Faculty of Engineering and Technology
Electrical and Computer Engineering Department

ENCS3340 (AI) Course Project Report

1st : Osama Zeidan
Student ID: 1210601
Section:1

2nd : Francis Miadi
Student ID: 1210100
Section:2

3rd : Ra'ad Shamieh
Student ID: 1210373
Section:1

***Abstract*—This project focuses on building a machine learning model for Arabic sentiment analysis, targeting commonly used Arabic dialects. The goal is to classify text into sentiment categories: positive, negative, or neutral. The project utilizes Python to implement and train models using algorithms such as decision trees, Naive Bayes, and other methods studied during the course.**

The text data is prepared through preprocessing techniques, including tokenization, stemming, and stopword removal, to ensure high-quality input for the analysis. The performance of the models is evaluated using metrics such as F-score, precision, and recall, which show how well they performed in identifying the correct sentiment. By focusing on Arabic dialects, this project addresses a unique challenge in natural language processing and contributes to improving sentiment analysis for Arabic text.

I. Introduction

Sentiment analysis is a field of natural language processing (NLP) that focuses on understanding and classifying the emotions or opinions expressed in text. It is widely used in various applications, such as customer feedback analysis, social media monitoring, and market research. In this project, we focus on Arabic sentiment analysis, which presents unique challenges due to the complexity of the language and its many dialects.

Arabic is one of the most widely spoken languages in the world, with many regional dialects that differ significantly in vocabulary and structure. These dialects are commonly used in informal communication, making it important to develop tools that can analyze them accurately. Unlike Modern Standard Arabic, dialectal Arabic lacks standardized grammar and spelling, which makes processing this type of text more difficult.

The goal of this project is to build a machine learning model that can classify Arabic text into one of three sentiment categories: positive, negative, or neutral (Mixed). Using Python, we trained models with supervised algorithms such as decision trees, Naive Bayes, and Neural Networks by applying preprocessing techniques to clean and prepare the data (Simple and Optimized Techniques). By evaluating the models and comparing their performance, we aim to contribute to the growing field of sentiment analysis for Arabic text and address the challenges posed by dialectal variations.

II. Literature Review

Sentiment analysis for Arabic text has become an important research area due to the language's complexity and its widespread use across various dialects. One of the key references for our project was a professional Arabic Sentiment Analysis model available on GitHub and Kaggle. This model is implemented as a Kaggle notebook, allowing users to experiment with its structure and functionality. It uses advanced preprocessing techniques and machine learning algorithms, achieving high accuracy in classifying text into positive, negative, or neutral sentiments.

The Kaggle project primarily focuses on short Arabic text, such as individual words or phrases, which are easier to analyze due to their reduced complexity. In contrast, our project worked with a dataset containing longer sentences. Analyzing longer sentences presents additional challenges, such as handling more complex grammatical structures and varying contextual meanings. This distinction in datasets influenced the design and performance of our models compared to the professional implementation.

While the professional model demonstrated higher accuracy, likely due to its focus on simpler data and advanced algorithms, we focused on building and testing our models using Python. By implementing algorithms like decision trees, Naive Bayes and Neural Networks, we gained a deeper understanding of the sentiment analysis process and the challenges associated with analyzing long Arabic text. Despite these challenges, our models produced reasonable results, though with lower accuracy compared to the professional model.

III. Methodology

The methodology for this project consisted of multiple steps to develop, evaluate, and deploy sentiment analysis models for Arabic reviews. Both Weka and Python libraries were utilized to implement and compare the performance of various machine learning algorithms, with a simple web interface created for testing and comparing models.

A. Dataset Description

The dataset used in this project is composed of 100,000 Arabic text reviews, each labeled with one of three sentiment categories: positive, negative, or neutral (mixed). The dataset contained longer reviews, making it challenging to extract meaningful patterns due to the complexity of the Arabic language and its dialects.

The dataset was processed and saved in ARFF format for compatibility with Weka, while Python libraries were used to handle raw CSV files during alternative implementations.

B. Preprocessing

To prepare the dataset for machine learning, the following preprocessing steps were applied:

- **Tokenization:** Breaking down sentences into individual words.
- **Stopword Removal:** Removing frequent Arabic words (e.g., "," "") that do not significantly contribute to sentiment.
- **Stemming:** Reducing words to their root forms to normalize input and account for morphological variations.
- **Encoding:** Converting Arabic text into numerical representations suitable for model training.

These steps ensured that the input data was standardized and free of noise, improving model performance.

1) Simple Preprocessing: The simple preprocessing method involves applying basic cleaning steps to prepare the text for machine learning models. This approach is lightweight and efficient, focusing on fundamental text normalization. The following steps were performed:

- **Punctuation Removal:** Eliminates special characters and symbols from the text.
- **Number Removal:** Removes all numerical digits to focus on the textual content.
- **Whitespace Normalization:** Replaces multiple spaces with a single space to standardize formatting.
- **Lowercasing:** Converts all text to lowercase to avoid inconsistencies due to case sensitivity.

While simple preprocessing is computationally efficient, it does not address linguistic complexities, making it less effective for nuanced text analysis.

2) Optimized Preprocessing: The optimized preprocessing method incorporates advanced techniques to address the specific challenges of Arabic text analysis. These steps include:

- **Diacritic Removal:** Removes diacritical marks such as Fatha, Damma, and Kasra to normalize words.
- **Character Normalization:** Standardizes characters by replacing variations with consistent forms, such as normalizing different types of the same letter.
- **Repetition Handling:** Reduces excessive repetition of characters to improve uniformity.
- **Stopword Removal:** Removes frequently used words that do not add meaningful context to the analysis.
- **Stemming:** Reduces words to their root forms for better morphological normalization.
- **Text Cleaning:** Removes punctuation, numbers, and normalizes whitespace for consistent input.

The optimized preprocessing method works well for formal Arabic (Fusha) because it focuses on standard rules and structures. However, it is not as effective for street language or dialects, which often have different words and grammar.

C. Model Selection

Several machine learning models were trained and evaluated on the dataset using both Weka and Python libraries, in addition, we tried to implement a model using the Linear SVM (Support Vector Machine) algorithm which is not included in our course outline, and this is to try using new algorithms to optimize accuracy measures.

- **Naive Bayes:** A probabilistic classifier that calculates the likelihood of each sentiment based on word occurrences.
- **Decision Tree:** A tree-based algorithm that splits data based on attribute values to classify reviews.
- **Artificial Neural Networks (Multilayer Perceptron):** A model that uses interconnected layers of nodes to capture complex relationships in the data.
- **Linear Support Vector Machine:** is a classification algorithm that aims to find the best hyperplane that separates data points into distinct classes in a linear manner.

D. Training and Testing

The dataset was divided into two subsets with varying proportions for training and testing. The models were evaluated using different configurations to observe the impact of data split sizes on their performance. In Weka, the models were trained using a 66% training and 34% testing split, while in Python, an 80% training and 20% testing split was used.

- **Training Set (66%, 80%):** The portion of the dataset used to train the models, enabling them to learn patterns and features.
- **Testing Set (34%, 20%):** The portion of the dataset used to evaluate the models' ability to generalize to unseen data.

Each model was trained on the training set using both Weka and Python, and predictions were made on the testing set to compare the implementations. To improve performance and accuracy, several techniques were employed. For instance, Randomized Search was applied to optimize the hyperparameters of the Decision Tree model. This method involves

randomly sampling a defined number of parameter combinations from a grid, including parameters such as maximum tree depth, minimum samples required for splitting, and the splitting criterion ("gini" or "entropy"). By leveraging cross-validation, this approach efficiently identified optimal parameter settings to enhance the model's accuracy without exhaustively testing all combinations.

E. Evaluation Metrics

The performance of the models was assessed using the following metrics:

- ***Accuracy:*** *The proportion of correctly classified reviews out of all reviews.*
- ***Precision:*** *The proportion of true positive predictions out of all positive predictions.*
- ***Recall:*** *The proportion of true positives identified out of all actual positives.*
- ***F-score:*** *The harmonic mean of precision and recall, providing a balanced evaluation metric.*
- ***Confusion Matrix:*** *A table summarizing true positives, false positives, true negatives, and false negatives for each sentiment category.*

F. Implementation Details

Weka: *The models were trained and tested using Weka's GUI and scripting tools. The ARFF file format facilitated the compatibility of Arabic reviews with Weka's machine learning framework. Additionally, an unsupervised filter called* ***StringToWordVector*** *was used to convert text data into numerical representations suitable for machine learning algorithms. This filter applied techniques like tokenization, stopword removal, and TF-IDF weighting, ensuring the text was prepared for effective analysis and training.*

G. Python Libraries

Models were also implemented using Python libraries such as Scikit-learn and TensorFlow for flexibility and comparison with Weka's results. The preprocessing steps and training were performed programmatically, allowing custom data handling.

- ***Scikit-Learn:*** *Used for implementing machine learning models and evaluating performance metrics such as accuracy, precision, and recall.*
- ***Pandas:*** *Utilized for data manipulation and cleaning, including handling datasets in CSV format and preparing them for analysis.*
- ***NLTK:*** *Applied for natural language processing tasks like tokenization, stopword removal, and stemming in Arabic text.*
- ***TensorFlow:*** *Used to build and train artificial neural networks, enabling the exploration of complex relationships in the data.*

H. Web Interface for Model Testing

A simple web page was developed to provide an interactive platform for testing the models and choosing between them. The web interface allows users to input Arabic text, which is then analyzed by the selected model to predict the sentiment. This deployment provided an easy way to compare the outputs of different models and evaluate their real-world performance.

The web page was built using ***HTML****,* ***CSS****, and* ***Python Flask*** *to integrate the trained models. Each model's name was displayed to assist in model selection, making the system user-friendly and practical for end-users.*

I. Results Comparison

The accuracy and performance of the models were evaluated and compared across both platforms (Weka and Python). Among the tested models, the Neural Network and SVM demonstrated the highest accuracy, followed by Naive Bayes and Decision Trees. Performance metrics, including accuracy, precision, recall, and F1-score, were analyzed to identify the strengths and limitations of each model and platform. Additionally, the models were tested on varying data sizes to assess their robustness and adaptability to different training and testing configurations.

Look at redAppendix to view the figures illustrating the results, including comparisons of accuracy, precision, recall, and F1-score for the evaluated models.

Algorithm	Accuracy	Precision	Recall	F1-Score	Tool	Preprocessing
DT	57.0%	57.0%	57.0%	57.0%	Python	Simple
DT	57.0%	52.0%	52.0%	52.0%	Python	Optimized
NB	63.5%	63.3%	63.5%	63.3%	Weka	STWV Filter
NB	63.0%	63.0%	63.0%	63.0%	Python	Simple
NB	63.0%	63.0%	63.0%	63.0%	Python	Optimized
ANN	65.0%	65.0%	65.0%	65.0%	Python	Simple
ANN	65.0%	65.0%	65.0%	65.0%	Python	Optimized
SVM	65.0%	65.0%	65.0%	65.0%	Python	Simple

TABLE I
PERFORMANCE COMPARISON OF ALGORITHMS WITH PREPROCESSING DETAILS

J. Conclusion

Our project focused on sentiment analysis of Arabic sentences in street-Arabic dialects, which presented significant challenges in training our models. To address these challenges, we attended the NLP lectures by Dr. Adnan Yahya at Birzeit University to gain a deeper understanding of Arabic language processing. We employed preprocessing techniques tailored to our dataset and experimented with multiple tools, including Weka and Python libraries, on datasets of varying sizes and with different preprocessing methods.

To evaluate the models comprehensively, we measured their accuracy and performance across diverse scenarios. Additionally, we implemented a simple user interface using Flask in Python to test the models with new inputs and simulate real-world applications.

Note that the figures of results included in the appendix are just the most important ones, we have additional findings that will be discussed in detail during the presentation.

Through this project, we gained valuable insights into the complexity and richness of the Arabic language, particularly in its dialectal variations. These learnings will significantly contribute to our ability to deal with similar challenges in future projects.

IV. References

itemsep=0.2em, leftmargin=1em, labelsep=0.3em

1) *Kaggle: magentahttps://www.kaggle.com/code/mksaad/sentiment-analysis-in-arabic-tweets-using-sklearn*
2) *GitHub: magentahttps://www.github.com/motazsaad/arabic-sentiment-analysis*
3) *Mazajak: magentahttps://www.research.ed.ac.uk/en/publications/mazajak-an-online-arabic-sentiment-analyser*

V. Appendix

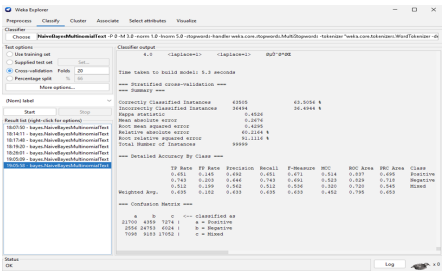

Fig. 1. NB Measures

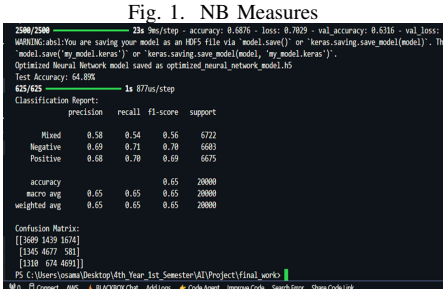

Fig. 2. ANN Measures

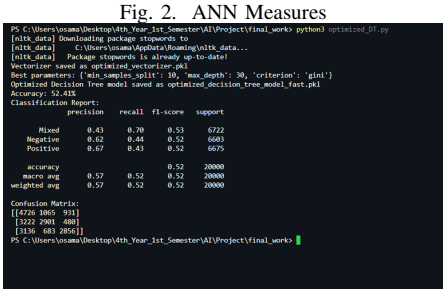

Fig. 3. DT Measures

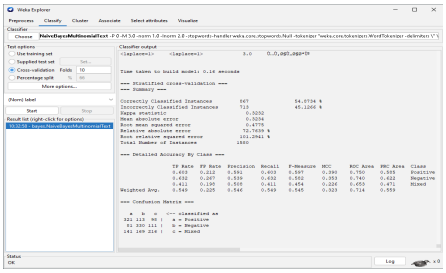

Fig. 4. NB Small Data Measures

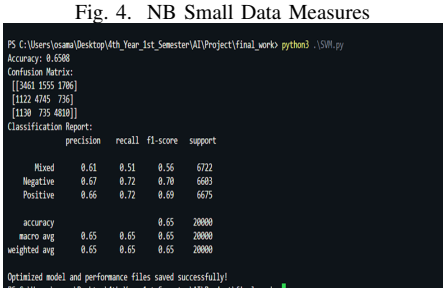

Fig. 5. SVM Measures

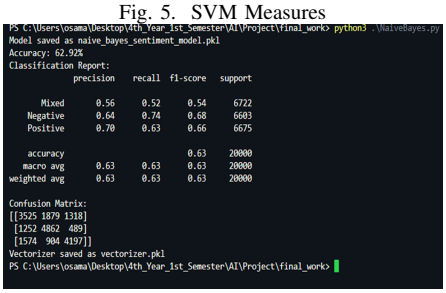

Fig. 6. NB Python Measures

```
# Step 6: Optimize Decision Tree Parameters Using Randomized Search
param_grid = {
    "max_depth": [10, 20, 30, None],
    "min_samples_split": [2, 5, 10],
    "criterion": ["gini", "entropy"],
}

random_search = RandomizedSearchCV(
    DecisionTreeClassifier(random_state=42),
    param_distributions=param_grid,
    n_iter=10,  # Number of random combinations to try
    cv=5,
    scoring="accuracy",
    n_jobs=-1,  # Use all CPU cores
    random_state=42,
)
```

Fig. 7. Randomized Search

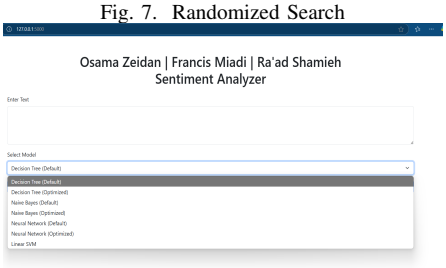

Fig. 8. Flask User Interface