URL Recommendation System from Popular Urls Bookmarked on Delicious

Min Shun
Department of Computer Science. Bina Nusantara University, Jakarta, Indonesia
Email: min.shun@binus.ac.id

This project was completed in January 2025 as part of the Machine Learning course in the Data Science program at Binus University. The author, Min Shun, would like to thank Mr. Jaffarus Sodiq for their guidance and support throughout the project.

Abstract: This paper presents a URL Recommendation System based on Content-Based Filtering, using popular URLs bookmarked on Delicious. The system leverages a dataset of 99,983 entries, including metadata such as SaveCount, Tags, and FirstSaveDate. Neural Networks are used to capture patterns in tag and URL relationships, while LightGBM enhances performance for large-scale data. Evaluation results show high accuracy in recommending relevant URLs, demonstrating the potential of advanced machine learning models to improve personalized content discovery.

Index Terms: URL Recommendation System, Delicious, Content-Based Filtering, Machine Learning, Neural Networks, LightGBM

1. Introduction

The rapid expansion of the internet has resulted in an overwhelming volume of online content, posing significant challenges for users seeking relevant and personalized information. Recommendation systems have emerged as indispensable tools in addressing this challenge by filtering and prioritizing content tailored to user preferences. Social bookmarking platforms, such as Delicious, offer an excellent case study for building such systems, as they provide insights into user behavior through metadata like tags, save counts, and temporal patterns. Harnessing this metadata can significantly improve the relevance and efficiency of content recommendations, thereby enhancing user satisfaction and engagement. [1][5]

One key motivation behind this study is the pressing need for scalable and effective recommendation systems that can adapt to diverse and sparse datasets. In platforms like Delicious, user-generated data often includes tagging behaviors and bookmarking trends, which are inherently heterogeneous and challenging to model.[2][4] The Delicious dataset, comprising 9,.983 popular URLs bookmarked within a historical time window, serves as a valuable resource for addressing this problem. Each URL entry in this dataset includes details such as the date of the first save, the total number of saves, and up to 10 user-generated tags with their respective counts. These features encapsulate temporal, numerical, and categorical dimensions of user interaction, offering a comprehensive foundation for building and evaluating recommendation systems.[5]

However, the problem of URL recommendation involves several challenges. First, the data is sparse, as many URLs may lack sufficient user activity or tagging information. Second, the temporal aspect of the data introduces dynamics that traditional recommendation algorithms may fail to capture. Third, the presence of missing values in tag-related columns necessitates careful preprocessing to ensure model performance. Addressing these challenges requires the integration of robust machine learning techniques capable of handling complex and multi-dimensional datasets.[3][6]

The primary objective of this study is to develop and evaluate machine learning approaches for building a URL recommendation system using the Delicious dataset. Specifically, this study aims to:

- Conduct an exploratory data analysis (EDA) to gain a deeper understanding of the dataset's characteristics, including trends, patterns, and potential anomalies.
- Implement data preparation techniques, including handling missing values, feature engineering, and data transformation, to ensure the dataset is suitable for modeling.
- Develop and evaluate three distinct machine learning models—k-Nearest Neighbors (KNN), neural networks, and LightGBM—to predict and recommend URLs based on user activity and tagging behavior.

This research is motivated by the need to achieve high accuracy in URL recommendations while maintaining computational efficiency. By leveraging modern machine learning techniques, the study seeks to provide insights into the effectiveness of different algorithms for handling sparse, temporal, and multi-modal data in social bookmarking platforms.[2] The findings of this research are expected to contribute to the broader field of recommendation systems, particularly in scenarios where user-generated metadata plays a crucial role.[1]

2. Methodology

The dataset used in this study comprises 99,983 entries representing popular URLs bookmarked on the Delicious platform. It provides a diverse and rich collection of metadata that captures user bookmarking behaviors over a historical time frame. Below is a detailed breakdown of the dataset's structure:

2.1. Dataset Details

The dataset used in this study comprises 99,983 entries representing popular URLs bookmarked on the Delicious platform. It provides a diverse and rich collection of metadata that captures user bookmarking behaviors over a historical time frame. Below is a detailed breakdown of the dataset's structure:

2.1.1. URL
- Represents the address of each bookmarked page.
- Treated as a categorical feature since each URL is unique and does not inherently possess ordinal or numerical characteristics.

2.1.2. SaveCount
- Indicates the total number of times a URL has been saved by users on the Delicious platform.
- This feature provides a measure of the URL's popularity and serves as a critical numerical variable in recommendation modeling.

2.1.3. FirstSaveDate
- Records the timestamp of the first instance a URL was bookmarked.
- While initially a temporal string, it was transformed into datetime format to extract additional features such as the year, month, and day.
- These extracted features capture temporal dynamics, such as trends in bookmarking activity over time.

2.1.4. Tags and Tag Counts
- The dataset includes up to 10 tags per URL (Tag1 to Tag10), which represent user-generated labels describing the URL’s content. Each tag is paired with a frequency count (Count1 to Count10), indicating the number of times that tag was associated with the URL.

- Tags are treated as categorical features, capturing semantic information about the URLs. The corresponding counts are numerical features that quantify the relevance of each tag to the URL.

The dataset spans from late 1970 onward, with nearly all URLs exhibiting save activity since their initial bookmarking. This historical depth provides an opportunity to analyze long-term trends in user behavior and tag usage. The data structure, consisting of 23 columns, can be divided into two main categories:

2.2. Challenges

The dataset also reflects several challenges:

2.2.1. Sparsity

Not all URLs have a complete set of 10 tags, resulting in missing values in Tag and Count columns.

2.2.2. High Dimensionality

With up to 10 tags and their corresponding counts for each URL, the dataset introduces high-dimensional features that require transformation to improve model efficiency.

2.2.3. Temporal Dynamics

The inclusion of FirstSaveDate allows for time-based analysis but requires careful encoding to preserve temporal relationships.

Overall, the dataset provides a robust foundation for developing a URL recommendation system, as it captures both the content characteristics of URLs (via tags) and their popularity trends (via save counts and timestamps). These attributes make it well-suited for machine learning applications, enabling the extraction of insights and the construction of accurate predictive models.

2.3. Exploratory Data Analysis

Exploratory Data Analysis (EDA) was conducted to gain a comprehensive understanding of the dataset's structure, uncover patterns, and identify potential challenges.[4] Key steps and insights derived from this process are summarized below.

2.3.1. Categorical Feature Analysis

- The frequency distributions of categorical columns (URL, FirstSaveDate, Tag1 to Tag10) were analyzed using value counts. This revealed the most frequently used tags, with popular ones like “news,” “tech,” and “blog” dominating the dataset.
- Insight: These tags indicate user preferences and prevalent themes on the Delicious platform, which may significantly influence URL recommendation models.

2.3.2. Temporal Feature Extraction

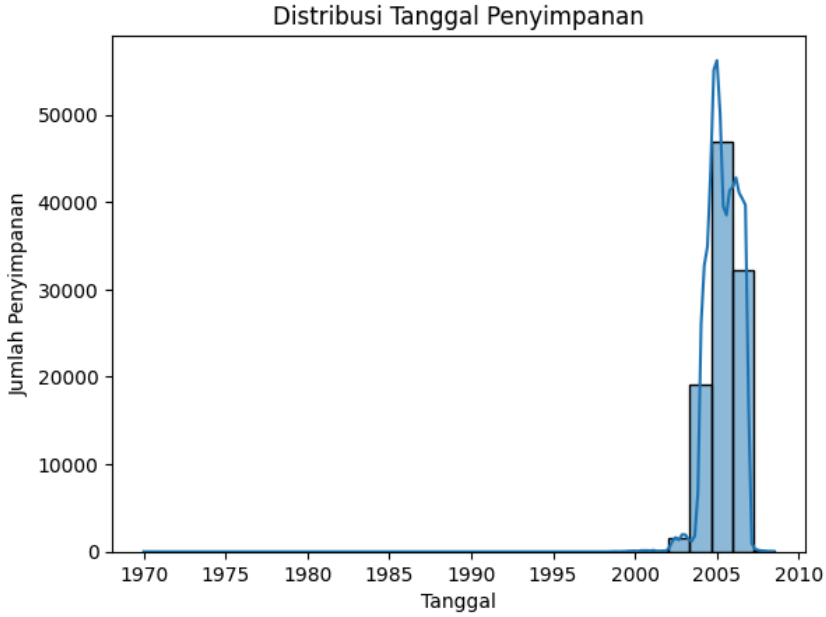
- The FirstSaveDate column was converted to datetime format, and additional features such as Year, Month, and DayOfWeek were derived.
- Insight: These temporal features allow for the identification of yearly growth trends, seasonal patterns, and day-of-week variations in bookmarking behavior.

2.3.3. Data Range Analysis

- The range of FirstSaveDate, spanning from January 1, 1970, to June 28, 2008, highlights a substantial historical coverage of approximately 38 years.
- Insight: This extensive time span enables an in-depth analysis of temporal trends, user engagement patterns, and evolutionary changes in user behavior over nearly four decades.

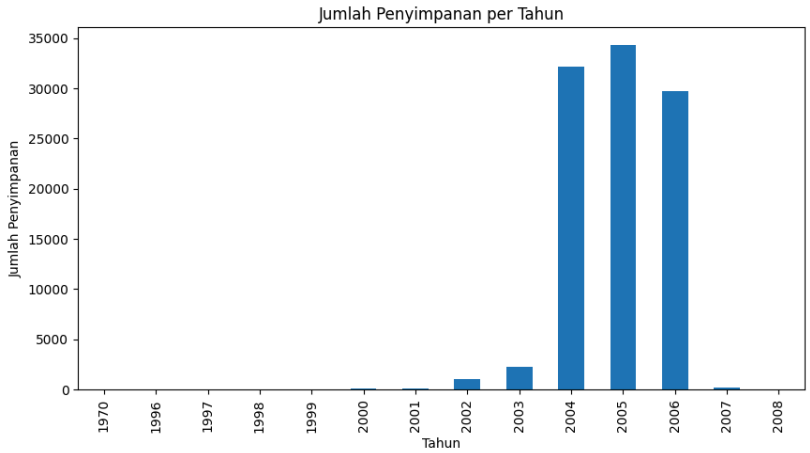
2.3.4. Temporal Trend Visualization

2.3.4.1. Distribution of First Save Dates



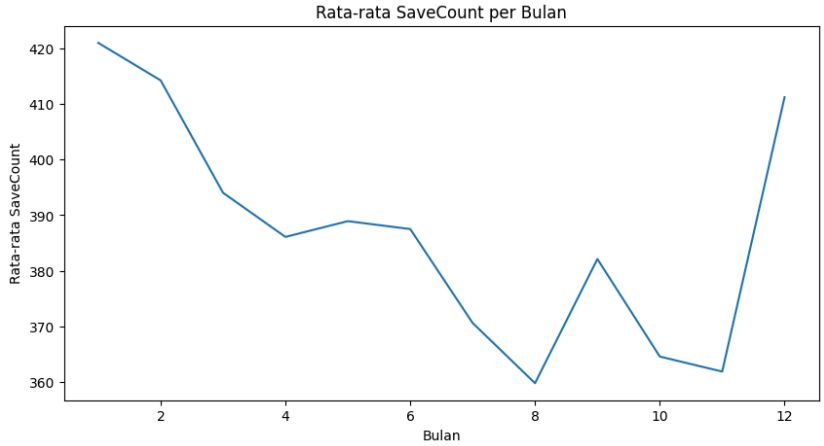
- The histogram with KDE reveals an uneven distribution of First Save Dates over time. Although the data technically starts from 1970, activity remains nearly nonexistent until around the early 2000s. A significant uptick in bookmarking activity begins around 2000–2005, reaching its peak between 2005 and 2010. Following this peak period, the activity shows a gradual decline in the subsequent years.
- Insight: The observed trend suggests that bookmarking activity was heavily influenced by the rise of internet adoption, increased availability of online platforms, and shifting user behaviors during the mid-2000s, before gradually tapering off.

2.3.4.2. Yearly Trends



- A bar chart of yearly save counts revealed distinct phases in user engagement over time. Bookmarking activity began to rise significantly around 2000, marking the start of rapid growth. This upward trend continued, reaching its peak between 2005 and 2006, before starting to decline in 2007 and 2008, the activity had dropped sharply, approaching near-zero levels.
- Insight: The observed growth and subsequent decline suggest that bookmarking activity was closely tied to the evolution and popularity of online platforms and user behavior trends during the early 2000s. The sharp drop in 2008 may indicate shifts in technology, platform usage, or changes in user engagement patterns.

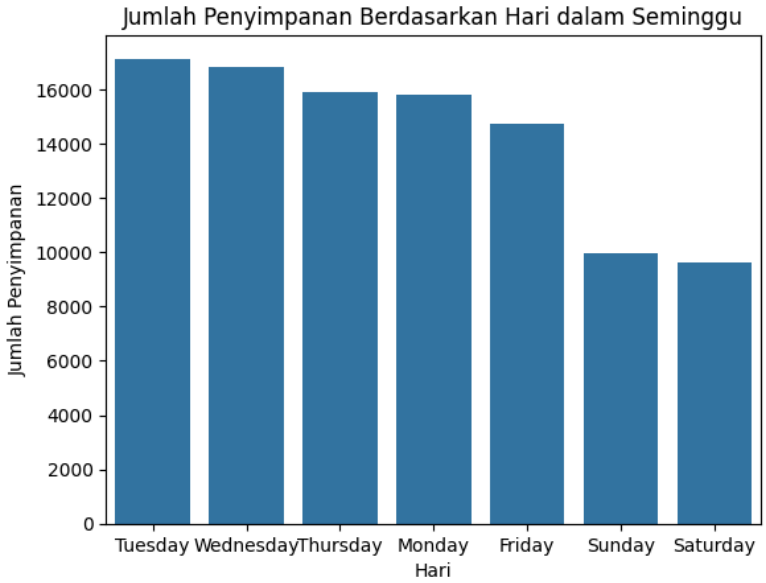
2.3.4.3. Monthly Save Count Trends



- A line chart of average SaveCount per month reveals distinct seasonal variations in bookmarking activity. The data shows a peak in January, followed by a gradual decline until April. Activity then experiences a slight increase towards June, before hitting its lowest point in August. In September, there's a noticeable uptick, but this is followed by another decline in October. Finally, activity surges again with a sharp increase in December.

- Insight: These monthly trends suggest that bookmarking activity might align with global seasonal patterns, holiday seasons, and user behavior cycles. The peaks in January and December could be influenced by New Year resolutions, end-of-year reflections, or increased online engagement during holiday periods, while the dip in August might correspond to vacation seasons or lower online activity globally.

2.3.4.4. Day of the Week Analysis



- An analysis of bookmarking activity across the days of the week reveals clear patterns in user engagement. The activity follows a specific order, with the highest activity occurring on Tuesday, followed closely by Wednesday and Thursday. Engagement then declines gradually through Monday and Friday, reaching its lowest points on Sunday and Saturday.
- Insight: This trend suggests that bookmarking activity is primarily driven by weekday routines, possibly related to work or academic tasks, with peak engagement mid-week. The drop during weekends indicates a shift in user focus towards leisure activities or reduced online productivity-related tasks.

2.3.5. Statistical Summary and Missing Value Analysis

- Summary statistics were computed for numerical features (SaveCount and Count1 to Count10), and missing values were analyzed. The missing percentages were minimal (<0.004%), and these rows were removed without significant data loss.
- Insight: The low percentage of missing data allowed for straightforward preprocessing by removing incomplete rows, ensuring the dataset remained intact and reliable for analysis without introducing bias or requiring complex imputation techniques.

2.3.6. Tag and Save Count Analysis

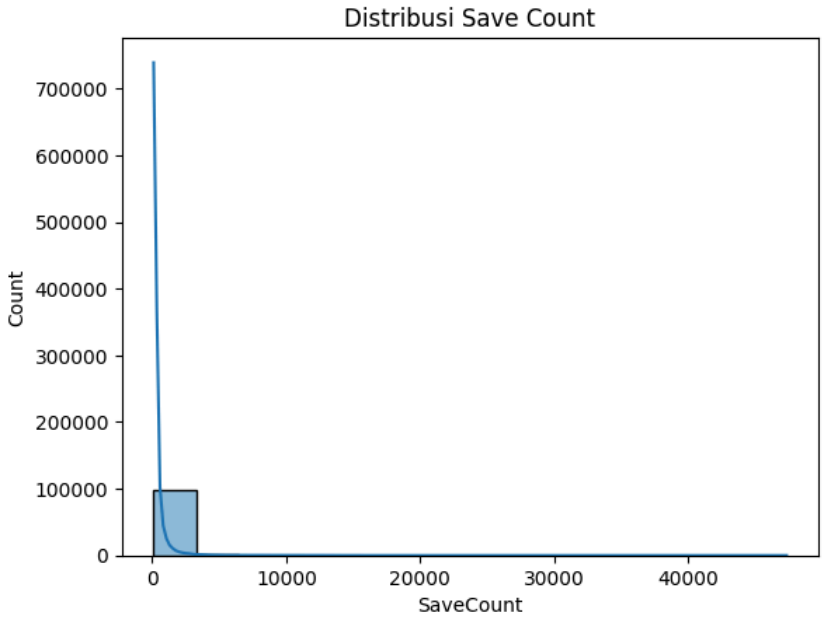
2.3.6.1. Most Common Tags

The top 10 tags for each Tag column were identified, with tags like “news” and “tech” appearing frequently.

2.3.6.2. Save Count by Tag

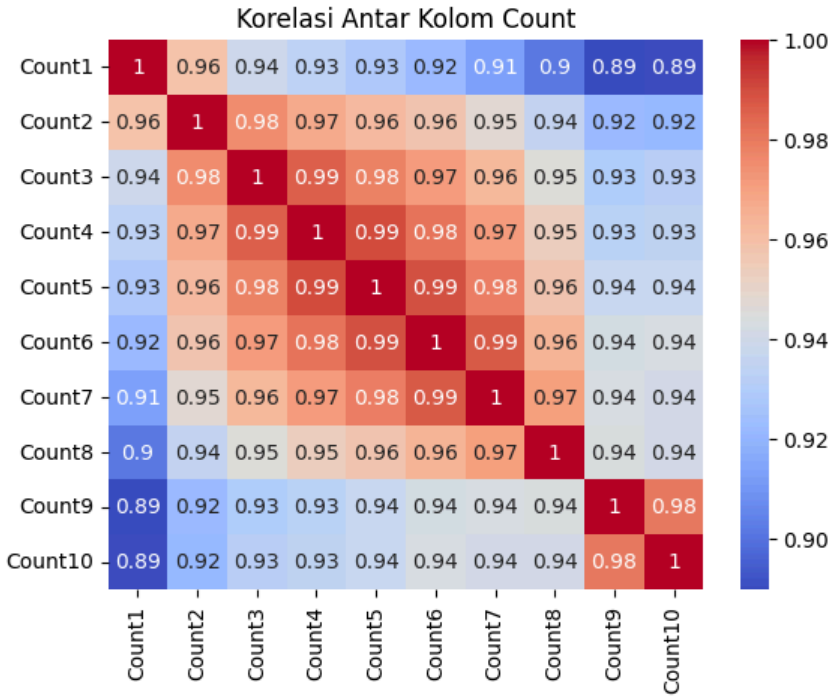
The average SaveCount for each tag highlighted the influence of specific tags on URL popularity.

2.3.6.3. Save Count Distribution



- A histogram with KDE for SaveCount identified a small subset of URLs with significantly higher save counts, indicating the presence of outliers.
- Insight: Popular tags significantly contribute to high save activity and are critical features for predictive modeling.

2.3.7. Correlation Analysis



- A heatmap of correlations between tag frequency columns (Count1 to Count10) revealed strong correlations (≥0.9), suggesting redundancy in these features.
- Insight: The high correlation indicates the potential for dimensionality reduction techniques, such as TF-IDF or PCA, to improve computational efficiency.

2.3.8. Unique URL Analysis

- A comparison of unique URLs to the total dataset confirmed that each entry corresponds to a unique URL.
- Insight: The absence of duplicates simplifies feature engineering and ensures data integrity for modeling.

2.4. Challenges and Solutions

This section outlines the key challenges identified during data exploration and describes the strategies employed to address them, along with the rationale for certain decisions.

2.4.1. Handling Null Values

- The dataset contained null values primarily in the Tag and Count columns. However, the percentage of missing values was extremely low (<0.004%)
- Solution: Rows with null values were removed to ensure data integrity while minimizing data loss. The negligible impact of this removal on the dataset size justified this approach.

2.4.2. Skewness in SaveCount

- The SaveCount feature exhibited significant skewness, with a small subset of URLs having exceptionally high save counts, indicating substantial differences in URL popularity.
- Solution: Rather than removing these outliers, they were retained to preserve the dataset's characteristic feature of capturing variations in URL popularity. These

variations are crucial for the recommendation system to distinguish between popular and less popular URLs.

2.4.3. High Correlation Among Count Features

- The tag frequency columns (Count1 to Count10) displayed extremely high correlations, with a minimum correlation of 0.89. While high correlation can introduce redundancy, it also reflects strong interdependence between tag frequencies for a given URL.
- Solution: To address this redundancy, TF-IDF (Term Frequency-Inverse Document Frequency) was applied to transform the tag features into a lower-dimensional space. This technique emphasizes unique tags while downweighting commonly used ones, thereby preserving the most meaningful aspects of tag data while reducing the impact of high correlations.

2.4.4. Imbalance in Yearly SaveCount

- The number of saves varied significantly across years, with a dominance of data from the period 2001–2006 and sparse activity afterward. The oldest save record was from 1970, suggesting potential inconsistencies in data collection or user activity.
- Solution: To preserve the temporal diversity of the dataset, all records were retained. This ensures that the recommendation system can account for variations in bookmarking trends over time. Temporal features, such as year, month, and day of the week, were engineered to help balance the influence of different time periods.

2.4.5. Skewness in Tag Frequency Count

- The distributions of tag frequency columns (Count1 to Count10) were heavily skewed, with a small number of tags having very high counts. This skewness reflects the uneven popularity of tags, where a few dominate across URLs.
- Solution: Skewness was addressed by applying TF-IDF during feature engineering. This transformation normalized tag frequencies, enabling the model to account for both dominant and less frequent tags effectively without losing critical information about tag usage patterns.

2.5. Data Preprocessing

Data preprocessing was conducted to prepare the dataset for machine learning and recommendation tasks, ensuring compatibility with the three models being developed: Content-Based Filtering, Neural Networks, and LightGBM. This involved encoding categorical variables, scaling numerical features, and splitting the data into training and testing subsets.

2.5.1. Encoding Categorical Features

Categorical variables were transformed into numerical representations using LabelEncoder. This included:

- Target Column Creation: The URL column was encoded into a new column, URL_encoded, to serve as the target variable (Y) for modeling. The original URL column was preserved for its role in the recommendation system.
- Other Categorical Features: Temporal and tag-related columns, such as FirstSaveDate, DayOfWeek, and Tag1 through Tag10, were encoded iteratively.

2.5.2. Scaling all Features (Except URL)

To ensure consistency across all numerical features, StandardScaler was used to standardize the dataset. This process was essential for Neural Networks, as they perform

best when features are normalized, and also beneficial for LightGBM to handle diverse feature scales.

2.5.3. Splitting the Dataset

The dataset was split into training and testing subsets to evaluate the performance of machine learning models, specifically LightGBM and Neural Networks. The feature set (X) included all columns except URL and URL_encoded, while the target (y) was the URL_encoded column. An 80:20 train-test split was used, with a fixed random state to ensure reproducibility.

2.6. Modelling

Three distinct approaches were implemented for building the URL recommendation system: Content-Based Filtering, Neural Networks, and LightGBM. Each model utilized different methodologies to leverage the features in the dataset, tailored to its strengths and requirements.

2.6.1. Content Based Filtering

Content-Based Filtering was implemented using a k-Nearest Neighbors (kNN) approach. Instead of the traditional matrix factorization, a kNN algorithm was chosen for its efficiency with large datasets.

2.6.1.1. Feature Engineering

- All feature columns, except URL and URL_encoded, were concatenated into a single string representation for each entry.
- A TF-IDF vectorizer was applied to transform the concatenated text into a sparse matrix representation.

2.6.1.2. Modeling

- A kNN model was built using the NearestNeighbors class with cosine similarity as the distance metric.
- The model was trained on the TF-IDF matrix to identify the 5 nearest neighbors for any given URL.

2.6.1.3. Recommendation Function

- A recommendation function was implemented to take a URL as input and return a list of the most similar URLs based on the trained kNN model.

2.6.2. Neural Network

A custom Neural Network model was developed to predict the URL_encoded target variable based on the dataset's features. The architecture was designed for regression tasks, avoiding embeddings since categorical data was pre-encoded.

2.6.2.1. Input Structure

- Separate input layers were created for each feature, including SaveCount, FirstSaveDate, and the tag-related columns (Tag1 to Tag10 and Count1 to Count10).
- These inputs were concatenated after flattening for further processing.

2.6.2.2. Architecture

- Dense layers were added with ReLU activations to learn complex patterns in the data.
- The output layer produced a single continuous value for the regression task.

2.6.2.3. Training and Evaluation

- The model was compiled with Mean Squared Error (MSE) as the loss function and the Adam optimizer for efficient gradient descent.
- Training was performed for 10 epochs with a batch size of 128, and RMSE was calculated on the test set.

2.6.3. LightGBM

LightGBM was implemented for its efficiency with large-scale data and its ability to handle mixed feature types.

2.6.3.1. Baseline Model

- A baseline LightGBM model was trained with standard parameters, using URL_encoded as the target.
- RMSE was computed on the test set for evaluation.

2.6.3.2. Hyperparameter Tuning

```
param_grid = {
    'learning_rate': [0.1, 0.2],
    'num_leaves': [31, 50, 100],
    'max_depth': [-1, 5, 10],
    'boosting_type': ['gbdt', 'dart']
}

model = lgb.LGBMRegressor(objective='regression', metric='rmse', verbose=-1, feature_pre_filter = False)
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, scoring='neg_root_mean_squared_error', cv=5, verbose=1)
grid_search.fit(X_train, y_train)

print("Best Hyperparameters:", grid_search.best_params_)
print("Best RMSE:", -grid_search.best_score_)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
C:\Users\minli\AppData\Local\Programs\Python\Python312\Lib\site-packages\numpy\ma\core.py:2846: RuntimeWarning: invalid va
  _data = np.array(data, dtype=dtype, copy=copy,
Best Hyperparameters: {'boosting_type': 'gbdt', 'learning_rate': 0.2, 'max_depth': 5, 'num_leaves': 31}
Best RMSE: 0.9769356223133141
```

- A grid search was performed to optimize hyperparameters, including learning_rate, num_leaves, max_depth, and boosting_type.
- The best parameters were selected based on RMSE from cross-validation.

3. Result

In this section, the performance of the three models: Content-Based Filtering, Neural Networks, and LightGBM are evaluated.

3.1. Content Based Filtering

Content - Based Filtering

```
# karena datanya banyak, saya menggunakan knn approach aja daripada matrix approach
feature = X.astype(str).apply(lambda x: ' '.join(x), axis=1)

tfidf = TfidfVectorizer()
tfidf_matrix = tfidf.fit_transform(feature)

model = NearestNeighbors(n_neighbors=6, metric='cosine', algorithm='brute')
model.fit(tfidf_matrix)

def recommend(url, df):
    if url not in df['URL'].values:
        return f"Error: URL '{url}' not found in the dataset."

    idx = df.index[df['URL'] == url][0]
    distances, indices = model.kneighbors(tfidf_matrix[idx])
    recommended_indices = indices[0][1:]
    return df.iloc[recommended_indices]['URL'].values

recommended_urls = recommend('http://www.elearningpapers.eu/index.php?page=home', df)
print("Recommended URLs:")
for url in recommended_urls:
    print(url)
```

```
Recommended URLs:
http://www.beatnikpad.com/archives/2006/06/04/firefox-1504
http://cooliris.com/
https://addons.mozilla.org/firefox/951/
http://labs.unoh.net/2006/10/web20firefox.html
http://69.90.152.144/collab/GreaseMonkeyUserScripts
```

The content-based filtering model, implemented using the k-Nearest Neighbors (kNN) algorithm, successfully returned the top 5 most similar URLs when a given URL was input. The model performed well in identifying relevant URLs based on metadata such as tags, save counts, and first save dates. The kNN algorithm was efficient, leveraging cosine similarity to calculate the nearest neighbors.

- Input: A URL with associated tags and save counts.
- Output: A list of 5 most similar URLs based on the trained kNN model.

3.2. Neural Network

Neural Network Approach

```
savecount_input = Input(shape=(1,), name='savecount_input')
firstsavedate_input = Input(shape=(1,), name='firstsavedate_input')
year_input = Input(shape=(1,), name='year_input')
month_input = Input(shape=(1,), name='month_input')
dayofweek_input = Input(shape=(1,), name='dayofweek_input')
tag_inputs = [Input(shape=(1,), name=f'tag{i}_input') for i in range(1, 11)]
count_inputs = [Input(shape=(1,), name=f'count{i}_input') for i in range(1, 11)]

savecount_vector = Flatten()(savecount_input)
firstsavedate_vector = Flatten()(firstsavedate_input)
year_vector = Flatten()(year_input)
month_vector = Flatten()(month_input)
dayofweek_vector = Flatten()(dayofweek_input)
tag_vectors = [Flatten()(tag_input) for tag_input in tag_inputs]  # Directly use encoded values
count_vectors = [Flatten()(count_input) for count_input in count_inputs]

concatenated = Concatenate()([
    savecount_vector, firstsavedate_vector, year_vector, month_vector, dayofweek_vector,
    *tag_vectors, *count_vectors
])

dense1 = Dense(128, activation='relu')(concatenated)
dense2 = Dense(64, activation='relu')(dense1)
output = Dense(1)(dense2)

model = Model(inputs=[savecount_input, firstsavedate_input, year_input, month_input, dayofweek_input] + tag_inputs + count_inputs, outputs=output)
model.compile(optimizer=Adam(learning_rate=0.001), loss='mean_squared_error')
```

The custom neural network model trained for URL recommendation showed promising results, with the loss decreasing over 10 epochs. The model's performance, evaluated using RMSE,

resulted in a value of 0.9948 on the test set.

```
history = model.fit(
    [X_train['SaveCount'], X_train['FirstSaveDate'], X_train['Year'], X_train['Month'], X_train['DayOfWeek']] +
    [X_train[f'Tag{i}'] for i in range(1, 11)] + [X_train[f'Count{i}'] for i in range(1, 11)],
    y_train,
    validation_data=(
        [X_test['SaveCount'], X_test['FirstSaveDate'], X_test['Year'], X_test['Month'], X_test['DayOfWeek']] +
        [X_test[f'Tag{i}'] for i in range(1, 11)] + [X_test[f'Count{i}'] for i in range(1, 11)],
        y_test
    ),
    epochs=10,
    batch_size=128
)

y_pred = model.predict(
    [X_test['SaveCount'], X_test['FirstSaveDate'], X_test['Year'], X_test['Month'], X_test['DayOfWeek']] +
    [X_test[f'Tag{i}'] for i in range(1, 11)] + [X_test[f'Count{i}'] for i in range(1, 11)]
)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"RMSE: {rmse:.4f}")
```

```
Epoch 1/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 9s 7ms/step - loss: 1.0131 - val_loss: 1.0001
Epoch 2/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.9932 - val_loss: 0.9890
Epoch 3/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.9803 - val_loss: 0.9877
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - loss: 0.9790 - val_loss: 0.9890
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.9762 - val_loss: 0.9918
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.9764 - val_loss: 0.9895
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.9740 - val_loss: 0.9883
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - loss: 0.9756 - val_loss: 0.9906
Epoch 4/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - loss: 0.9790 - val_loss: 0.9890
Epoch 5/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.9762 - val_loss: 0.9918
Epoch 6/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.9764 - val_loss: 0.9895
Epoch 7/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.9740 - val_loss: 0.9883
Epoch 8/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 7ms/step - loss: 0.9756 - val_loss: 0.9906
Epoch 9/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.9643 - val_loss: 0.9877
Epoch 10/10
625/625 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.9705 - val_loss: 0.9896
625/625 ━━━━━━━━━━━━━━━━━━━━ 5s 7ms/step
RMSE: 0.9948
```

```
model.summary()
```

			flatten_120[0][0], flatten_121[0][0], flatten_122[0][0], flatten_123[0][0], flatten_124[0][0]
dense_12 (Dense)	(None, 128)	3,328	concatenate_4[0][0]
dense_13 (Dense)	(None, 64)	8,256	dense_12[0][0]
dense_14 (Dense)	(None, 1)	65	dense_13[0][0]

Total params: 34,949 (136.52 KB)

Trainable params: 11,649 (45.50 KB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 23,300 (91.02 KB)

The training and validation loss decreased steadily, with a slight divergence in the final epochs, indicating that the model was converging and finding optimal weights. The neural network performed reasonably well, considering the complexity of the data.

3.3. LightGBM

LightGBM Approach

```
[66]: train_data = lgb.Dataset(X_train, label=y_train)
      test_data = lgb.Dataset(X_test, label=y_test, reference=train_data)

      params = {
          'objective': 'regression',
          'metric': 'rmse',
          'boosting_type': 'gbdt',
          'learning_rate': 0.2,
          'num_leaves': 50,
          'max_depth': 5,
          'feature_pre_filter': False,
          'verbose': -1
      }

      model = lgb.train(
          params,
          train_data,
          valid_sets=[test_data]
      )

      y_pred = model.predict(X_test, num_iteration=model.best_iteration)

      rmse = np.sqrt(mean_squared_error(y_test, y_pred))

      print(f"RMSE: {rmse:.4f}")

      importance = model.feature_importance()
      feature_names = X.columns
      feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importance})
      print(feature_importance_df.sort_values(by='Importance', ascending=False))
```

The LightGBM model, trained on the dataset after hyperparameter tuning, exhibited strong performance, with an RMSE of 0.9757. Feature importance analysis revealed that the tags (Tag1, Tag2, and Tag4) played significant roles in the model's predictions, with SaveCount and FirstSaveDate also being key predictors.

```
RMSE: 0.9757
          Feature  Importance
2            Tag1         282
4            Tag2         221
8            Tag4         160
6            Tag3         159
1   FirstSaveDate         156
12           Tag6         144
10           Tag5         143
0       SaveCount         138
16           Tag8         134
20          Tag10         131
14           Tag7         125
18           Tag9         119
3          Count1         109
5          Count2          99
7          Count3          72
23          Month          57
21        Count10          50
9          Count4          48
11         Count5          46
13         Count6          44
17         Count8          43
15         Count7          42
19         Count9          39
24      DayOfWeek          32
22           Year           1
```

4. Discussion

The results obtained from the three models (Content-Based Filtering, Neural Networks, and LightGBM) demonstrate promising performance, but also highlight key challenges.

4.1. Content-Based Filtering:

The k-Nearest Neighbors approach for content-based filtering efficiently provided similar URLs based on tags and save count. While the model worked well, it could be further improved by incorporating more sophisticated text representations such as word embeddings. Additionally, increasing the number of neighbors or exploring other distance metrics could refine the recommendation process.

4.2. Neural Networks:

Although the neural network model showed steady progress in minimizing the loss over the epochs, it exhibited slight divergence between the training and validation losses in the later epochs, suggesting potential overfitting. This is a common issue with deep learning models, particularly when trained on complex datasets like the Delicious dataset. The model's ability to generalize could be further improved by employing techniques such as regularization, dropout, or increasing the size of the training data.[6]

As the author is still in the process of deepening their understanding of deep learning, this area will be explored further. The neural network model provided valuable insights into how deep learning can be leveraged for recommendation tasks, and these insights will serve as important learning experiences for future exploration and application.

4.3. LightGBM:

The LightGBM model demonstrated high efficiency in handling the large-scale dataset, and hyperparameter tuning significantly improved its performance, yielding an RMSE of 0.9757. The model's feature importance scores indicated that tags such as "Tag1" and "Tag2," along with SaveCount, played critical roles in the recommendations. LightGBM’s ability to handle categorical features and large datasets efficiently, while maintaining high predictive accuracy, makes it a strong candidate for large-scale recommendation systems.

5. Future Challenges

5.1. Model Overfitting in Neural Networks:

Despite showing a decrease in loss, the slight divergence between training and validation loss in later epochs indicated the possibility of overfitting, a common issue with neural networks on complex datasets.

5.2. Feature Engineering:

While feature importance analysis helped identify key features in LightGBM, the process of selecting and fine-tuning features for neural networks and kNN required careful adjustments, especially when dealing with noisy or sparse data.

6. Conclusions

In this study, we compared three different recommendation techniques—Content-Based Filtering, Neural Networks, and LightGBM—to assess their effectiveness in predicting and recommending URLs. Each

model was evaluated using RMSE as the primary metric, which allowed us to quantify and compare performance across methods.

6.1. Content-Based Filtering:

The content-based filtering method successfully leveraged tag-based information to recommend URLs, producing satisfactory results in identifying similar items. However, there is potential for improving the model by exploring more sophisticated feature engineering techniques and incorporating advanced distance metrics.

6.2. Neural Networks:

The neural network model demonstrated reasonable performance, with the training process showing gradual loss reduction over epochs. Despite some overfitting towards the later stages of training, the model showed promise in learning complex patterns.

6.3. **LightGBM**:

The LightGBM model outperformed the other methods in terms of predictive accuracy, yielding an RMSE of 0.9757 after hyperparameter tuning. The feature importance analysis revealed that tags and save count played significant roles in making predictions. LightGBM proved to be an efficient tool for handling large datasets, especially when combined with feature engineering and optimization techniques.

Based on the results, LightGBM demonstrated the best performance with an RMSE of 0.9757, outperforming the neural network (RMSE: 0.9948) and providing valuable insights into feature importance, where tag-related features and `FirstSaveDate` were the most influential. The content-based filtering approach successfully generated relevant recommendations but lacked a quantitative evaluation metric. While the neural network showed potential, its fluctuating loss suggests room for improvement through hyperparameter tuning, architectural adjustments, or data augmentation. Overall, LightGBM stands out as the most effective and interpretable model for this dataset. However, combining content-based filtering for initial recommendations with LightGBM for ranking could enhance both accuracy and usability. Additionally, future work could explore hybrid models integrating collaborative filtering and content-based approaches or refine neural network architectures to develop more accurate and scalable recommendation systems. These insights provide valuable guidance for advancing recommendation system methodologies.

7. References

[1] M. K. Delimayanti et al., "Web-Based Movie Recommendation System using Content-Based Filtering and KNN Algorithm," *2022 9th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)*, Semarang, Indonesia, 2022, pp. 314-318.

[2] T. Badriyah, S. Azvy, W. Yuwono, and I. Syarif, "Recommendation system for property search using content based filtering method," *2018 International Conference on Information and Communications Technology (ICOIACT)*, Yogyakarta, Indonesia, 2018, pp. 25-29.

[3] G. Huang, "E-Commerce Intelligent Recommendation System Based on Deep Learning," *2022 IEEE Asia-Pacific Conference on Image Processing, Electronics and Computers (IPEC)*, Dalian, China, 2022, pp. 1154-1157.

[4] K. N. Muflih Hunna, F. Renaldi and I. Santikarama, "Paper Recommendation for Research References in Data Mining using Content-Based Filtering," *2022 International Conference on Science and*

Technology (ICOSTECH), Batam City, Indonesia, 2022, pp. 1-6.

[5] J. T. Anthony, G. E. Christian, V. Evanlim, H. Lucky and D. Suhartono, "The Utilization of Content Based Filtering for Spotify Music Recommendation," *2022 International Conference on Informatics Electrical and Electronics (ICIEE)*, Yogyakarta, Indonesia, 2022, pp. 1-4.

[6] A. N. K. Albayati and Y. Ortakci, "Recommendation Systems on Twitter Data for Marketing Purposes using Content-Based Filtering," *2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, Ankara, Turkey, 2022, pp. 1-5.