Project Documentation: Blog Author Classification

by Raffael Brandt, Michael Maurer, Victor Nann, Kianusch Sezari and Friederike Weilbeer

Introduction

Blogs on the internet are a diverse collection of voices, opinions, and narratives. Bloggers express their thoughts, emotions, and experiences through writing, creating a dynamic and ever-evolving ecosystem. As the blogosphere continues to expand, it becomes increasingly important to understand and analyze the distinct characteristics of individual bloggers.

Research Problem

The amount of blog content available presents a challenge. It can be difficult to identify the unique traits and patterns associated with different blog authors. Traditional content analysis methods may not be sufficient for handling the scale and complexity of this task. Our research aims to address this issue by utilizing natural language processing (NLP) and machine learning techniques to explore the inherent characteristics of bloggers through their writing styles.

Project Goal

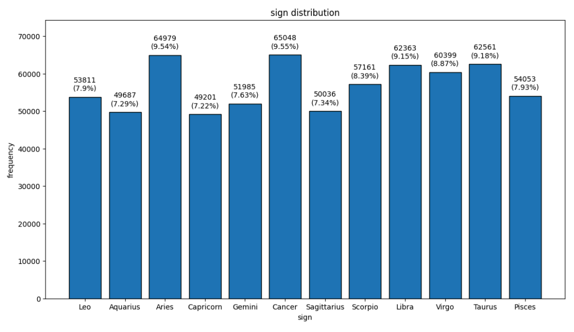
Our project aims to develop a text classification pipeline that can analyze the Blog Authorship Corpus and identify author-specific traits. We will train machine learning models on this dataset to create a tool that can classify and predict the authorship of blog posts.

The research aims to answer the following question: Can machine learning models accurately differentiate between various blog authors based on their writing styles? We considered age, gender and sign for this matter.

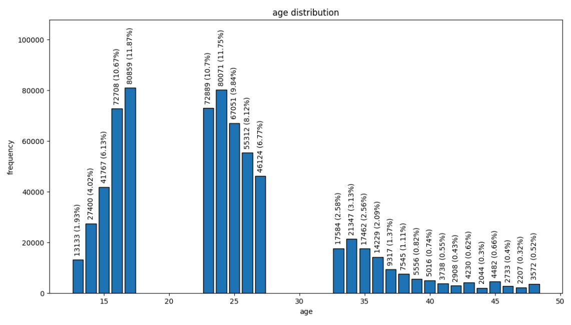
Our goal is to explore natural language processing, machine learning, and authorship attribution to gain a better understanding of the relationship between language and individual expression in the world of blogs.

Blog Authorship Corpus

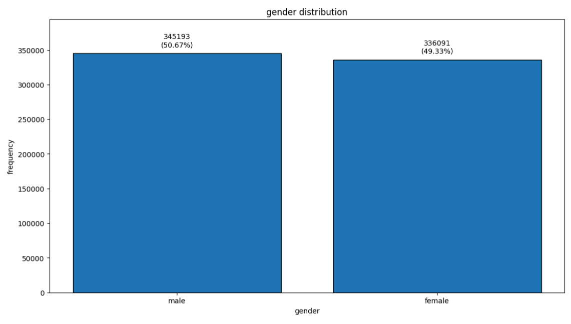
The Blog Authorship Corpus is a dataset of over 600,000 blog posts from more than 19,000 bloggers. The corpus was collected from blogger.com and contains over 140 million words. The dataset can be found on Kaggle and can be used for various natural language processing tasks. For that matter the corpus can also be used to train machine learning models that can identify the authors characteristics for a blog author classification.



Sign Distribution



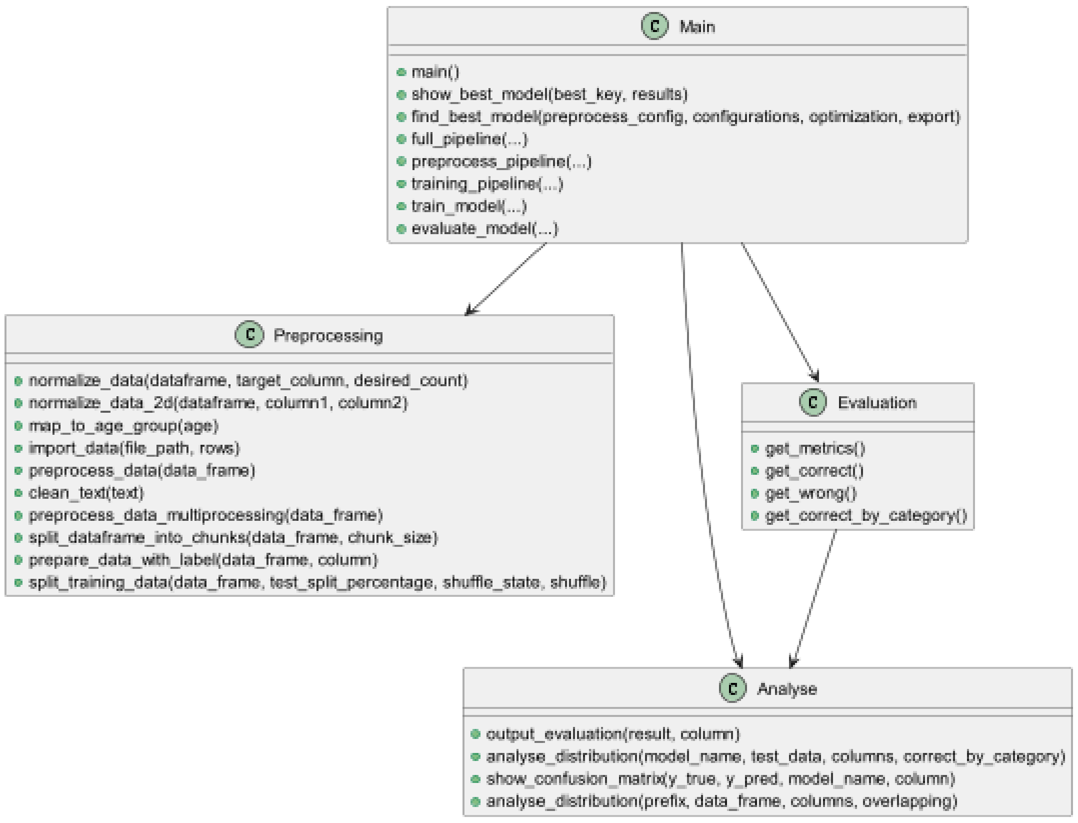
Age Distribution



Gender Distribution

The sign distribution ist relatively equally distributed the dataset analysis has shown. As for the age distribution the author can be separated into three age groups: <18, 23-27, and >32. The distribution of gender is equally distributed with 50.67% male and 49.33% female.

Project Structure


C Main
main()
show_best_model(best_key, results)
find_best_model(preprocess_config, configurations, optimization, export)
full_pipeline(...)
preprocess_pipeline(...)
training_pipeline(...)
train_model(...)
evaluate_model(...)
C Preprocessing
normalize_data(dataframe, target_column, desired_count)
normalize_data_2d(dataframe, column1, column2)
map_to_age_group(age)
import_data(file_path, rows)
preprocess_data(data_frame)
clean_text(text)
preprocess_data_multiprocessing(data_frame)
split_dataframe_into_chunks(data_frame, chunk_size)
prepare_data_with_label(data_frame, column)
split_training_data(data_frame, test_split_percentage, shuffle_state, shuffle)
C Evaluation
get_metrics()
get_correct()
get_wrong()
get_correct_by_category()
C Analyse
output_evaluation(result, column)
analyse_distribution(model_name, test_data, columns, correct_by_category)
show_confusion_matrix(y_true, y_pred, model_name, column)
analyse_distribution(prefix, data_frame, columns, overlapping)

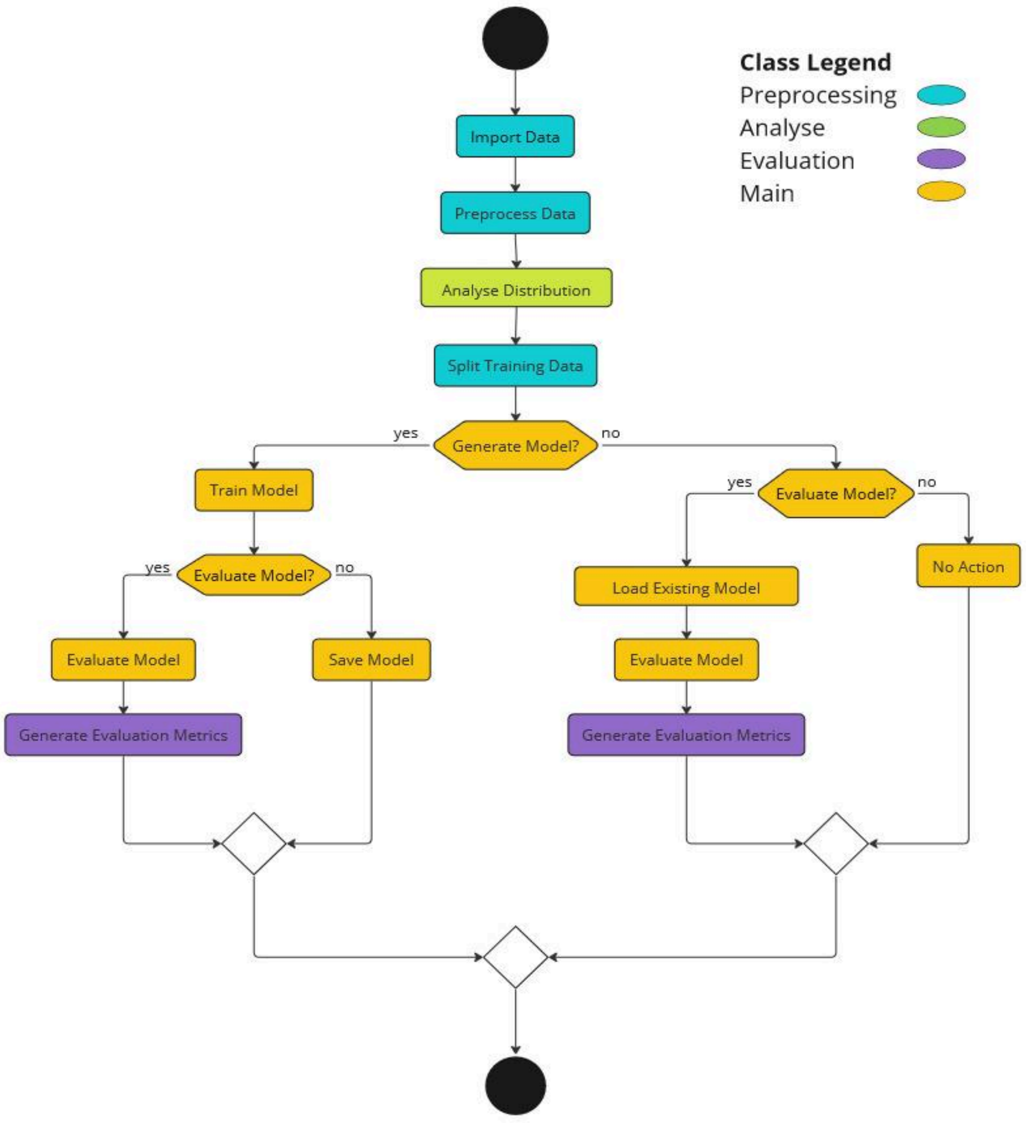
The diagram provides a simplified visual representation of the primary class structure within our project. The class diagram shows the main class, denoted as *main*, which orchestrates the entire workflow and interacts with three major components: the *Preprocessing class,* the *Analyse Class* and the *Evaluation class*.

The main function in the provided script serves as the entry point and orchestrates the execution of the entire machine learning pipeline.

1. **Preprocessing Module (Preprocessing.py):**
   - The main function interacts with the Preprocessing module to import and preprocess the dataset. It uses the import_data and preprocess_data_multiprocessing functions from this module.
   - The preprocess_pipeline function in main utilizes the prepare_data_with_label function from the Preprocessing module to prepare the data with labels.
2. **Utility Modules (Analyse, Logger, PrintColors, Evaluation, etc.):**
   - The main function utilizes various utility modules for logging (Logger), analyzing results (Analyse), and handling color-coded output (PrintColors).
   - The find_best_model function calls the output_evaluation function of the Analyse class to output evaluation results during the model search process.
3. **Training and Evaluation Pipeline (training_pipeline, train_model, evaluate_model):**
   - The main function encapsulates the entire training and evaluation pipeline. It calls the training_pipeline function, which, in turn, calls the train_model and evaluate_model functions.
   - The train_model function trains a machine learning model using scikit-learn components (LogisticRegression and TfidfVectorizer) and handles saving and loading the model.
   - The evaluate_model function evaluates the trained model using test data and provides insights like confusion matrices.
4. **Best Model Search (find_best_model):**
   - The main function calls the find_best_model function to search for the best model configuration among different parameter settings.
   - The find_best_model function iterates over various model configurations, calling the training_pipeline function for each. It compares models based on a specified comparison attribut .
5. **Full Pipeline (full_pipeline):**

- The full_pipeline function coordinates the entire pipeline from data preprocessing to model training. It calls the preprocess_pipeline and training_pipeline functions.

The flowchart below shows a simplified end-to-end pipeline for our machine learning project. It starts with importing data, then goes through preprocessing and splitting the data for training. Based on user preferences, the process either generates a new model or loads an existing one. If a new model is created, it is trained and optionally evaluated, and then comprehensive evaluation metrics are generated. If a model is already loaded, the pipeline skips training and proceeds to model evaluation.



Technologies and Libraries

1. **Core Libraries:**
    - **hashlib**: for hashing functions.
    - **time**: for time-related functions.
    - **re**: for regular expressions.
    - **os**: for interacting with the operating system.
2. **Data Analysis and Manipulation:**
    - **numpy**: for numerical operations.
    - **pandas**: for data manipulation and analysis.
3. **Data Visualization:**
    - **matplotlib.pyplot**: for creating visualizations.
    - **seaborn**: for statistical data visualization.
4. **Natural Language Processing (NLP):**
    - **nltk**: for natural language processing tasks.
        - **stopwords**: for handling common words.
        - **word_tokenize**: for tokenizing words.
        - **FreqDist**: for frequency distribution analysis.
        - Downloading NLTK data for stopwords and punkt tokenizer.
5. **Machine Learning:**
    - **scikit-learn (sklearn)**: for machine learning algorithms and tools.
        - **TfidfVectorizer** and **CountVectorizer**: for text vectorization.
        - **LogisticRegression**: for logistic regression.
        - **train_test_split**: for splitting data into training and testing sets.
        - **make_pipeline**: for constructing pipelines of estimators.
        - **confusion_matrix**, **recall_score**, **precision_score**, **accuracy_score**, **f1_score**: for evaluating classification models.
        - **resample** from **sklearn.utils**: for data resampling.
6. **Parallel Processing:**
    - **multiprocessing**: for parallel processing.
7. **Web Development (Flask):**
    - **Flask**: for creating web applications.

   - **render_template**, **url_for**, **request**, **redirect**: for handling web requests and rendering templates.
8. **Additional Utility:**
   - **joblib**: for parallel processing and handling long-running tasks.

User Interface

Our project's frontend is powered by a Flask web application that offers a user-friendly interface for interacting with our machine learning models. The application has two main routes: the home route ('/') and the processing route ('/process').
The home route ('/') renders the main page of the application, displaying an HTML template (index.html). Users can input raw text and choose a machine learning model from the provided options gender, age and sign.
The processing route ('/process') is responsible for processing the user's input. The application processes user input data with the selected machine learning model. The backend loads the pre-trained model, cleans the input text, and generates predictions. If the selected model file is not found, an error message is displayed.
The Flask framework is used to connect the frontend and backend components seamlessly. The web application allows users to interact with machine learning models in real-time. It provides predictions and associated probabilities, making the models more accessible and usable for end-users. Our machine learning pipeline is practically deployed in a web-based environment.

Evaluation

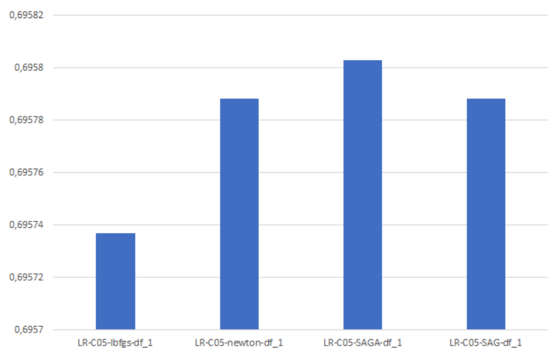
We used multiple approaches to improve the performance of both the program and the different models. One aspect utilized to accelerate the execution of the code is the implementation of multiprocessing when transforming the input data to become usable. This only makes sense if the data has a certain size, since otherwise the program execution will actually slow down because of the pre-required distribution to different threads. Since our input data is sizeable, this process leads to an improvement in runtime.

To make re-runs of the program faster, models that are finished training are saved and marked by both the column they were trained on (e.g. “age” or “gender”) as well as a hash compiling the settings used to train the model. Thereby the model doesn’t have to be retrained every single time the program is run. This process is further discussed in the previous chapter about the Project Structure. Finished models are also saved as a “.joblib”-file, and can be exchanged between users, thereby making the usage of the frontend-functionalities without previous training of models possible.

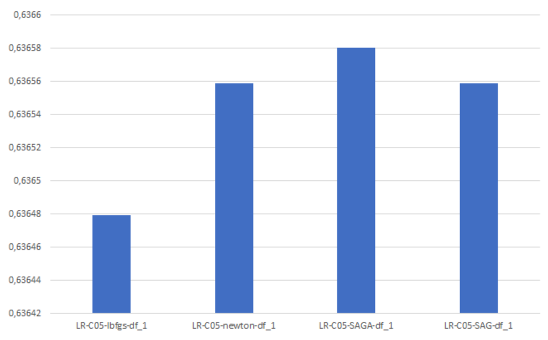
Since evaluation values are immediately calculated after the training of individual models, this makes their comparison much easier. After the execution of the program is finished, the terminal will state the best model with its configuration, as well as mark it in the color green to improve user friendliness and readability. The next step to use this resulting model is to copy the configuration data from the terminal and include it in the code. Please note that in order to use this functionality, the configurations of the different models (that you wish to compare) must be specified in the "configurations" function and will then be automatically compared.

These evaluation values include the usual metrics of accuracy, precision, recall and F1-score. In addition, the simplistic metric of "correct" and "incorrect" estimates is also displayed in the terminal after execution, to allow the user to make very rudimentary comparisons at first glance.

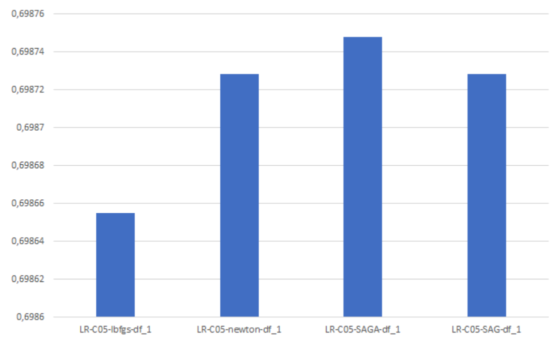
Our final scores and additional evaluation data are shown in the graphs below.



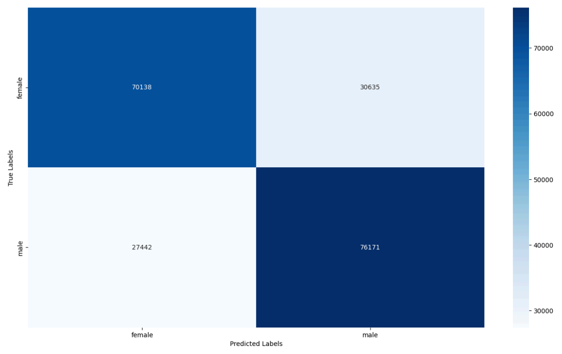
Accuracy of base models



F1-Scores of base models



Precision of base models



Confusion matrix of base models

AGE MODELS

	Recall	**Precision**	**F1**	**Accuracy**
LR-C05-SAGA-df_5-09_sublin-max_200000	**0,63089**	**0,70826**	**0,64739**	**0,70412**
LR-C05-SAGA-df_5-09_sublin-max_no	0,63089	0,70826	0,64738	0,70412
LR-C05-SAGA-df_5-09_sublin-max_300000	0,63089	0,70824	0,64738	0,70411
LR-C05-SAGA-df_5-09_sublin	0,63045	0,70752	0,64693	0,70360
LR-C05-SAGA-df_5-09-sublin-max_100000	0,63042	0,70751	0,64690	0,70359
LR-C07-SAGA-df_5-09_sublin	0,63089	0,69997	0,64683	0,70090
LR-C06-SAGA-df_5-09_sublin	0,62830	0,70015	0,64422	0,69993
LR-C05-SAGA-df_5-08_sublin	0,62558	0,70093	0,64149	0,69899
LR-C05-SAGA-df_5_sublin	0,62555	0,70093	0,64145	0,69899
LR-C05-SAGA-df_5	0,62555	0,70094	0,64145	0,69899

LR-C05-SAGA-df_5-07_sublin	0,62555	0,70094	0,64145	0,69899
LR-C05-SAGA-df_15	0,62548	0,70069	0,64136	0,69891
LR-C05-SAGA-df_1	0,62548	0,70074	0,64136	0,69893
LR-C05-SAGA-df_20	0,62532	0,70063	0,64122	0,69876
LR-C04-SAGA-df_5-09_sublin	0,62167	0,70177	0,63750	0,69744
LR-C05-SAG-df_1	0,62093	0,69873	0,63656	0,69579
LR-C05-newton-df_1	0,62093	0,69873	0,63656	0,69579
LR-C05-lbfgs-df_1	0,62086	0,69865	0,63648	0,69574
LR-C05-SAGA-df_30	0,61901	0,69693	0,63456	0,69394
LR-C03-SAGA-df_5-09_sublin	0,61543	0,70168	0,63066	0,69470

GENDER MODELS

	Recall	Precision	F1	Accuracy
LR-C05-liblinear-ovr_	0,712448235	0,712872596	0,712468455	0,712734025
LR-C05-SAGA-df_5-09_sublin-max_no	**0,713681054**	**0,714195166**	**0,713694119**	**0,713996345**
LR-C05-SAGA-tfidf_base	0,710221631	0,710713152	0,710233021	0,710532303
Final result	**0,713681054**	**0,714195166**	**0,713694119**	**0,713996345**

SIGN MODELS

	Recall	Precision	F1	Accuracy
SIGN_Kiki	0,23771	0,24887	0,23852	0,24234
SIGN_Kiki-1	0,24120	0,25270	0,24208	0,24587
SIGN_Kiki-2	0,24754	0,25968	0,24852	0,25231
SIGN_Kiki-3	0,25375	0,26363	0,25488	0,25797
SIGN_Kiki-4	0,25375	0,26363	0,25488	0,25797
SIGN_Kiki-5	0,25556	0,26471	0,25672	0,25958
SIGN_Kiki-6	**0,25557**	**0,26472**	**0,25673**	**0,25959**
Final result	**0,255568259**	**0,264716111**	**0,256725384**	**0,259590333**

Configurations of the best models:

Age:

[TfidfVectorizer(max_df=0.9, max_features=200000, min_df=5, sublinear_tf=True), LogisticRegression(C=0.5, max_iter=1000, n_jobs=18, solver='saga')]

Gender:

TfidfVectorizer(max_df=0.9, min_df=5, sublinear_tf=True), LogisticRegression(C=0.5, max_iter=1000, n_jobs=18, solver='saga')

Sign:

[TfidfVectorizer(max_df=0.9, min_df=7, sublinear_tf=True), LogisticRegression(C=0.8, max_iter=1000, n_jobs=10, solver='saga', verbose=2)]

Final Results

We developed text classification models that analyze and predict blog authorship using the Blog Authorship Corpus. These models were successful. We also built an efficient user interface for interacting with the models, with the frontend being a Flask web application. We created a prediction interface and a game on the frontend. The game allows users to test their ability to guess the blogger's features better than the model(s).

The research question posed in the project goal was whether machine learning models can accurately differentiate between various blog authors based on their writing styles. Our best models have successfully predicted author features with F-1 scores of (Age: 0.64738, Gender: 0.71369, Sign: 0.25673). Therefore, we conclude that a machine learning model can effectively differentiate between authors. However, the model for astrological signs shows low predictive capability. This suggests a questionable correlation between a blogger's astrological sign and their writing style. Drawing meaningful inferences from certain abstract personal characteristics is challenging. The importance of context and tangible features in text classification tasks is underscored by this outcome.

Who did what?

Raffael Brandt - Framework
Michael Maurer - Preprocessing
Victor Nann - Evaluation
Kianusch Sezari - Analyse
Friederike Weilbeer - Frontend