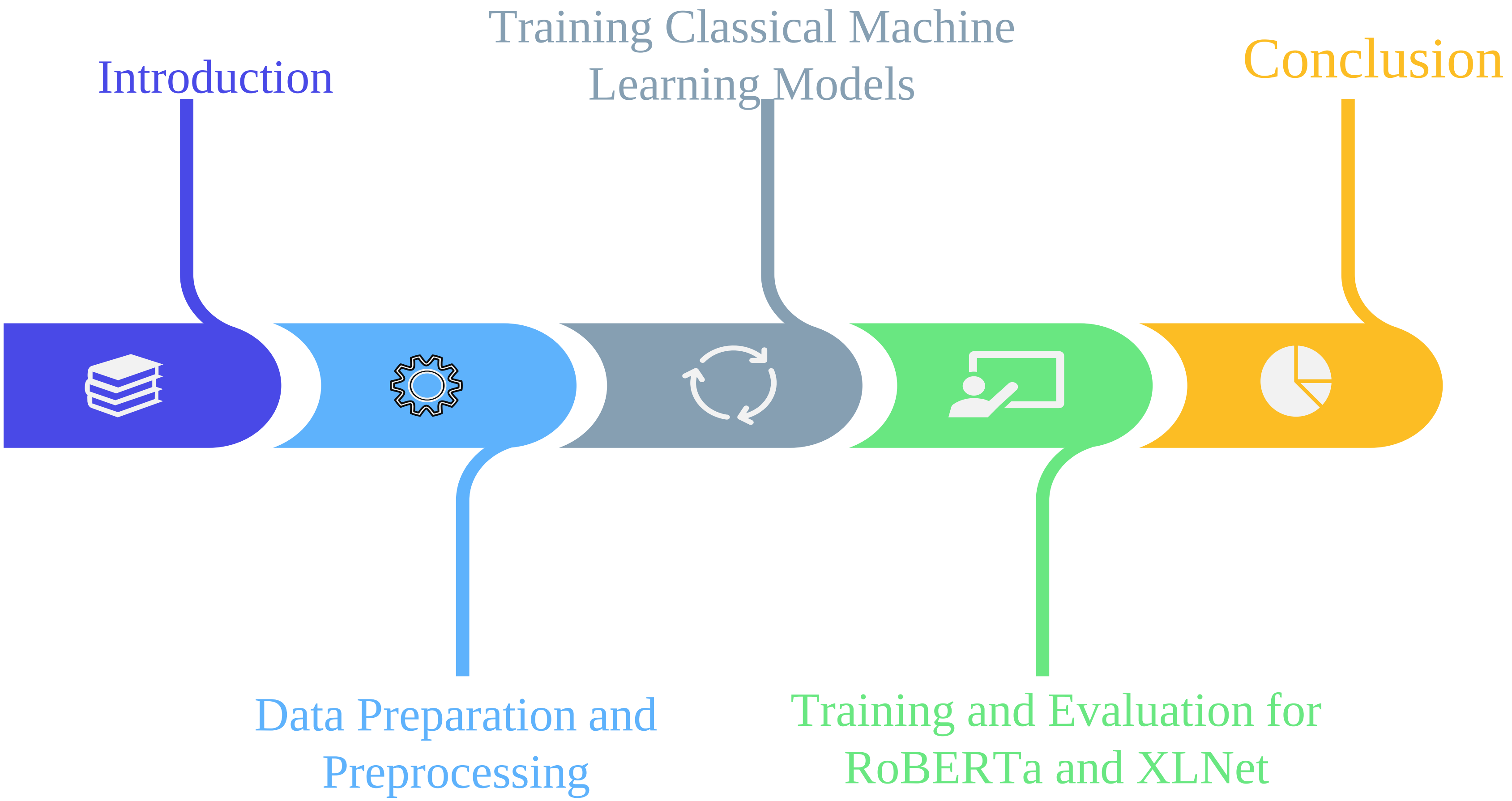
Customer Reviews Classification

Submitted by:
Abdellah Ennajari
Abdellatif Chakor
Yassine Ben Acha

Supervisor: Dr. Bentaleb Asmae
Jury:Dr. Alami Nabil

Academic Year: 2024/2025

Timeline


Introduction
Data Preparation and Preprocessing
Training Classical Machine Learning Models
Training and Evaluation for RoBERTa and XLNet
Conclusion

Data Preparation and Preprocessing

Data Preparation and Preprocessing :

The objective of this section is to perform essential data cleaning and preprocessing steps to prepare the customer review dataset for sentiment analysis.

Load the Dataset :

```
# LOAD THE DATASET :
df = pd.read_csv("../Data/raw/subset_data.csv")
# DISPLAY THE FIRST 5 ROWS :
df.head()
```

Dataset Overview :

```
# SHAPE OF THE DATASET :
print(f"Shape of the dataset : {df.shape}")
```

```
Shape of the dataset : (50000, 9)
```

Data Preparation and Preprocessing :

Dataset Overview :

```
# COLUMNS INFORMATIONS :
print("Column Count,Names and The data type (dtype) of each column :")
df.info()
```

```
Column Count,Names and The data type (dtype) of each column :
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   review_id    50000 non-null  object
 1   user_id      50000 non-null  object
 2   business_id  50000 non-null  object
 3   stars        50000 non-null  float64
 4   useful       50000 non-null  int64
 5   funny        50000 non-null  int64
 6   cool         50000 non-null  int64
 7   text         50000 non-null  object
 8   date         50000 non-null  object
dtypes: float64(1), int64(3), object(5)
memory usage: 3.4+ MB
```

```
# STATISTICS OF NUMERICAL COLUMNS :
df.describe()
```

	stars	useful	funny	cool
count	50000.000000	50000.000000	50000.000000	50000.000000
mean	3.848000	0.889540	0.250440	0.345060
std	1.350308	1.864481	0.941455	1.072388
min	1.000000	0.000000	0.000000	0.000000
25%	3.000000	0.000000	0.000000	0.000000
50%	4.000000	0.000000	0.000000	0.000000
75%	5.000000	1.000000	0.000000	0.000000
max	5.000000	91.000000	38.000000	49.000000

Handling Duplicated Values :

```python
# CHECKING FOR DUPLICATED VALUES :
df_duplicates = df.duplicated()
print(f"number of duplicated rows : {df_duplicates.sum()}")
```

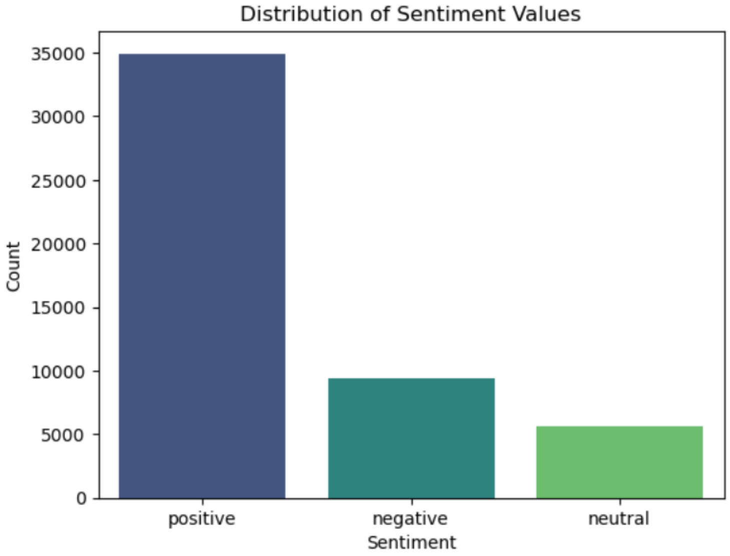
```
number of duplicated rows : 0
```

Target Variable Distribution :

```python
# ASSINING EACH STAR VALUE A SENTIMENT :
df.loc[df['stars'] == 3, 'sentiment'] = 'neutral'
df.loc[df['stars'] < 3, 'sentiment'] = 'negative'
df.loc[df['stars'] > 3, 'sentiment'] = 'positive'
```

Visualizing the distribution of sentiment :

```
# VISUALIZING THE DISTRIBUTION OF SENTIMENT VALUES :
sentiment = df_reviews['sentiment'].value_counts()

# Convert to a DataFrame for sns.barplot compatibility
sentiment_counts_df = sentiment.reset_index()
sentiment_counts_df.columns = ['sentiment', 'count']

# Create a bar plot for the distribution of sentiment values
sns.barplot(data=sentiment_counts_df, x='sentiment', y='count',hue='sentiment', palette="viridis",legend=False)

# Adding title and labels
plt.title('Distribution of Sentiment Values')
plt.xlabel('Sentiment')
plt.ylabel('Count')

# Show the plot
plt.show()
```



This distribution shows how customer opinions are spread across different sentiment categories, with **positive** reviews dominating, followed by **negative** reviews, and **neutral** reviews making up a smaller portion.

Text Preprocessing :

Before we start, we will create a new dataset with only the relevant columns for our analysis :

```python
# NEW DATASET :
df_reviews = df[['sentiment','text']]
```

Convert Text to Lowercase :

```python
# PREPERING TEXT :
df_reviews.loc[:,'text'] = df_reviews["text"].str.lower()
```

Remove Punctuations from Text :

```python
# Removal of Punctuations !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~
import string

PUNCT_TO_REMOVE = string.punctuation

def remove_punctuation(text):
    return text.translate(str.maketrans('', '', PUNCT_TO_REMOVE))

df_reviews.loc[:,"text"] = df_reviews["text"].apply(lambda text: remove_punctuation(text))
```

Remove Numbers from Text :

```
# Removal of Numbers
df_reviews.loc[:,'text'] = df_reviews['text'].str.replace(r'\d+', '', regex=True)
```

Handle newline and carriage return characters:

```
# Replace Newline and Carriage Return Characters
df_reviews.loc[:,'text'] = df_reviews['text'].str.replace('\n',' ', regex=True).str.replace('\r','', regex=True)
```

removing **stopwords** helps improve the quality of analysis by focusing on the words that hold the most **meanin** for **understanding** the text.such as "the", "is", "in", "and", "at" for english

```
# Removal of stopwords
STOPWORDS = set(stopwords.words('english'))
def remove_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in STOPWORDS])

df_reviews.loc[:, "text"] = df_reviews["text"].apply(lambda text: remove_stopwords(text))
```

Text Lemmatization :

Lemmatization is a text preprocessing technique that reduces words to their base or root form (called a "**lemma**"), it also considers the word's meaning and context.

We also have **stemming**, which simply chops off prefixes or suffixes to reduce words.

lemmatization is generally preferred for tasks like sentiment analysis where the meaning of the word is important.

```
# Lemmatization
lemmatizer = WordNetLemmatizer()

def lemmatize_text(text):
    return " ".join([lemmatizer.lemmatize(word) for word in text.split()])

df_reviews.loc[:,'text'] = df_reviews['text'].apply(lemmatize_text)
```

Text **tokenization** is the process of splitting a string of text into smaller units, typically words or phrases, called **tokens**. **Tokenization** breaks down text into individual words, making it easier to analyze and understand the meaning of each word in a review

```
# Tokenization
df_reviews.loc[:,'tokens'] = df_reviews['text'].apply(word_tokenize)
```

Relationship Between Sentiment and Length of Text :

The objective of analyzing the relationship between sentiment and text length is to understand whether the length of a review is related to its sentiment (positive, negative, or neutral).

```
# CREATING A NEW COLUMN IN THE DATASET FOR THE NUMBER OF WORDS IN THE REVIEW
df_reviews.loc[:, 'length'] = df_reviews['text'].apply(len)
```

Relationship Between Sentiment and Length of Text :

After visualizing the plot :

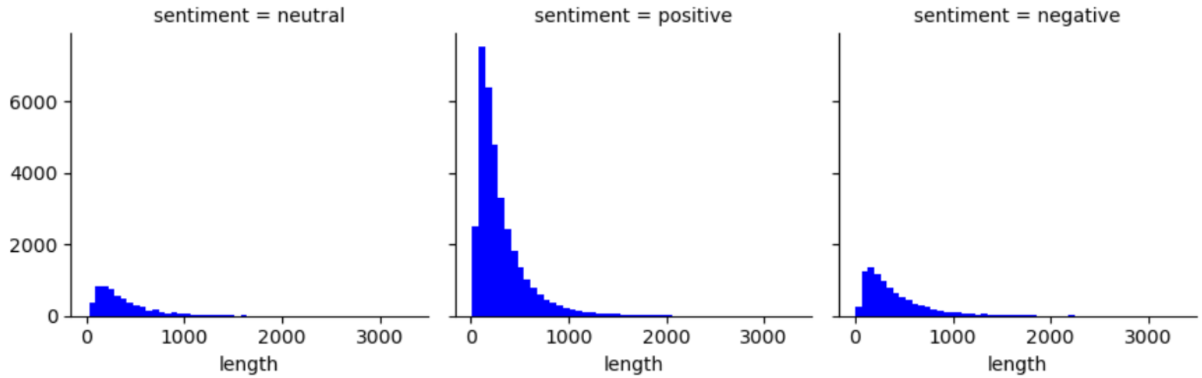
we observe that **positive reviews** tend to have longer text lengths, indicating that customers may express their satisfaction in more detail.

Negative reviews show a moderate length, suggesting that dissatisfied customers provide sufficient information but not as extensively as positive reviewers.

Neutral reviews, on the other hand, tend to have shorter texts, possibly reflecting more neutral or less detailed feedback.

```
# COMPARING TEXT LENGTH TO SENTIMENT
graph = sns.FacetGrid(data=df_reviews,col='sentiment')
graph.map(plt.hist,'length',bins=50,color='blue')
```

```
<seaborn.axisgrid.FacetGrid at 0x205342ea0f0>
```



Feature Encoding :

We will use **TF-IDF Vectorization (Term Frequency-Inverse Document Frequency)** to convert text data into numerical features.

- Why Choose **TF-IDF**?

. **TF-IDF** assigns higher weights to words that are important in a document.

. It transforms **text** data into a sparse **numerical matrix**, making it computationally efficient while retaining meaningful information.

. **TF-IDF** vectors integrate seamlessly with traditional **machine learning** algorithms.

- How **TF-IDF** Works ?

```
# Combine token lists back into a single string for each review
df_reviews.loc[:,'processed_text'] = df_reviews['tokens'].apply(lambda tokens: ' '.join(tokens))

# Initialize TF-IDF vectorizer
tfidf_vectorizer = TfidfVectorizer()

# Fit and transform the processed text to get the TF-IDF matrix
X_tfidf = tfidf_vectorizer.fit_transform(df_reviews['processed_text'])
print(f"the shape of the TF-IDF matrix :{X_tfidf.shape}")
```

```
the shape of the TF-IDF matrix :(50000, 62622)
```

Feature Encoding :

```python
# Initialize OrdinalEncoder
ordinal_encoder = OrdinalEncoder(categories=[['negative', 'neutral', 'positive']])

# Fit and transform the 'sentiment' column
y = ordinal_encoder.fit_transform(df_reviews[['sentiment']])

y = y.flatten()

# Check the result
print(f"the shape of target :{y.shape}")
```

```
the shape of target :(50000,)
```

For **sentiment** column we use **OrdinalEncoder** from sklearn.preprocessing with a predefined order for the sentiment categories **(negative, neutral, positive.**

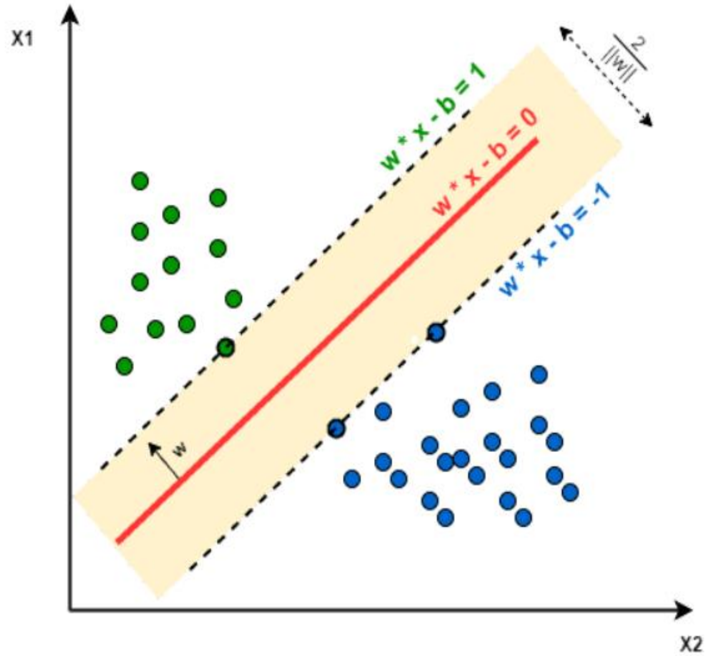
Data Splitting :

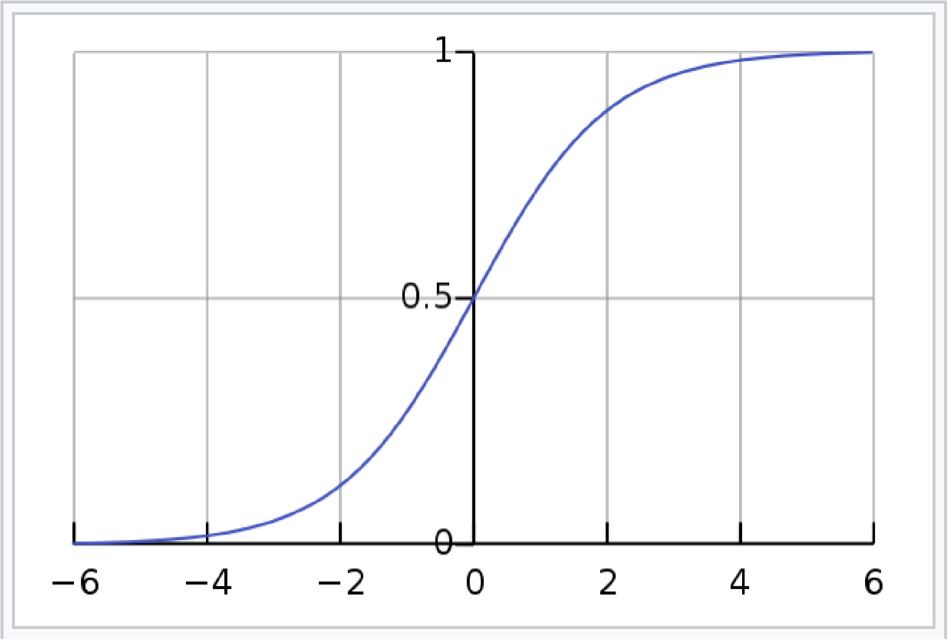
```
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.2, random_state=42, stratify=y)
```

ensuring the class distribution of the target variable is preserved **(stratify=y)** to handle its imbalance.

Training Classical Machine Learning Models

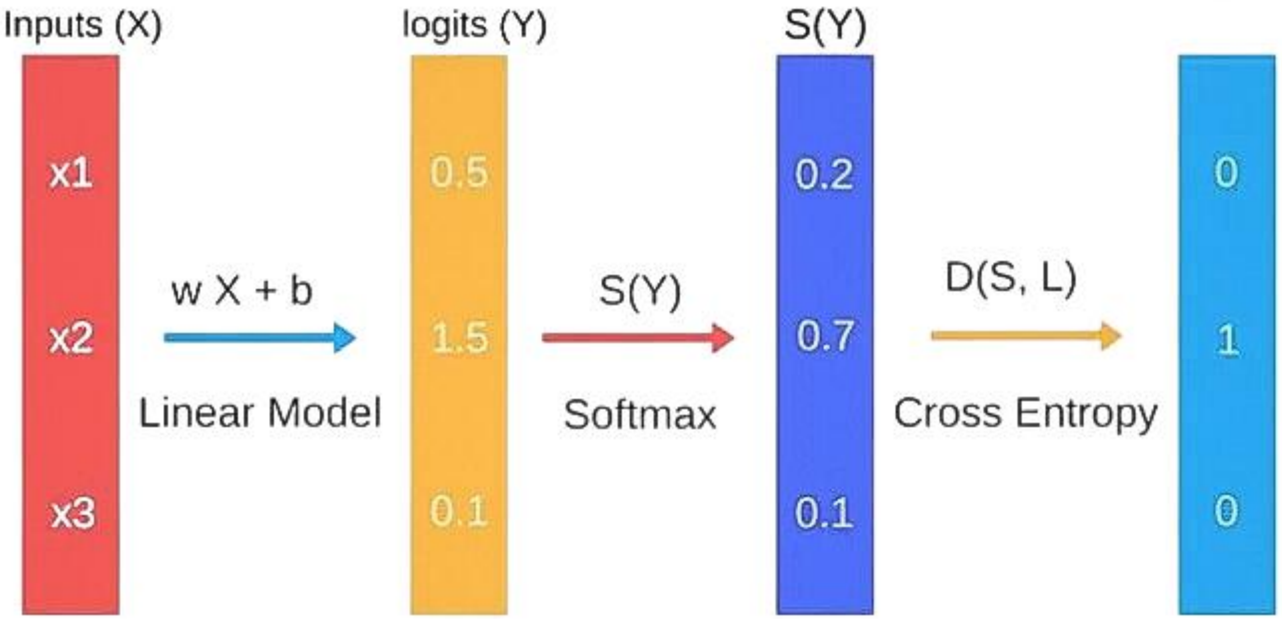
Logistic Regression & SVM





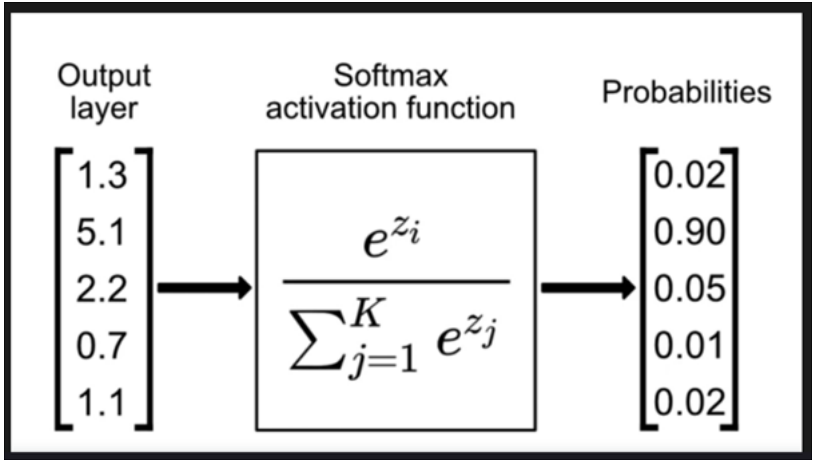
Before starting the next steps of our project, I will give a very brief introduction to our problem and its type.

Logistic Regression



Logistic Regression is a supervised machine learning algorithm used for classification tasks. It predicts the probability of a categorical dependent variable

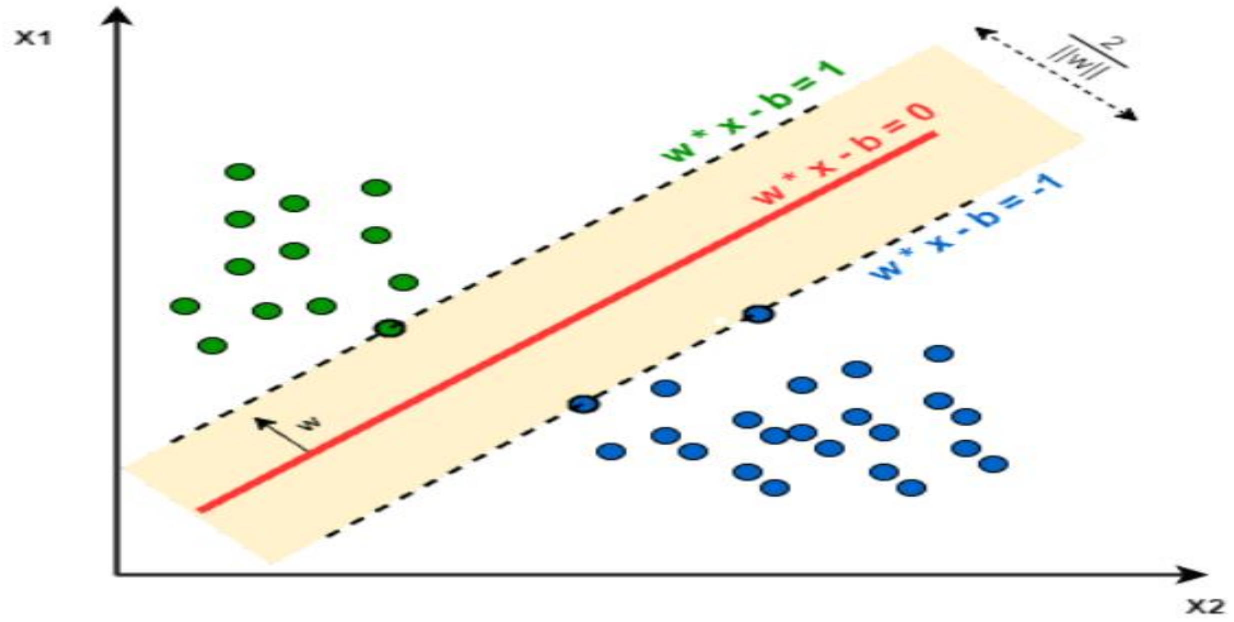
Softmax & Cross_entropy



$$logloss = -\frac{1}{N}\sum_{i}^{N}\sum_{j}^{M} y_{ij}\log(p_{ij})$$

- N is the number of rows
- M is the number of classes

Support Vector Machine



Support Vector Machine is a linear model for classification and regression problems. It can solve linear and non-linear problems

Code Breakdown

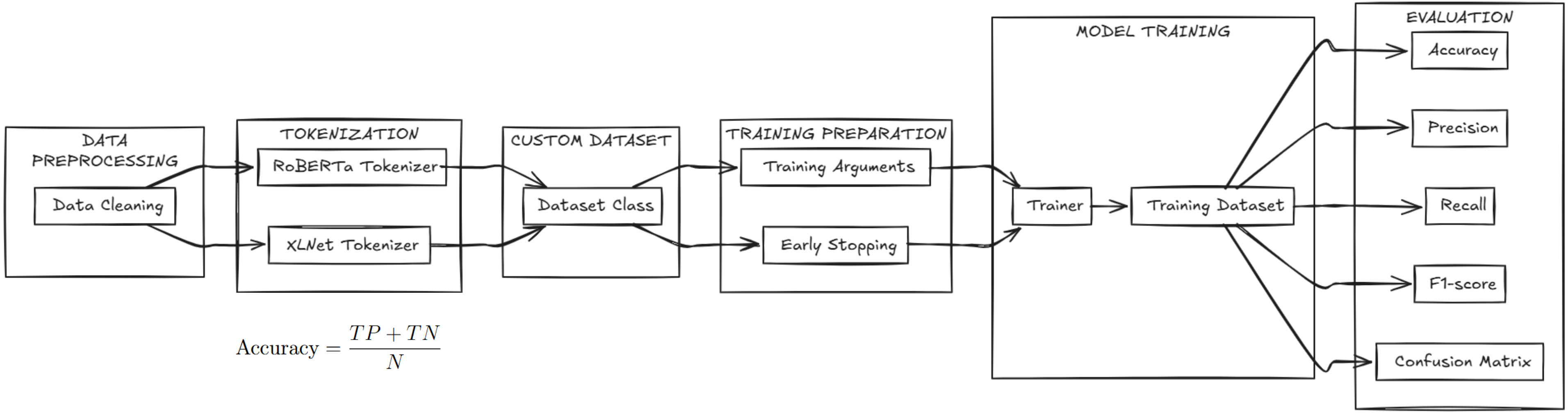
```
from sklearn.svm import SVC
```

```
# Entraîner le modèle
svm_model = SVC(probability=True)
svm_model.fit(X_train, y_train)
```

```
# Prédictions
y_pred = svm_model.predict(X_test)
y_pred_prob = svm_model.predict_proba(X_test)
```

Training and Evaluation of RoBERTa and XLNet

Transfer Learning Architecture



$$\text{Accuracy} = \frac{TP + TN}{N}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

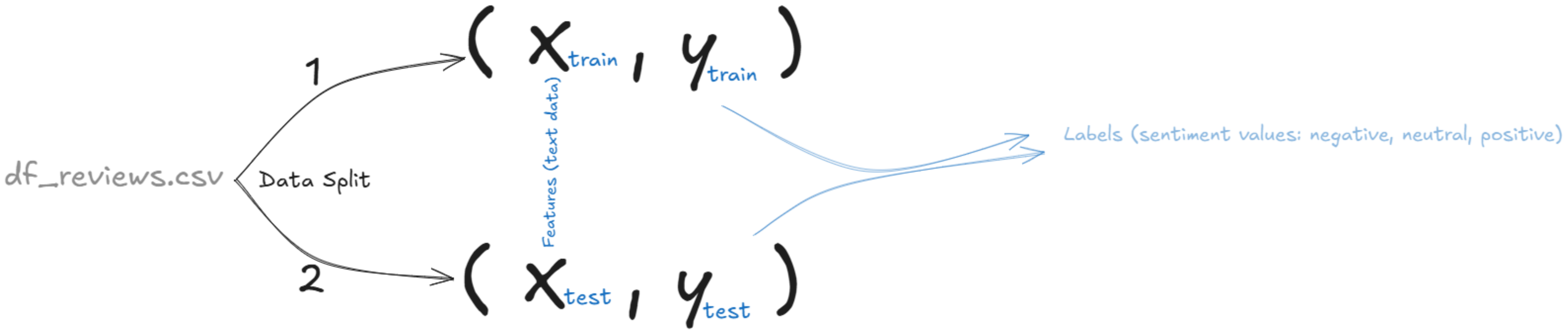
$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Data Split

df_reviews

```python
# Split the data
X_train, X_test, y_train, y_test = train_test_split(
    df_reviews['text'],
    df_reviews['sentiment'],
    test_size=0.2,
    random_state=42
)
```

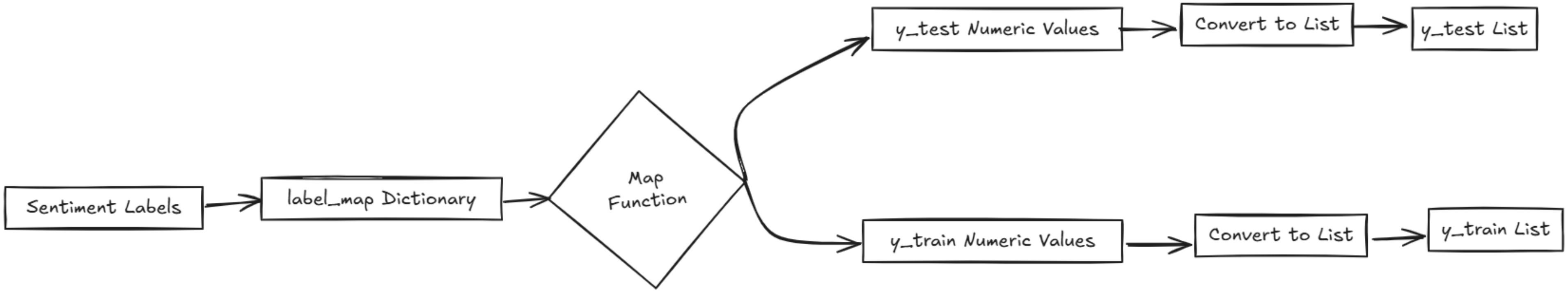
X_train, X_test, y_train, y_test



Map Sentiment

```python
# Map sentiment labels to numeric values
label_map = {'negative': 0, 'neutral': 1, 'positive': 2}
y_train_numeric = y_train.map(label_map).tolist()
y_test_numeric = y_test.map(label_map).tolist()
```

Label_map
y_train_numeric
y_test_numeric

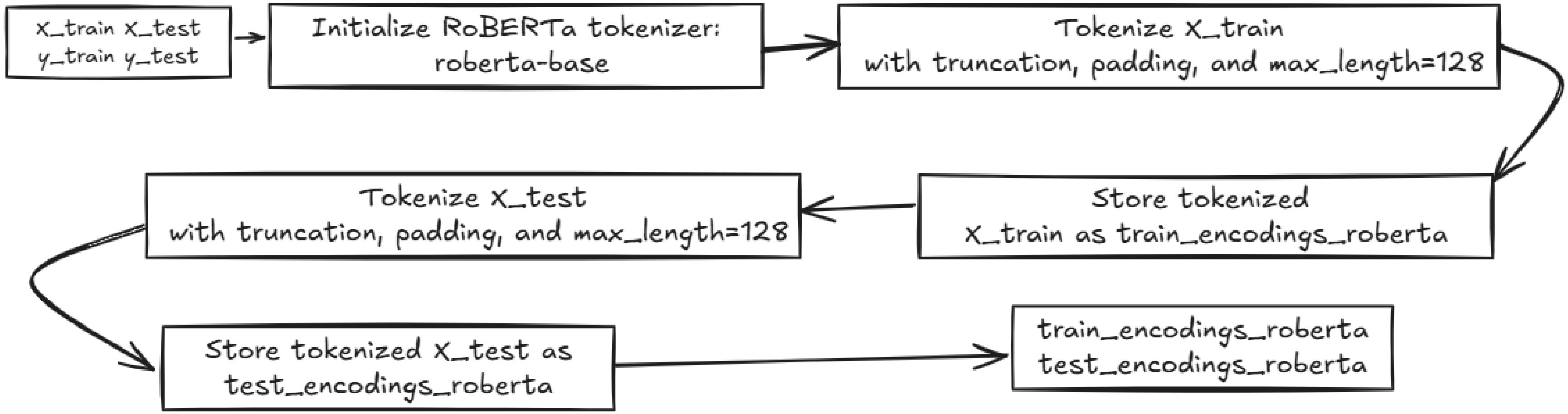


Tokenization

X_train
X_test
y_train
y_test

```python
# Tokenization for RoBERTa
tokenizer_roberta = RobertaTokenizer.from_pretrained('roberta-base')
train_encodings_roberta = tokenizer_roberta(
    X_train.tolist(), truncation=True, padding=True, max_length=128, return_tensors="pt"
)
test_encodings_roberta = tokenizer_roberta(
    X_test.tolist(), truncation=True, padding=True, max_length=128, return_tensors="pt"
)
```

train_encodings_xlnet
test_encodings_xlnet

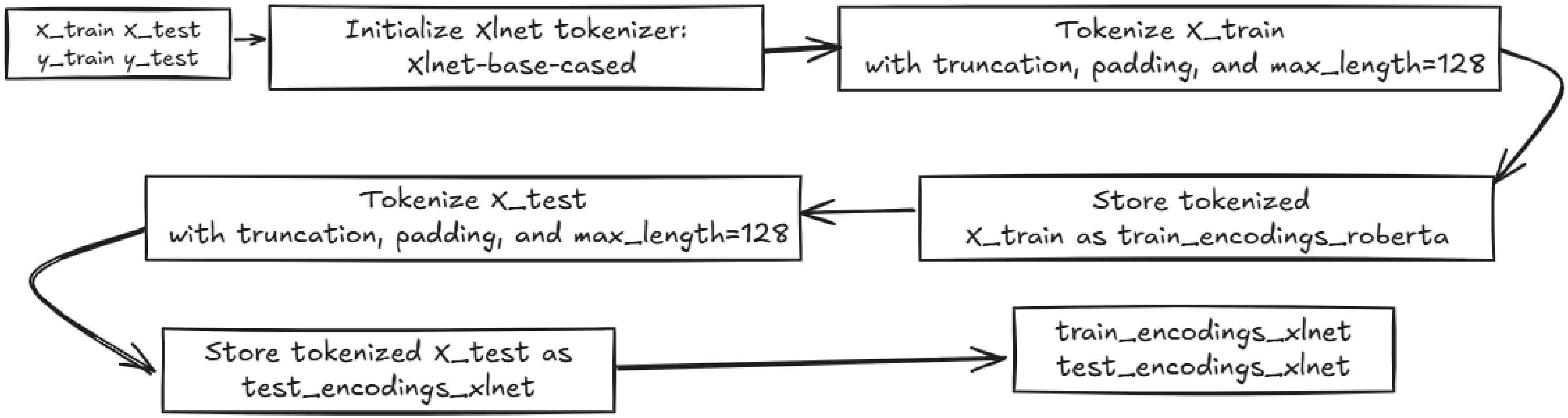


Tokenization

X_train
X_test
y_train
y_test

```
# Tokenization for XLNet
tokenizer_xlnet = XLNetTokenizer.from_pretrained('xlnet-base-cased')
train_encodings_xlnet = tokenizer_xlnet(
    X_train.tolist(), truncation=True, padding=True, max_length=128, return_tensors="pt"
)
test_encodings_xlnet = tokenizer_xlnet(
    X_test.tolist(), truncation=True, padding=True, max_length=128, return_tensors="pt"
)
```

train_encodings_xlnet
test_encodings_xlnet

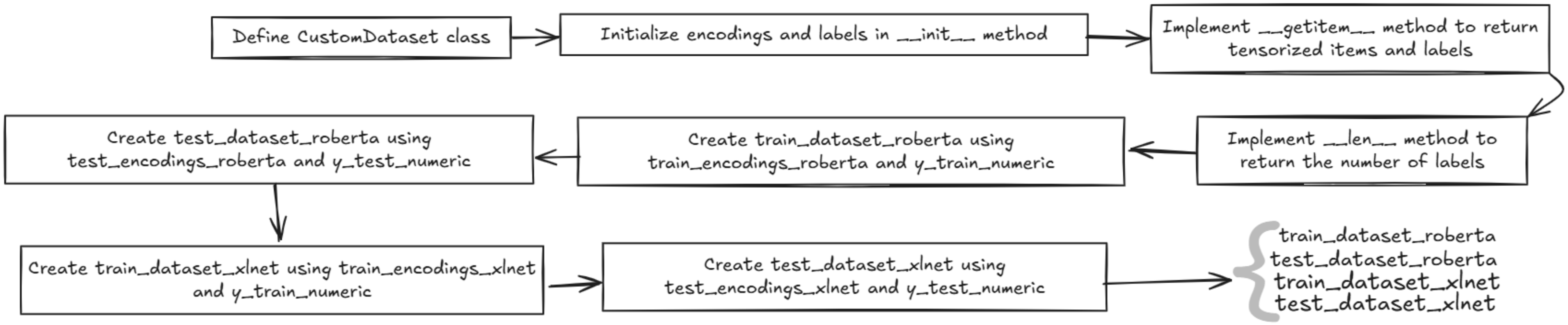


Custom Datase

train_encodings_xlnet
test_encodings_xlnet

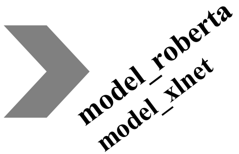
```python
# Define a custom dataset class
class CustomDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)
```

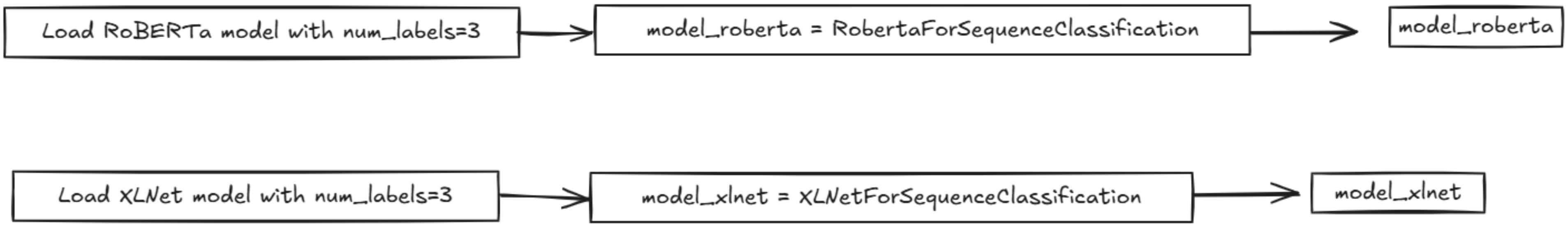
train_encodings_xlnet
test_encodings_xlnet



Define models

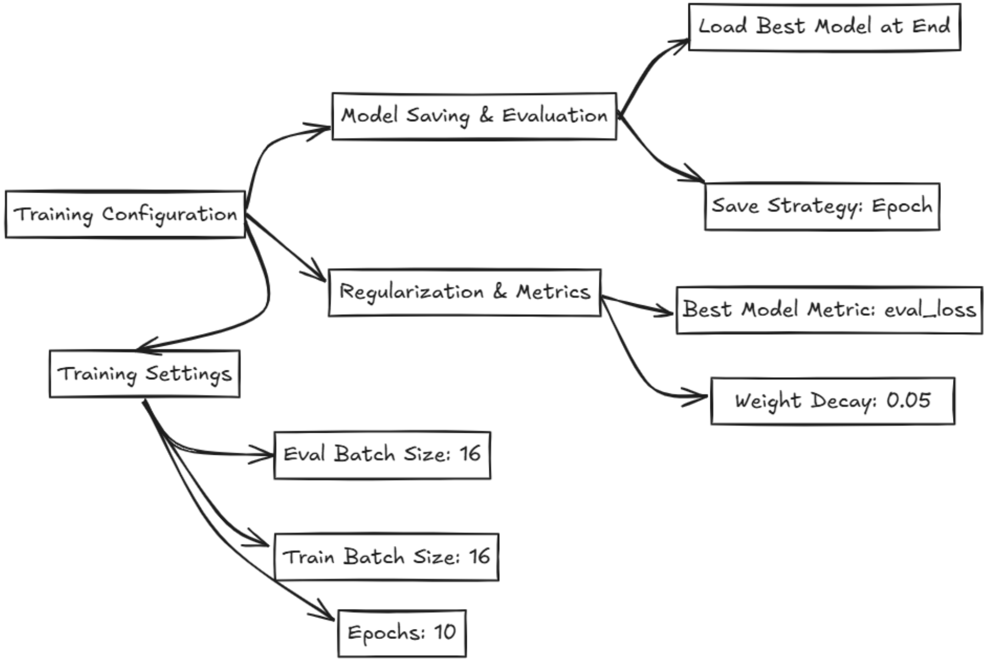
```python
# Define models
model_roberta = RobertaForSequenceClassification.from_pretrained('roberta-base', num_labels=3)
model_xlnet = XLNetForSequenceClassification.from_pretrained('xlnet-base-cased', num_labels=3)
```





Training Arguments

```python
# Training arguments with early stopping and regularization
training_args = TrainingArguments(
    output_dir='/content/drive/MyDrive/models',
    num_train_epochs=10,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    warmup_steps=500,
    weight_decay=0.05,
    logging_dir='/content/drive/MyDrive/logs',
    evaluation_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
    metric_for_best_model="eval_loss",
    greater_is_better=False,
    gradient_accumulation_steps=2,
    logging_steps=10,
    save_total_limit=2,
    report_to="none",
)
```



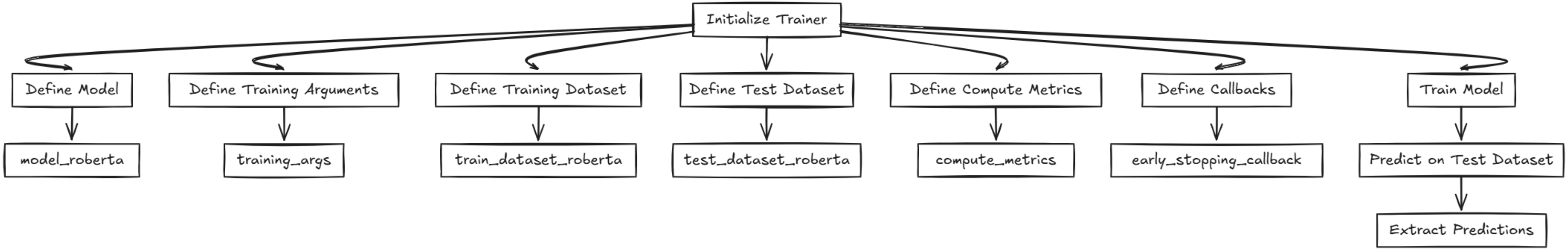
Trainer Model

```python
# Train and evaluate using RoBERTa
trainer_roberta = Trainer(
    model=model_roberta,
    args=training_args,
    train_dataset=train_dataset_roberta,
    eval_dataset=test_dataset_roberta,
    compute_metrics=compute_metrics,
    callbacks=[early_stopping_callback]
)
```

```python
trainer_roberta.train()
predictions_roberta = trainer_roberta.predict(test_dataset_roberta)
y_pred_roberta = np.argmax(predictions_roberta.predictions, axis=1)
```

[8750/12500 1:41:34 < 43:32, 1.44 it/s, Epoch 7/10]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.277800	0.456380	0.854700	0.833057	0.854700	0.834065
2	0.320900	0.365075	0.857200	0.860256	0.857200	0.858447
3	0.169300	0.395160	0.861100	0.853861	0.861100	0.856225
4	0.210100	0.453184	0.861500	0.859812	0.861500	0.860493
5	0.172900	0.520737	0.855200	0.863618	0.855200	0.857308
6	0.163200	0.609383	0.858900	0.861633	0.858900	0.860224
7	0.137900	0.771200	0.859800	0.862131	0.859800	0.860689



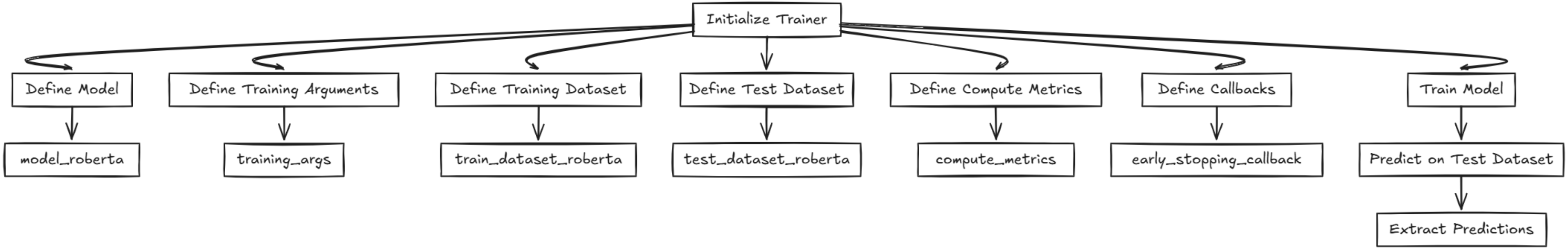
Trainer Model

```python
# Train and evaluate using XLNet
trainer_xlnet = Trainer(
    model=model_xlnet,
    args=training_args,
    train_dataset=train_dataset_xlnet,
    eval_dataset=test_dataset_xlnet,
    compute_metrics=compute_metrics,
    callbacks=[early_stopping_callback]
)
```

```python
trainer_xlnet.train()
predictions_xlnet = trainer_xlnet.predict(test_dataset_xlnet)
y_pred_xlnet = np.argmax(predictions_xlnet.predictions, axis=1)
```

[8750/12500 2:18:29 < 59:21, 1.05 it/s, Epoch 7/10]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.311800	0.411099	0.853700	0.827353	0.853700	0.827457
2	0.322100	0.375768	0.856800	0.854428	0.856800	0.855095
3	0.224500	0.451623	0.854600	0.840628	0.854600	0.844280
4	0.116600	0.492462	0.856500	0.858611	0.856500	0.856312
5	0.161400	0.532378	0.858300	0.848452	0.858300	0.851994
6	0.154900	0.763616	0.852000	0.853786	0.852000	0.852870
7	0.119800	0.890975	0.849600	0.853505	0.849600	0.851453



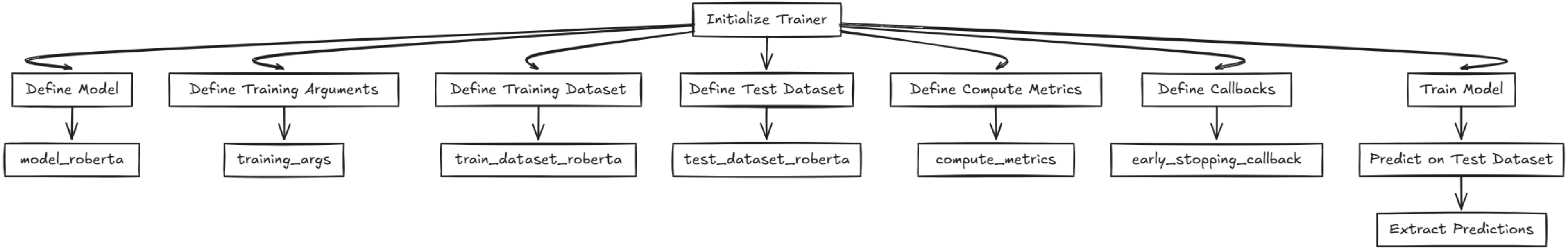
Trainer Model

```python
# Train and evaluate using XLNet
trainer_xlnet = Trainer(
    model=model_xlnet,
    args=training_args,
    train_dataset=train_dataset_xlnet,
    eval_dataset=test_dataset_xlnet,
    compute_metrics=compute_metrics,
    callbacks=[early_stopping_callback]
)
```

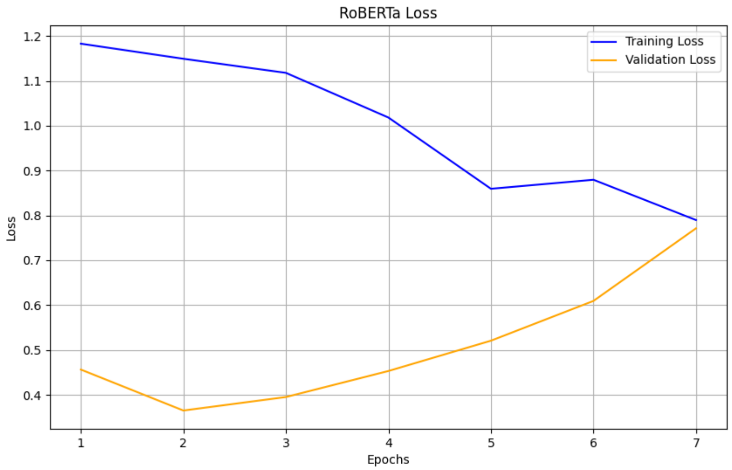
```python
trainer_xlnet.train()
predictions_xlnet = trainer_xlnet.predict(test_dataset_xlnet)
y_pred_xlnet = np.argmax(predictions_xlnet.predictions, axis=1)
```

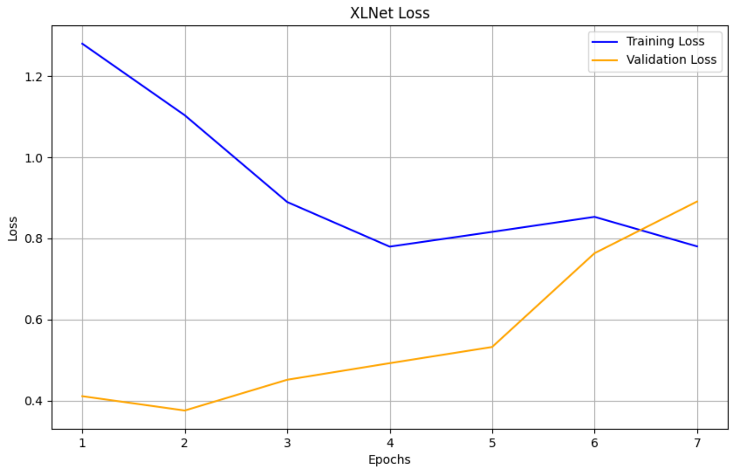
[8750/12500 2:18:29 < 59:21, 1.05 it/s, Epoch 7/10]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.311800	0.411099	0.853700	0.827353	0.853700	0.827457
2	0.322100	0.375768	0.856800	0.854428	0.856800	0.855095
3	0.224500	0.451623	0.854600	0.840628	0.854600	0.844280
4	0.116600	0.492462	0.856500	0.858611	0.856500	0.856312
5	0.161400	0.532378	0.858300	0.848452	0.858300	0.851994
6	0.154900	0.763616	0.852000	0.853786	0.852000	0.852870
7	0.119800	0.890975	0.849600	0.853505	0.849600	0.851453



Training and Validation Loss





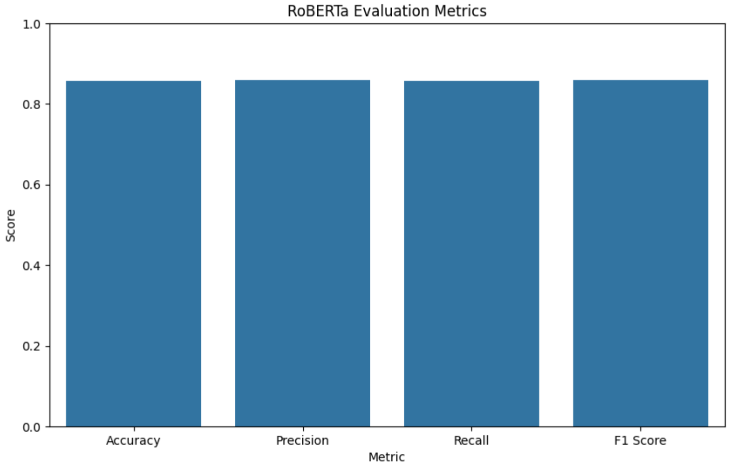
Both models show a decreasing trend in training loss initially, indicating that they are learning from the training data.

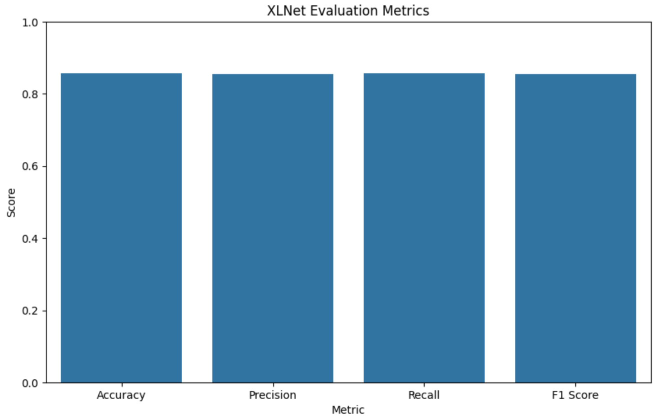
The validation loss for both models increases over time, suggesting potential overfitting as the models become more specialized to the training data.

For both models, early stopping could be considered around epoch 4 or 5, where the validation loss starts to increase more significantly, to prevent further overfitting

while both XLNet and RoBERTa show initial improvements in training loss, RoBERTa demonstrates better generalization to the validation data with a slower increase in validation loss, suggesting it might be more robust against overfitting compared to XLNet.

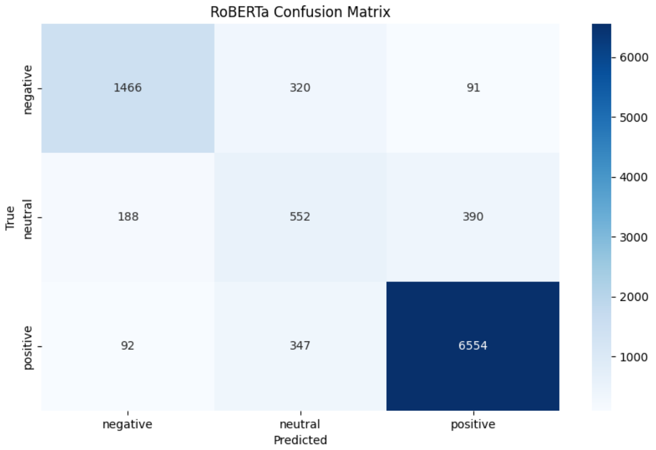
Evaluation Metrics

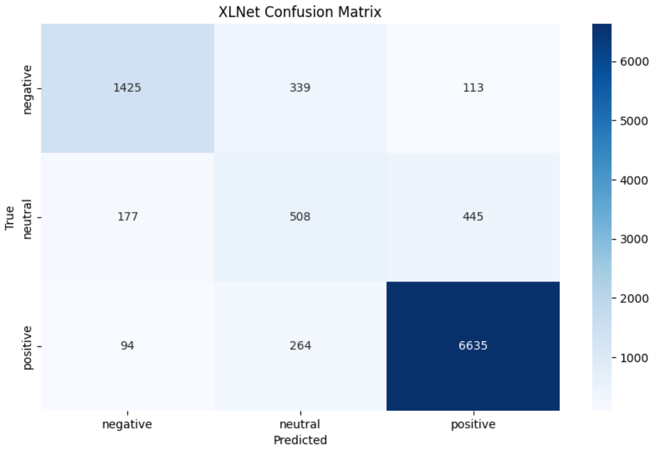




both XLNet and RoBERTa demonstrate very similar performance across all evaluated metrics, with scores around 0.85 for accuracy, precision, recall, and F1 score. This suggests that both models are equally effective in this particular classification task based on the provided evaluation metrics.

Confusion Matrix





while both models show strong performance in classifying sentiments, RoBERTa demonstrates slightly better accuracy and fewer misclassifications, particularly in neutral and positive categories.

Comparison of Models

Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	0.81	0.78	0.74	0.76
SVM	0.80	0.83	0.03	0.07
RoBERTa	0.85	0.84	0.83	0.83
XLNet	0.84	0.83	0.82	0.82

Table 4.1: Comparison of Models

RoBERTa and XLNet outperform the traditional machine learning models (Logistic Regression, Random Forest, and SVM) across all metrics. RoBERTa shows the best overall performance with the highest accuracy, precision, recall, and F1 Score, indicating it is the most effective model for this classification task among the ones compared. XLNet also performs very well, closely following RoBERTa in all metrics.

Customer Reviews Sentiment Analysis Interface

Conclusion

We focus on building a robust sentiment classification system to analyze customer reviews using advanced machine learning techniques. By leveraging classical models like Logistic Regression and cutting-edge transfer learning models such as RoBERTa and XLNet, the system achieved superior performance. Key phases included data preprocessing, feature engineering, and model evaluation, paving the way for impactful applications in customer sentiment analysis.