Arabic Dialect Classifier

Agenda

- Problem definition.
- Solution Approach.
- Data fetching Phase.
- Preprocessing Phase.
- Training Phase.
- Deployment Phase.

Problem definition:

- There are many countries speak Arabic.
- Each country has its own dialect.
- There are many tasks that requires models to understand Arabic sentences.
- It will be a great addition if we used dialect as a feature.
- So we need aclassifier that is able to predict the dialect of a sentence.

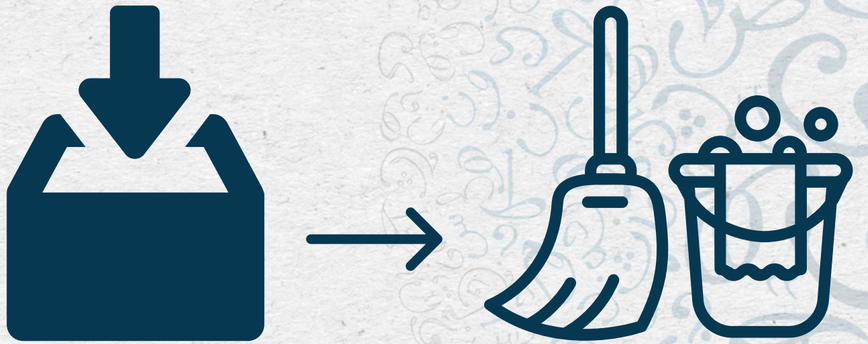
Solution Approach:

- Our solution will be devided into 4 main stages (phases):

Fetching the data

Solution Approach:

- Our solution will be devided into 4 main stages (phases):



Cleaning the data
(Preprocessing)

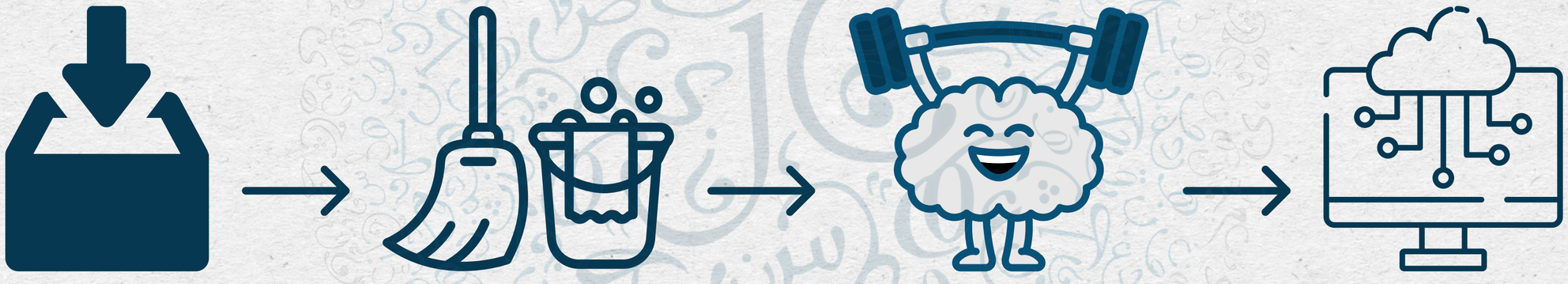
Solution Approach:

- Our solution will be devided into 4 main stages (phases):

Training

Solution Approach:

- Our solution will be devided into 4 main stages (phases):



Deployment

Data Fetching Phase:

- We have implemented a function to read tables from the database file using SQL query and then convert this data to a data frame：

```
def sql_to_df(path,query):
    con = sq3.Connection(path)
    observations = pd.read_sql(query, con)
    df = pd.DataFrame(observations)
    return df
```

Data Fetching Phase (cont'd):

- **We have two tables** id_dialect **and** id_text **which are labels and tweets respectively. we have used queries for each table like:**

```
select * from id_text;
```

- **We then saved data frames as CSV files**

Preprocessing Phase:

- We have implemented some functions that we will discuss it in the order they were used in:

  - Replace new lines:
    - some data had new lines and they needed to be replaced with a spaceto have one sentence and be able to use word tokenizer and split our sentences into words.

```
def replace_newlines(txt):
    return txt.replace('\n', ' ')
```

Preprocessing Phase (cont'd):

- We have implemented some functions that we will discuss it in the order they were used in:

  - Remove tags:
    - All the data had some tags that needed to be removed as they are from a different language and tell nothing about the dialect.

```
def remove_tag(txt):
    return re.sub(r'@\w+\s*', '', txt)
```

Preprocessing Phase (cont'd):

- We have implemented some functions that we will discuss it in the order they were used in:

  - Remove links:
    - some data had links to some websites that needed to be removed because they are in a different language.

```
def remove_links(txt):
    return re.sub(r'https?\S+\s*', '', txt)
```

Preprocessing Phase (cont'd):

- We have implemented some functions that we will discuss it in the order they were used in:

  - Remove English letters:
    - some data had links to some English letters that were non-sense to be used to classify the dialect.

```
def remove_english(txt):
    return re.sub(r'[a-zA-Z]+\s*', '', txt)
```

Preprocessing Phase (cont'd):

- We have implemented some functions that we will discuss it in the order they were used in:

  - Remove emojis:
    - some of the tweets had some emojis that can't help in our task.
    - we used 'emoji' library to help in this task.

```
def remove_emoji(txt):
    return emoji.replace_emoji(txt, '')
```

Preprocessing Phase (cont'd):

- We have implemented some functions that we will discuss it in the order they were used in:

  - Remove punctuation marks:
    - Even the underscore were removed because they were used to make old style emojis that may be harmful for our training :(

```
def remove_punctuation(txt):
    return re.sub(r'[^\w\s]|[_]', '', txt)
```

Preprocessing Phase (cont'd):

- We have implemented some functions that we will discuss it in the order they were used in:

  - Normalize laughter expressions:
    - In some tweets there were many laughter expressions {'ههههه','ههه, ...} that may vary in the length, so we normalized them into on expression which is 'هه'

```
def map_laughter(txt):
    return re.sub(r'(هه' ,'+ه(هه', txt)
```

Preprocessing Phase (cont'd):

- We have implemented some functions that we will discuss it in the order they were used in:

  - Remove repeated letters:
    - Repeated letters may be used to express Excitement {'حلووووو'}.
    - Arabic words may have two consecutive similar letters but never three, so we replaced any 3 or more repeated letters with only 2.

```
def remove_repeated_letters(txt):
    return re.sub(r'(.)\1{2,}', r'\1', txt)
```

Preprocessing Phase (cont'd):

- We have implemented some functions that we will discuss it in the order they were used in:

  - Remove numbers:
    - In our task numbers has no effect so we can remove it safely.

```
def remove_numbers(txt):
    return re.sub(r'\d+', '', txt)
```

Preprocessing Phase (cont'd):

- We have implemented some functions that we will discuss it in the order they were used in:

  - Character normalization:
    - We needed to normalize many characters to one form {ٱ,إ,ا,أ}.
    - We also neede to remove diacritics and elongation.
    - We used 'tashaphyne' library to help in that.

```
def normalize_arabic(txt):
    return normalize.normalize_searchtext(txt)
```

Preprocessing Phase (cont'd):

- We have implemented some functions that we will discuss it in the order they were used in:

  - Remove stop words:
    - There is some stop words that are common in all dialects and have no effect in our problem {'لو', 'يا', ...}

```
def remove_stop_words(txt, stop_words):
    return " ".join([word for word in
        word_tokenize(txt) if word not in
        stop_words])
```

Preprocessing Phase (cont'd):

- We have implemented some functions that we will discuss it in the order they were used in:

  - Remove repeated spaces:
    - Finally, the previous stages may removed some words leaving some spaces around them, so we needed to remove any consecutive spaces.

```
def remove_repeated_spaces(txt):
    return re.sub(r'\s{2,}', ' ', txt).strip()
```

ML Training Phase :

Feature Extraction

- TF-IDF Vectorizer: Converts text into feature vectors based on term importance.

- Mazaj: Arabic-specific word embedding model capturing semantic and syntactic information.

ML Training Phase :

Classification Models:

- LogisticRegression
- MultinomialNB
- LinearSVC
- Ridge
- RandomForestClassifier

ML Training Phase :

Preprocessing and Classification:

- Fit the pipeline on the training data and predict labels for the test set.
- Evaluated the performance of each classifier using classification metrics (accuracy, precision, recall, and F1-score) calculated with classification_report().
- Stored the results, including metrics and pipeline objects, in the results and pipeline_dict dictionaries.

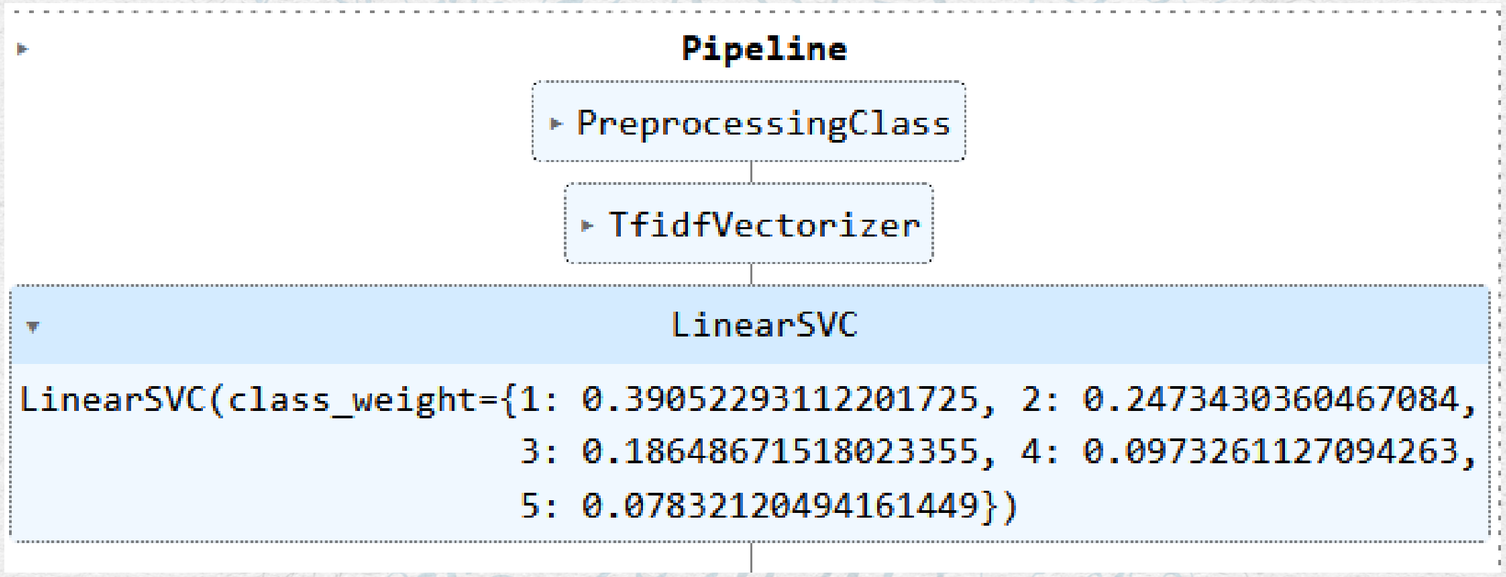
ML Training Phase :

Best Model : Linear SVC with TF-IDF

- The SVC classifier demonstrates an overall accuracy of 81%, indicating its effectiveness in correctly classifying instances.
- The macro average scores (Precision: 0.85, Recall: 0.72, F1-score: 0.77) indicate a balanced performance across all classes, considering class imbalance.
- The weighted average scores (Precision: 0.82, Recall: 0.81, F1-score: 0.80) demonstrate the classifier's ability to handle varying class sizes and provide reliable predictions.

ML Training Phase :

pipeline architechture



DL Training Phase :

- We fine-tuned two different pretrained models:
  - AraBERT
  - BERT-Base-Arabic-model

DL Training Phase (cont'd):

AraBERT Model:

- AraBERT sets new state-of-the-art for Arabic downstream tasks.
- Smaller than multilingual BERT by 300MB.
- Pre-trained on encoder with 12 Transformer blocks, hidden size of 768, and 12 self-attention heads.

DL Training Phase (cont'd):

AraBERT Model:

- Trained on large Arabic news corpus with 8.5M articles and 2.5B tokens.
- Two versions: Farasa-segmented and SentencePiece (BP).
- AraBERT with BP (which we fine-tuned) used for consistency with mBERT.

DL Training Phase (cont'd):

AraBERT Model:

- We encode train and test input using AraBERT tokenizer.
- We initiated datasets suitable for training by converting encoding to torch tensor and adding class weights to handle imbalanced labels.
- training is done on 2 epochs with 84% accuracy and 0.16 val loss.
- macro and weighted avg f1 scores are 0.81 and 0.84 respectively .

DL Training Phase (cont'd):

Bert-Base-Arabic Model:

- We also used the pre-trained BERT-Base-Arabic model, which is a variant of the BERT model and has 12 transformer layers with 768 hidden units and 110 million parameters. The model was trained on a large corpus of Arabic text.

DL Training Phase (cont'd):

Bert-Base-Arabic Model:

- The tokenizer used in this task is also part of the pre-trained BERT-base-arabic model and has a vocabulary of 32,000 subword units. It takes tweet as input and outputs a sequence of subword units used as input to the model.text.

DL Training Phase (cont'd):

bert-base-arabic Model:

The training process used:

- **Optimizer used: Adam**
- **Learning rate: 3e-5**
- **Batch size: 32**
- **Number of epochs: 2**
- **Loss function: Categorical Cross-Entropy**
- **Metric used for evaluation: Categorical Accuracy**
- **Test accuracy achieved: 83.18%**

Deployment Phase:

- We used Flask server as a development server to deploy our models:

Flask

Deployment Phase (cont'd):

- We have made two web pages.
  - One page is for getting the data from the user.

Deployment Phase (cont'd):

- Here we made an option to let the user choose between our two models, either Machine learning or Deep learning.

Deployment Phase (cont'd):

- We have made two web pages.
  - The other page is for showing the result.

Thank you