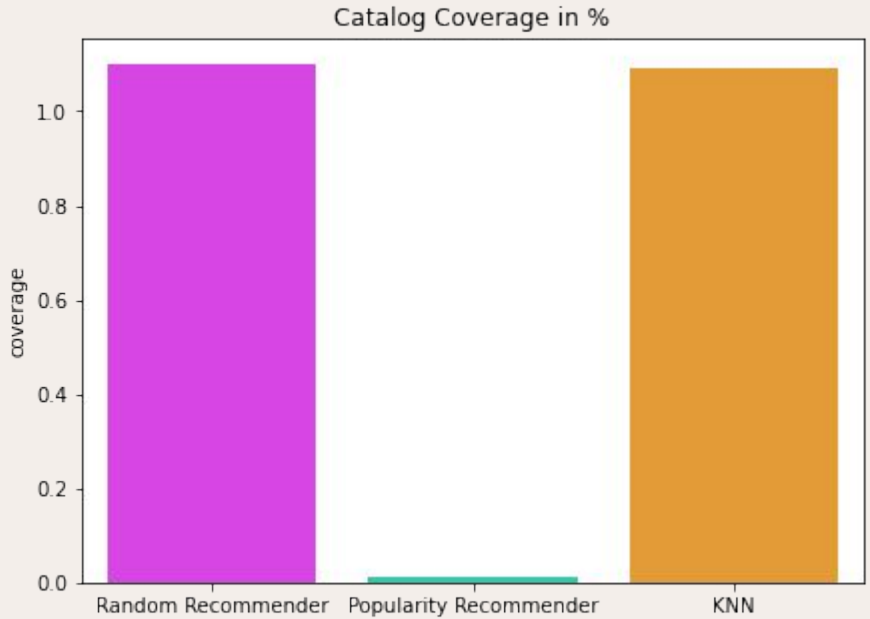


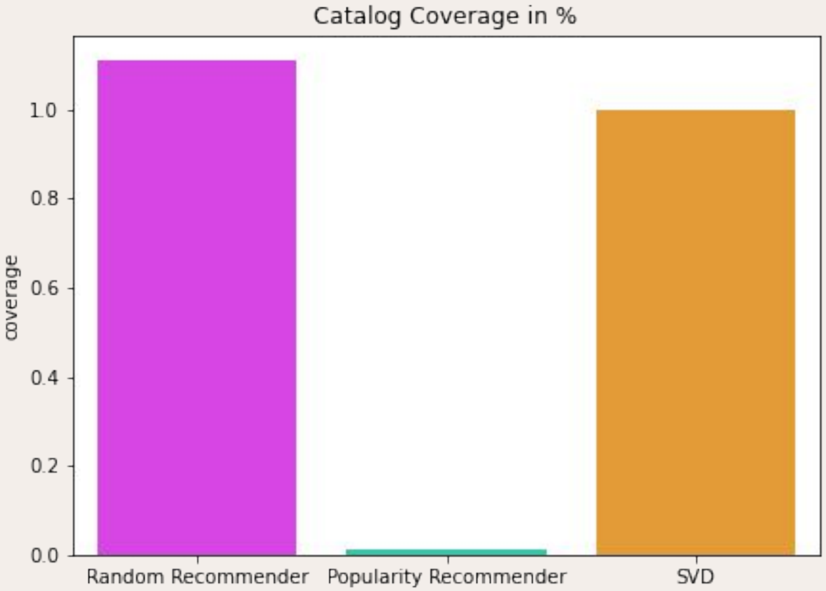
2-class Precision-Recall curve: AP=0.83



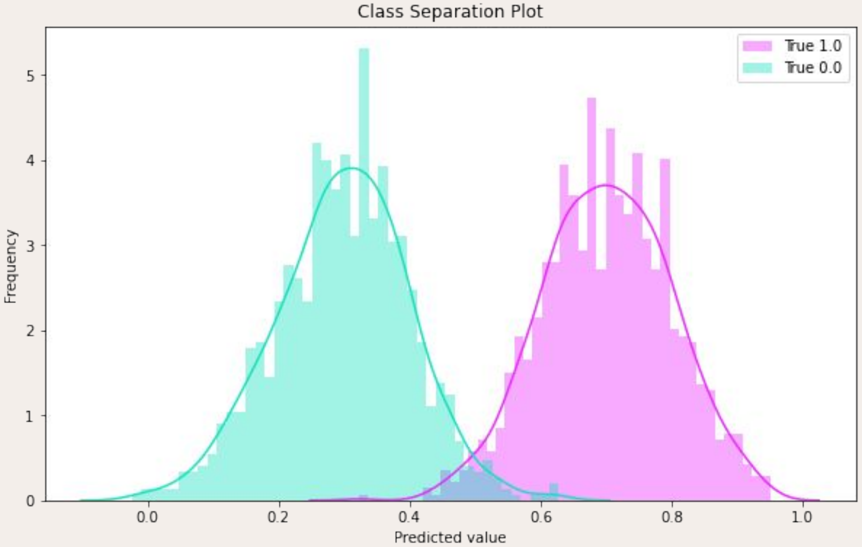
2-class Precision-Recall curve: AP=0.84

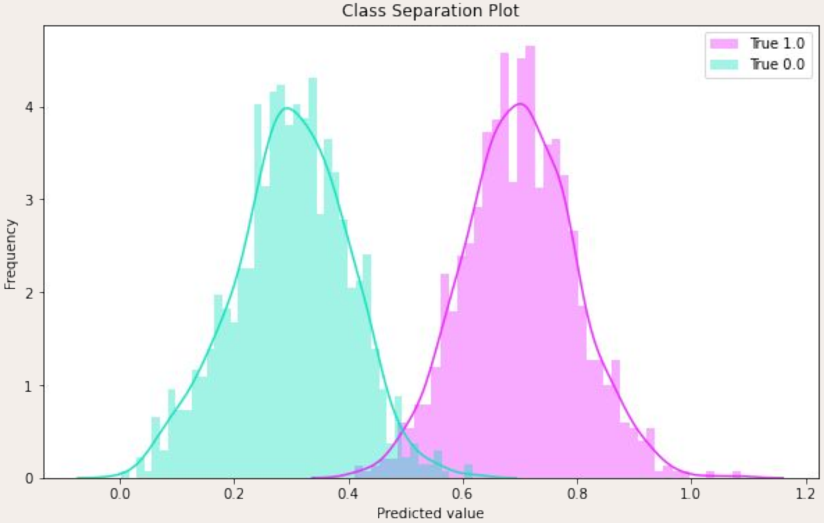
Coverage Plot



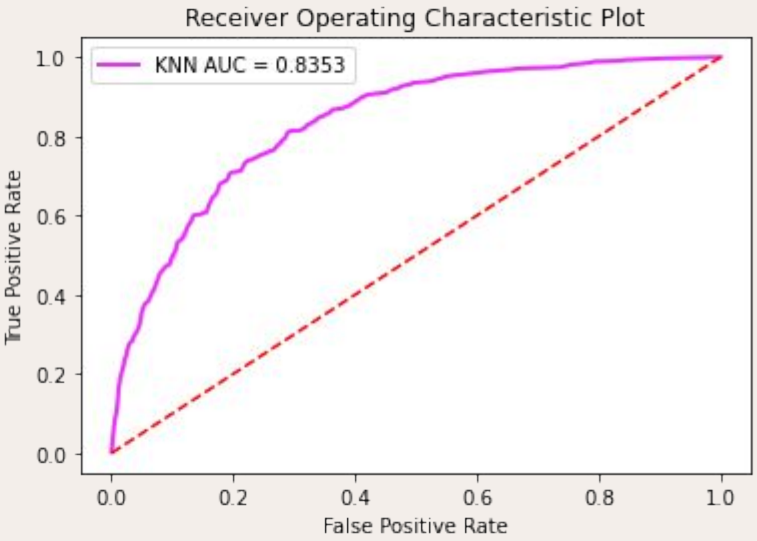


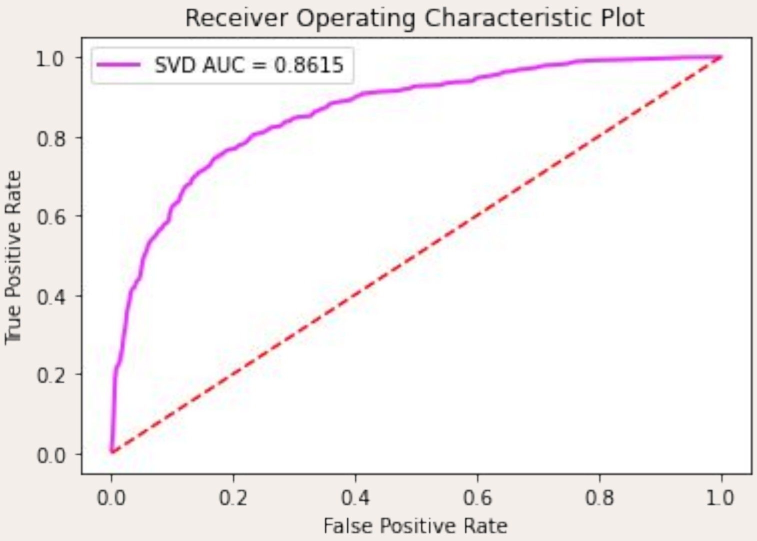
Classification Probability Plot





ROC AUC Plot


Receiver Operating Characteristic Plot
KNN AUC = 0.8353
True Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

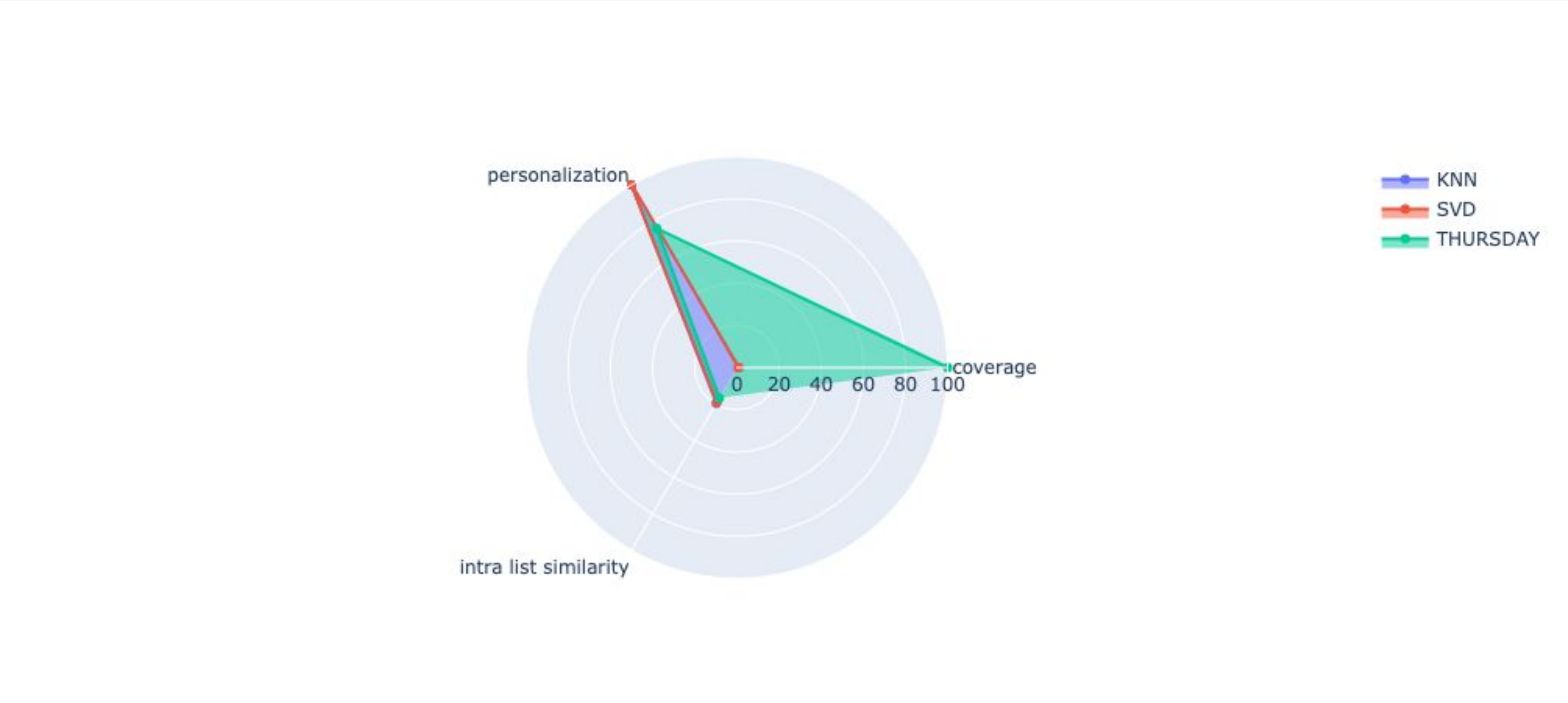

Receiver Operating Characteristic Plot
SVD AUC = 0.8615
True Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate

Cross-model Evaluation Metric

- **Coverage Score**
  - Percent of items that is able to recommend
- **Personalization Score**
  - Dissimilarity b/w user's lists of recs
- **Intra-list Similarity Score**
  - Calculate the cosine similarity b/w the items in a list of recs

$$coverage = \frac{I}{N} \times 100\%$$

Cross-model Evaluation Metric



Future Steps

- Create **Ingredients Object Detection** module
- Upgrading 3 control units to a **NLP module interpreting** sentences
- Implement **Deep Learning** recommender systems
- Develop **advanced features**
    - Search time estimation, recipe uploads, user profile creations
- Optimize **Frontend UI Design**
- Generalize hybrid algorithm into **Independent API** for arbitrary dataset and backend RS algorithms choices

THANKS!

DO YOU HAVE ANY QUESTIONS?

CS 4675/6675 Group 8
Shiyi W. Taichang Z. Shuangyue C.
Shuyan L. Haoran Z.

https://github.com/Shiyi-Wang/recipeRecSys

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon** and infographics & images by **Freepik**