Classification of Urdu News Articles

Fiza Liaqat
Lahore University of Management Sciences
Lahore, Pakistan

Sameera Salman
Lahore University of Management Sciences
Lahore, Pakistan

Zahra Kazmi
Lahore University of Management Sciences
Lahore, Pakistan

Amna Hassan
Lahore University of Management Sciences
Lahore, Pakistan

Reeha Sajjad
Lahore University of Management Sciences
Lahore, Pakistan

Abstract

Our project addresses the significant lack of tools for classifying and personalizing Urdu news content, a gap that limits access to relevant information for Urdu-speaking audiences. We collected around 1800 articles from prominent Urdu news websites, categorized them into predefined segments, and subjected them to exploratory analysis to enhance their reliability and coherence. To classify the articles, we implemented and assessed the performance of three machine learning models and used a comparative analysis of their accuracies to identify the optimal approach for this task. The findings not only contribute to advancing natural language processing for Urdu but also lay the groundwork for personalized news delivery systems that better serve this underrepresented language.

1 Introduction

As the need for personalized digital content grows, we recognize that many widely spoken languages, including Urdu, remain underrepresented in personalized news systems. Urdu, despite being spoken by millions worldwide, remains underserved in terms of computational tools for content classification and personalization. This limitation restricts access to tailored news experiences for Urdu-speaking audiences. To bridge this gap, we undertook the development of a machine learning-based system aimed at categorizing Urdu news articles into distinct categories.

Our project involved collecting approximately 1,800 news articles from well-known Urdu news platforms, including Geo Urdu, Jang, Dunya News Urdu, and Express News. These articles were sorted into predefined categories: Entertainment, Business, Sports, Science-Technology, and International. To prepare the dataset for model training, we carried out comprehensive preprocessing and exploratory data analysis, ensuring data quality and uncovering meaningful patterns. We then implemented three different machine learning models—Naive Bayes, Neural Networks, and Random Forests—to perform the classification task. Each model was rigorously evaluated based on accuracy and other performance metrics, enabling us to determine the most effective solution.

By addressing the challenges of processing Urdu text, our project showcases how machine learning can be used in enhancing personalized content delivery for low-resource languages. Our findings contribute valuable insights into natural language processing and establish a foundation for developing inclusive digital platforms for Urdu-speaking users.

ML Research Project 2024, December 2024, Lahore, Pakistan
2024.

2 Methodology

2.1 Data Scraping

For the data collection phase of this project, we implemented a systematic web scraping process to gather Urdu news articles from four major online news platforms: Dunya News, Express.pk, Jang News, and Geo News. The scraping approach was tailored to efficiently extract category-specific articles, ensuring high-quality content for training and evaluation.

Data Sources: The platforms selected provide diverse Urdu-language news articles, categorized into entertainment, business, sports, world, and science-technology.

Web Scraping Framework: We used Python's `requests` library to fetch webpage content and `BeautifulSoup` for HTML parsing. The `pandas` library was employed to structure and store the scraped data.

Content Extraction: For each news platform, we defined the structure of target pages to locate article titles, links, and main content blocks. Only non-English text was retained to ensure focus on Urdu content. A regular expression (`re.sub`) was used to remove English characters, numbers, and special symbols from the extracted text.

Category and Pagination Handling: Each platform's URL structure was analyzed to identify category-specific pages. We iterated through these pages up to a predefined limit (e.g., 10 pages per category). Articles were mapped to their respective categories (*gold labels*) based on the originating URL paths or predefined labels.

Article Content Extraction: For each article, a secondary request was made to the article's page to extract its full text. Content was assembled from paragraphs found in HTML tags (`<p>` or `<div>`), depending on the platform's structure. Special cases, such as empty or inaccessible pages, were handled gracefully with error logging to ensure uninterrupted scraping.

Platform-Specific Adjustments: The scraping process was customized to cater to the unique structure of each platform. Articles were located and parsed as follows:

- **Dunya News:** Articles were located via `<a>` tags on category pages. Full URLs were constructed, and their content was parsed.
- **Express.pk:** Article links and headlines were extracted from `<div>` blocks within a news listing. Detailed content was retrieved from each article's page using `<p>` tags.
- **Jang News:** Articles were identified within categorized lists. Each article's full text was extracted from a `detail_view_content` block.

- **Geo News:** Articles were scraped from `<div>` blocks containing headline links. Full content was gathered from a `content-area` block on individual article pages.

Data Consolidation: After scraping, data from all platforms was combined into a unified `DataFrame`. Each record included:

- Unique identifier (`id`)
- Article title
- URL link
- Article content
- Corresponding category label (`gold_label`)

The consolidated dataset was exported as a CSV file with UTF-8 encoding for downstream processing.

2.2 Data Cleaning and Preprocessing

The data cleaning process involved several key steps to ensure the dataset was prepared for further analysis. Initially, a comprehensive list of Urdu stopwords from Kaggle was loaded from an external text file. This list was utilized during subsequent stages to filter out non-informative words from the text data.

To simplify the dataset and remove unnecessary elements, all digits, both Urdu and English, were eliminated from the text. Additionally, common Urdu diacritics were removed to standardize the representation of words. Punctuation marks and special characters were also identified and removed to clean the textual content further.

The dataset contained a mix of Urdu and English words. Using regular expressions, any English words were detected and removed to focus solely on Urdu text. Afterward, stopwords from the Urdu stopword list were filtered out to retain only meaningful words for analysis.

An analysis of the content length revealed the presence of outliers. These outliers were identified using the interquartile range (IQR) method and subsequently removed to ensure a consistent dataset free of extreme variations in content length.

2.3 Data Analysis and Visualization

After scraping and consolidating the dataset, several analyses were conducted to better understand the distribution and characteristics of the collected data. These analyses provided valuable insights into the data's quality and readiness for training machine learning models.

Label Distribution Analysis: The `gold_label` column, representing the predefined categories for each article, was normalized to lowercase to ensure consistency across the dataset. A count of articles per category was then computed to examine the distribution of the labels. The results indicated that the sports, entertainment, business, and world categories were well-balanced, each containing approximately 426–430 articles. However, the science-technology category had a significantly lower count, with only 100 articles. A visualization of the label distribution was created using a count plot to illustrate the relative proportions of each category, providing a clear view of the label balance.

Content Length Analysis: To assess the completeness and richness of the scraped articles, the length of each article was measured in terms of character count. A new column, `content_length`, was added to the dataset, capturing the number of characters in the content field for each article. Missing or empty content fields were handled by filling them with empty strings before calculating the lengths. A histogram with a Kernel Density Estimate (KDE) overlay was used to visualize the distribution of article lengths. This analysis revealed variations in content lengths, with most articles falling within a specific range. These insights were valuable for understanding the data quality and identifying potential preprocessing needs.

Since the data was scraped individually for each article, there were no missing values in the dataset. This ensured the integrity of the data and its suitability for further analysis and model training.

These analyses were essential for ensuring the dataset's readiness for machine learning. They highlighted imbalances in label distribution and variations in article lengths, which could influence model performance and guide subsequent preprocessing steps.

2.4 Model 1: Naive Bayes Classifier

This model classifies Urdu news articles into predefined categories using a bag-of-words representation and a custom implementation of a Naive Bayes classifier. Below is a detailed explanation of the steps involved:

Data Preprocessing: The preprocessing pipeline includes several important steps to prepare the data for classification:

- Tokenization: Each article's text (`cleaned_content`) is split into individual words.
- Vocabulary Construction: A set is created containing all unique words across the dataset, and a mapping (`vocab_to_index`) is built to assign each word a unique index.
- Encoding: Each article is converted into a vector representing word frequencies using the constructed vocabulary.

Feature Representation: The feature matrix and label array are as follows:

- **X:** A matrix where each row corresponds to the frequency vector of an article.
- **y:** A label array representing the article categories.

Train-Test Split: A custom `train_test_split` function divides the data into 80% training and 20% testing, ensuring random shuffling with a fixed seed (`random_state=30`).

Naive Bayes Classifier: The Naive Bayes classifier is implemented in two phases:

- **Training Phase (`fit` method):**
  - Computes the prior probabilities of each class (`P(class)`).
  - Estimates feature probabilities for each class using Laplace smoothing.
- **Prediction Phase (`predict` method):**
  - Computes log-probabilities for each class and selects the class with the highest score.

Evaluation Metrics: We evaluate the model using several key metrics:

- **Accuracy:** Proportion of correct predictions.
- **Precision:** $\frac{\text{TP}}{\text{TP+FP}}$ — how many predicted positives are actual positives.
- **Recall:** $\frac{\text{TP}}{\text{TP+FN}}$ — ability to detect all actual positives.
- **F1 Score:** Harmonic mean of precision and recall.

- **Misclassification Rate:** Proportion of incorrectly classified samples per class.

Model Performance: The model's performance is assessed as follows:

- **Accuracy:** The overall model performance on test data.
- **Precision, Recall, and F1 Score per Class:** These metrics highlight the model's strengths and weaknesses across different categories.
- **Confusion Matrix:** Displays the counts of true positives, false positives, and false negatives for each class, providing insight into common misclassifications.

Insights from Misclassified Samples: The script prints examples of misclassified articles, showing their true labels and predicted labels. This helps identify patterns in errors, such as vocabulary overlap between categories.

2.5 Model 2: Neural Networks

This model uses a simple feedforward neural network to classify Urdu news articles based on their text content. The implementation leverages PyTorch for constructing, training, and evaluating the model. Below is a breakdown of the steps involved:

Data Preprocessing: The preprocessing pipeline includes several key steps to prepare the data for classification:

- Tokenization: Each article (`cleaned_content`) is tokenized into words.
- Vocabulary Construction: A vocabulary of all unique words across the dataset is created, mapping each word to a unique index.
- Encoding: Each article is converted into a vector of word frequencies using the vocabulary.
- Label Encoding: The labels (`gold_label`) are transformed into numeric format using `LabelEncoder` from scikit-learn.
- Train-Test Split: Data is split into training (80%) and testing (20%) sets using a custom implementation of `train_test_split`.

Model Architecture: The Neural Network (NN) consists of the following components:

- **Input Layer:** The input dimension equals the vocabulary size (`input_size`).
- **Two Hidden Layers:** Each hidden layer has 128 units and uses ReLU activation for non-linearity.
- **Output Layer:** The output dimension equals the number of unique labels (`output_size`).

The model's output layer produces raw logits, which are unscaled scores for each possible category. Softmax is applied automatically when the cross-entropy loss function is computed, which combines softmax with the negative log-likelihood to output the model's loss for optimization.

Training: The training process involves the following steps:

- **Loss Function:** `CrossEntropyLoss` is used, as it is suitable for multi-class classification tasks.
- **Optimizer:** The `Adam Optimizer` is used with a learning rate of 0.002, adjusting the learning rate for each parameter to ensure faster and more reliable convergence.
- **Mini-Batch Training:** Data is processed in batches of size 32. For each batch:
  - Forward pass: Computes predictions.
  - Loss computation: Measures the difference between predictions and ground truth.
  - Backpropagation: Updates model parameters using gradients.
- **Epochs:** The model is trained for 10 epochs, with the loss printed at the end of each epoch.

Evaluation: The evaluation steps are as follows:

- **Forward Pass on Test Data:** Predictions are generated for the test set.
- **Accuracy:** The proportion of correct predictions is computed as the evaluation metric.

2.6 Model 3: Random Forest

This model is a custom implementation of a Random Forest classifier for classifying text data based on articles.

Data Preprocessing: The preprocessing pipeline includes the following steps:

- Data is loaded from a CSV file, and the target labels (`gold_label`) are encoded using `LabelEncoder`. Label encoding was chosen to convert categorical labels into a numerical format.
- The text data (`cleaned_content`) undergoes stemming using `PorterStemmer` to standardize words and reduce their dimensionality.

Feature Extraction: The text data is transformed into numerical features using `TfidfVectorizer`, selecting the top 1000 features to capture the importance of words within the context of the entire dataset.

Dataset Splitting: The dataset is split into training (64%), validation (16%), and testing (20%) sets to ensure a balanced amount of data for model training, validation, and testing.

Random Forest Implementation: The implementation is based on several key principles:

- **Entropy and Splitting:** Entropy impurity is used to measure disorder at each split, guiding the decision on the best feature to split on.
- **Tree Building:** A tree is built recursively, splitting until the maximum depth is reached or the number of samples falls below the minimum allowed. Each terminal node holds the most common class within that subset.
- **Training:** Multiple decision trees are trained on bootstrapped samples, each considering a subset of features to improve generalization and reduce overfitting.
- **Prediction:** Each tree makes a prediction, and the final prediction is determined by the class with the majority of votes across all trees.

Model Hyperparameters: The hyperparameters were chosen as follows:

- `n_trees` (20): The number of trees was set to 20 based on preliminary experiments showing that additional trees did not significantly improve performance but increased computational time.
- `max_depth` (30): This was chosen to prevent overfitting by limiting the depth of each tree.

- `min_size` (1): The minimum number of samples per leaf node was set to 1 to allow deeper splits where necessary.
- `n_features` (square root of the feature matrix size): This standard approach improves generalization by introducing randomness.

Model Training and Evaluation: The training and evaluation steps are as follows:

- **Training:** The Random Forest model is trained using the `fit` method, iteratively building decision trees on bootstrapped data.
- **Validation and Testing:** The model is evaluated on validation and test sets using the following metrics:
  - **Accuracy**: Proportion of correct predictions.
  - **Precision, Recall, and F1-Score**: Metrics calculated with `average='weighted'` to handle class imbalances.
  - **Confusion Matrix**: Provides insight into true positives, false positives, true negatives, and false negatives, revealing common misclassifications.

Classification Report: A detailed evaluation report is generated, displaying precision, recall, and F1-score per class. This report highlights the model's strengths and weaknesses across different categories.

3 Findings

In this study, we compared the performance of three text classification models: Naive Bayes (NB), Neural Networks (NN), and Random Forests (RF) for classifying Urdu news articles. The dataset consisted of around 1,800 articles from prominent Urdu news websites, categorized into Entertainment, Business, Sports, Science-Technology, and International. For feature representation, both Naive Bayes and Random Forest used TF-IDF, while the Neural Network utilized FastText word embeddings.

Naive Bayes Classifier (NB): The Naive Bayes model achieved an impressive test accuracy of 99%, making it the top-performing model in this study. This high accuracy can be attributed to the following factors:

- **Effective in High-Dimensional Spaces:** Naive Bayes is particularly suited for high-dimensional, sparse datasets, such as text data represented by TF-IDF. The simplicity of the model allows it to efficiently handle a large number of features without overfitting.
- **Independence Assumption:** While the assumption of conditional independence between features (words) may not always hold in real-world text data, Naive Bayes performed exceptionally well in this case, as the relationships between features were weakly correlated. This made the model highly effective in categorizing the articles, especially when the textual relationships weren't complex.
- **Efficiency:** Naive Bayes is computationally efficient, particularly when dealing with large-scale datasets. Its training time was significantly lower than that of more complex models like Neural Networks, making it a great choice for real-time classification in resource-constrained environments.

Neural Network (NN): The NN model, which used FastText word embeddings for feature representation, achieved a slightly lower accuracy than Naive Bayes (97.99%). The NN model's architecture consisted of two hidden layers with 128 neurons each, utilizing ReLU activation functions. Despite the high accuracy, the model's performance was not superior due to several factors:

- **Complexity:** Neural Networks are capable of modeling complex, non-linear relationships between features. However, in this case, the relatively simpler nature of the dataset (text data with weak feature correlations) did not fully leverage the strengths of the NN.
- **Contextual Understanding:** While the NN model, especially when using word embeddings, could capture some degree of context and semantic relationships between words, the performance improvement over Naive Bayes was marginal. This suggests that word embeddings did not provide a substantial advantage in this classification task, possibly due to the nature of the dataset and the categories involved.
- **Training Time:** The NN model required significantly more computational resources and training time, making it less efficient compared to Naive Bayes, especially when considering scalability.

Random Forest (RF): The Random Forest model, trained on TF-IDF features, performed well with an accuracy ranging between 94% and 96%. It outperformed NN in some cases but was still slightly behind Naive Bayes in terms of raw accuracy. The reasons for this include:

- **Robustness to Overfitting:** Random Forests are robust against overfitting, which made them effective in dealing with noisy data and feature variability. However, the model's ensemble approach (using multiple decision trees) didn't offer a significant boost in performance compared to Naive Bayes.
- **Feature Importance:** Random Forests work by evaluating feature importance and splitting decision trees based on information gain. While this can be advantageous for capturing complex relationships, it did not seem to outperform Naive Bayes when dealing with the simpler, high-dimensional feature space of TF-IDF representations.

4 Limitations

The impressive 99% accuracy we achieved demonstrates Naive Bayes' strength in sparse, high-dimensional datasets with weakly correlated features. However, its limitations arise from its simplicity and its reliance on straightforward feature relationships. While its efficiency and simplicity make it a practical choice for this task, it may not scale effectively to more semantically complex or context-dependent text data.

Limited Handling of Complex Relationships:

- While the independence assumption worked well here due to weakly correlated features, it may fail in datasets where strong dependencies exist between features (e.g., idiomatic expressions in text). This could limit its generalizability to more complex text datasets or languages with a richer syntactic structure.
- Naive Bayes cannot model interactions between features effectively. For example, it cannot understand the difference

between "not good" and "good" as it treats words independently. Additionally, words with multiple meanings (polysemy) or synonyms with similar meanings are treated as distinct features, potentially leading to a loss of semantic understanding and misclassifications, especially in nuanced text data.
- Naive Bayes' lack of contextual understanding limits its effectiveness in scenarios where the order or context of words is crucial, such as sarcasm, idioms, or detailed analytical text.

Class Imbalance Risks:

- If the dataset contains imbalanced class distributions, Naive Bayes might favor majority classes, leading to reduced recall or F1 scores for minority classes, despite achieving high overall accuracy.

Feature Engineering Dependency:

- Naive Bayes' success depends heavily on preprocessing and feature engineering. Effective tokenization, stopword removal, and appropriate smoothing techniques are essential for its performance. Any shortcomings in data preprocessing could negatively impact the results.

Performance on Rare Words or Classes:

- While Naive Bayes performs well on frequent words or classes, it may struggle with rare classes or low-frequency words, which could be crucial for certain predictions.

Overestimation of Probabilities:

- Due to its additive nature (especially with Laplace smoothing), Naive Bayes can overestimate the posterior probabilities for certain classes. The algorithm may become overly confident in its predictions, even with limited evidence.

5 Conclusion

In this study, we explored the performance of three machine learning models—Naive Bayes, Neural Networks, and Random Forests—for classifying Urdu news articles into predefined categories. Among these, Naive Bayes demonstrated exceptional performance with a test accuracy of 99%, proving to be highly effective in handling high-dimensional and sparse text data. Its efficiency and simplicity make it a strong candidate for real-time text classification tasks, particularly in resource-constrained environments. However, its reliance on the independence assumption and limited ability to model complex relationships restricts its scalability to more nuanced text datasets.

Neural Networks and Random Forests, while slightly less accurate, showcased their strengths in different areas. Neural Networks leveraged FastText word embeddings to capture some semantic and contextual information but were computationally expensive and provided only marginal improvement in handling complex text relationships. Random Forests proved robust against overfitting and handled feature variability effectively but did not outperform Naive Bayes in this specific task due to the simpler structure of the dataset.

These results highlight that the choice of model depends heavily on the dataset's characteristics and the task's requirements. For relatively straightforward classification tasks with weak feature correlations, Naive Bayes remains a practical and efficient choice. However, for datasets requiring contextual understanding, more sophisticated models like Neural Networks might be more suitable despite their higher computational costs. Neural Networks are very good at capturing semantic relationships and non-linear patterns in data, which is important for understanding nuanced text categories or languages with rich syntactic structures, much like Urdu.

Future work could focus on addressing the limitations observed in this study, such as improving class balance, incorporating advanced feature engineering techniques, and exploring ensemble methods to combine the strengths of multiple models. Additionally, expanding the dataset and incorporating richer syntactic and semantic representations could further enhance classification performance, especially for underrepresented categories and complex text data.

References

FastText Pre-trained Word Vectors. 2018. Available at: https://fasttext.cc/docs/en/crawl-vectors.html. Accessed: 2024-12-08.