E-Commerce Text Classification

Beyza TÜRK

OVERVIEW

This project focuses on **text classification** of an e-commerce dataset, where product descriptions are categorized into predefined classes using **Machine Learning (ML) and Deep Learning (DL) approaches**.

Methodologies Used:

- **Text Preprocessing**: Before applying ML and DL models, the text data was cleaned and preprocessed.
- **Machine Learning Approach**:

TF-IDF (Term Frequency-Inverse Document Frequency) was used for feature extraction.

Support Vector Machine (SVM) was applied for classification.

- **Deep Learning Approach**:

A Convolutional Neural Network (CNN) was implemented to learn patterns and improve classification accuracy.

Preprocessing the Text

Text preprocessing is an important step in text classification to avoid noise, unnecessary characters, and inconsistencies that can affect model's performance.

Standardizing text data, removing irrelevant parts, and converting words into a format suitable for machine learning and deep learning models is the main goal of the text preprocessing.

Regardless of the applied model, following steps applied to obtain clean dataset.

Lowercasing → Converts all text to lowercase to maintain consistency (e.g., "Paper" → "paper").

Punctuation Removal → Removes unnecessary characters like . ; : ? (e.g., "great product!" → "great product").

Stopword Removal → Eliminates common words without any meaning like "is", "the", "and" .

Lemmatization → Converts words into their base form (e.g., "liking" → "like", "worse" → "bad").

Even though both CNN and SVM require text preprocessing, the way they process text features is different.

Preprocessing Differences: CNN vs. SVM

STEP	SVM (TF-IDF)	CNN (Embeddings)
Tokenization	Splits text into words (word-based TF-IDF).	Converts words into numeric embeddings.
Vectorization	Uses TF-IDF (term frequency-inverse document frequency).	Uses word embeddings (Word2Vec, GloVe, or learned embeddings).
Word Order	❌ Ignored (Bag-of-Words model)	✔Preserved (CNN captures local word sequences).
Fixed Lengths	❌ No need for padding (TF-IDF adapts automatically).	✔Requires padding to ensure all sequences have the same length.
Numerical Representation	Sparse matrix (high-dimensional, each word is a feature).	Dense vector (low-dimensional, words are embedded in a continuous space).
Handling Context	❌ No context captured (each word is independent).	✔Context-aware (CNN extracts relationships between words).

SUPPORT VECTOR MACHINE

✓ TF-IDF transforms text into a numerical matrix (used as input features for the) where each word has a weight representing its importance in the dataset.

✓ Dataset splitted into training and testing sets before training the model.

✓Normalized the TF-IDF features using StandardScaler. Standardization helps SVM perform better by ensuring all feature values have a similar scale. "`with_mean=False`" is used because sparse matrices (like TF-IDF) do not support mean centering.

✓SVM model is defined with hyperparameter tuning using GridSearchCV which performs 5-fold cross-validation (`cv=5`) to find the best hyperparameters.

✓ **SVM Hyperparameters:**

- **`C` (Regularization Parameter)**: Controls the trade-off between correct classification of training data and simplicity of the decision boundary.
- **`kernel`**:

`linear`: Best for **high-dimensional data** (like TF-IDF).`rbf`: Captures **non-linear relationships**.`poly`: Works well if text data has **complex patterns**.

- **`gamma`**: Controls how much influence a single training example has.

EVALUATION OF THE SVM

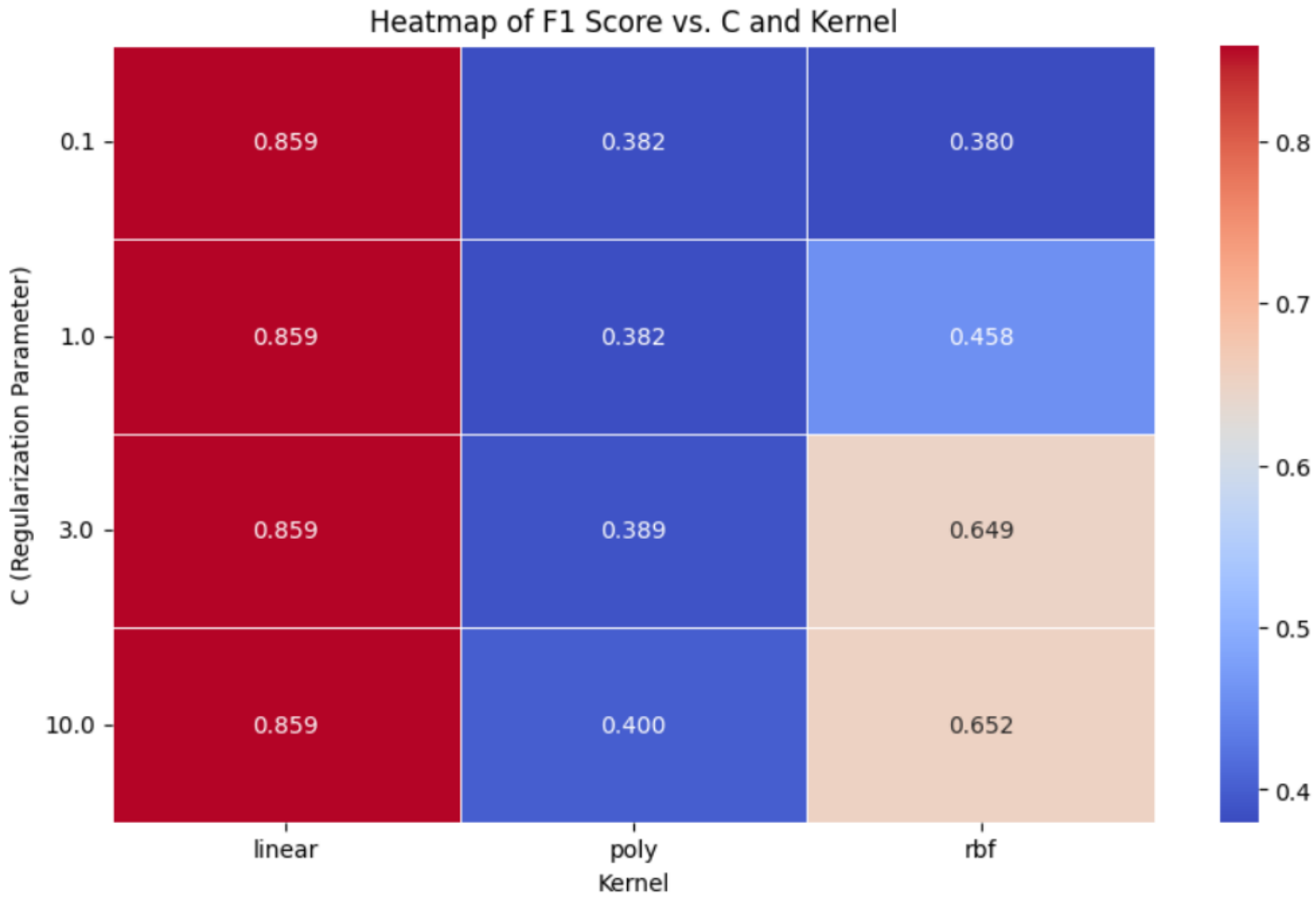
The best SVM model used to make predictions and calculate **F1-score.**

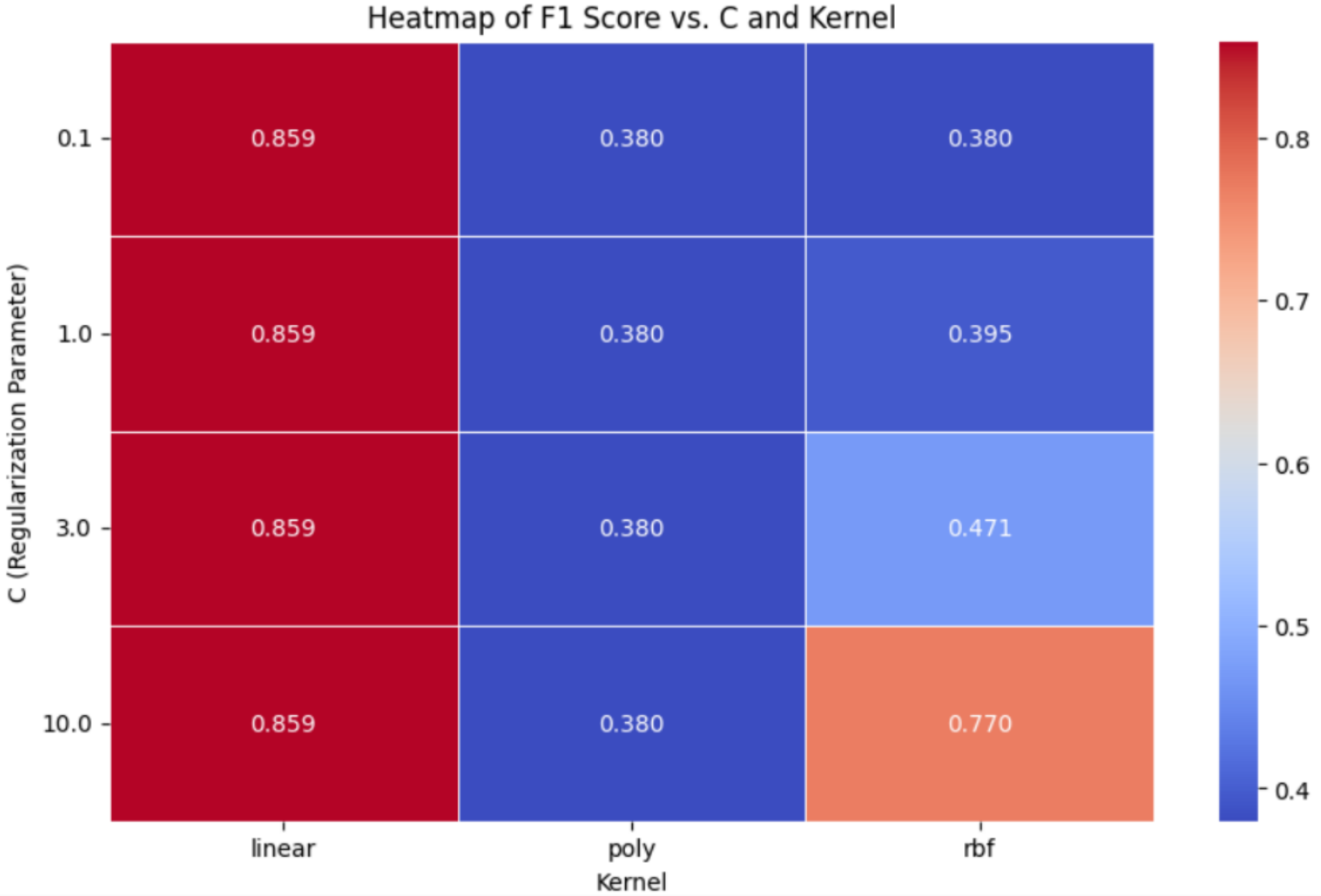
F1-score is used as the evaluation metric to balance precision and recall.

Heatmaps are used to visualize how different kernels and C-values impact F1-score.

```
predicts=best_model.predict(X_test)
print(f1_score(y_test,predicts,average='macro'))
```

```
0.9073512585812358
```


Heatmap of F1 Score vs. C and Kernel
C (Regularization Parameter)
0.1
1.0
3.0
10.0
0.859
0.382
0.380
0.859
0.382
0.458
0.859
0.389
0.649
0.859
0.400
0.652
linear
poly
rbf
Kernel
0.8
0.7
0.6
0.5
0.4


Heatmap of F1 Score vs. C and Kernel
C (Regularization Parameter)
0.1
1.0
3.0
10.0
0.859
0.380
0.380
0.859
0.380
0.395
0.859
0.380
0.471
0.859
0.380
0.770
linear
poly
rbf
Kernel
0.8
0.7
0.6
0.5
0.4

Key Observations

Linear Kernel Always Performs Best (0.859 F1-Score)

- Across all values of **C (0.1, 1, 3, 10)**, the **Linear kernel achieves a stable F1-score of 0.859**.
- This suggests that **a simple linear decision boundary is sufficient** for this classification task.
- Increasing **C** doesn't improve performance for the linear kernel.

RBF Kernel Shows Sensitivity to C and Gamma

When **gamma = scale**:

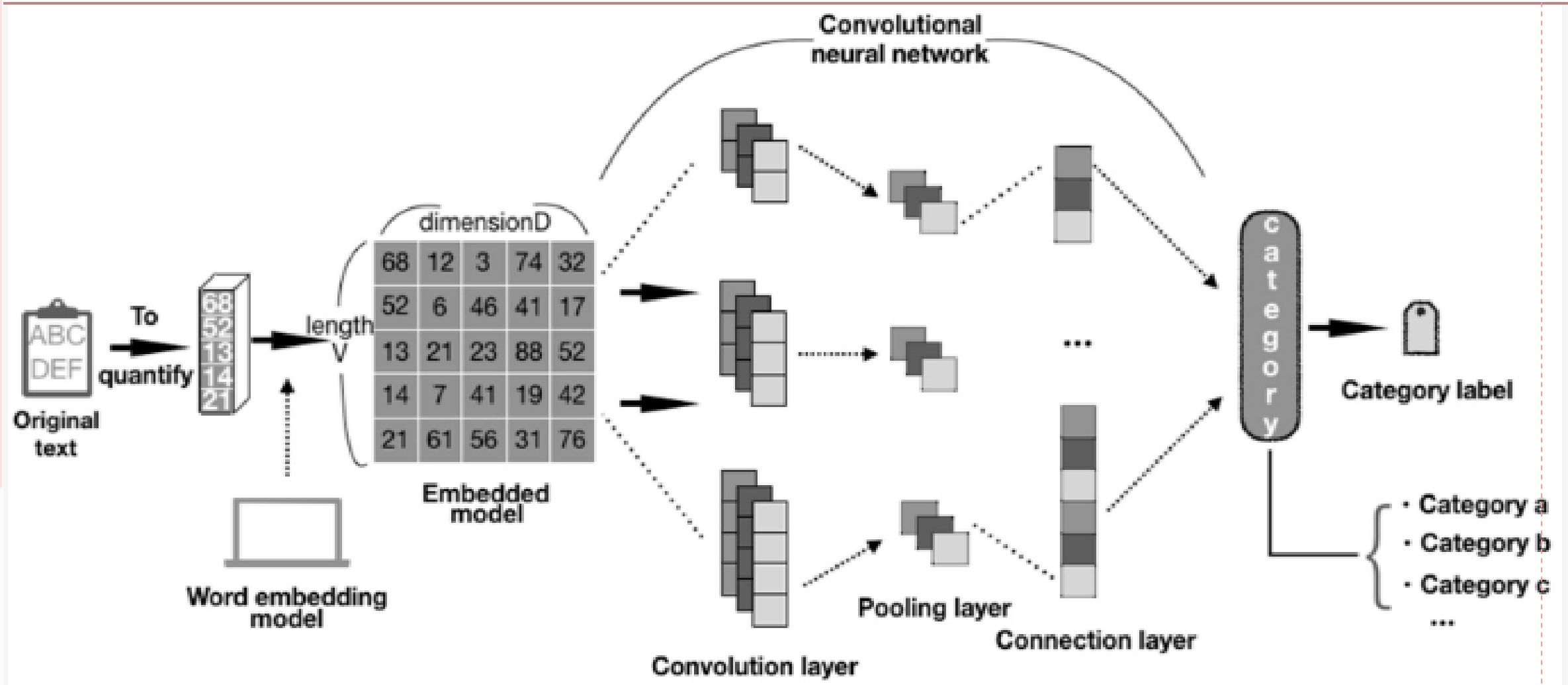
- RBF kernel performs **worse at lower C (0.380–0.652 F1-score)**.
- Best performance at **C=3 or C=10** (F1-score = 0.652).

When **gamma = auto**:

- **RBF kernel reaches 0.770 F1-score at C=10**, showing **higher non-linear capability** at larger C values.
- This suggests that **RBF performs best with a high regularization parameter (C)**.

Polynomial Kernel Performs Poorly

- The **poly kernel has the worst performance (0.380 - 0.400 F1-score)**.
- This means **polynomial decision boundaries do not fit well** for this dataset.



Convolutional Neural Network (CNN)

CNN

- Before feeding text into CNN, it needs to be numerically represented as word embeddings. Following steps applied to obtain numerical representations.

Tokenization: Converts words into integer indices based on vocabulary.
Padding: Ensures all sequences are of the same length (100 words).

- Unlike TF-IDF (which creates sparse matrices), CNN **learns word relationships** through embeddings.
- After preprocessing the text train-test split applied.

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

- **80-20 split** ensures the model is trained on most of the data while keeping some for testing.

CNN ARCHITECTURE

The CNN consists of 5 layers.

LAYER	PURPOSE
Embedding Layer	Converts words into dense vector representations (word embeddings).
Conv1D Layer	Detects local patterns and n-grams in text data.
GlobalMaxPooling1D	Reduces feature size while keeping the most important information.
Dense Layer (64 neurons, ReLU)	Adds non-linearity and improves learning.
Dropout (50%)	Prevents overfitting by randomly dropping neurons during training.
Output Layer (Softmax)	Produces probability scores for each category.

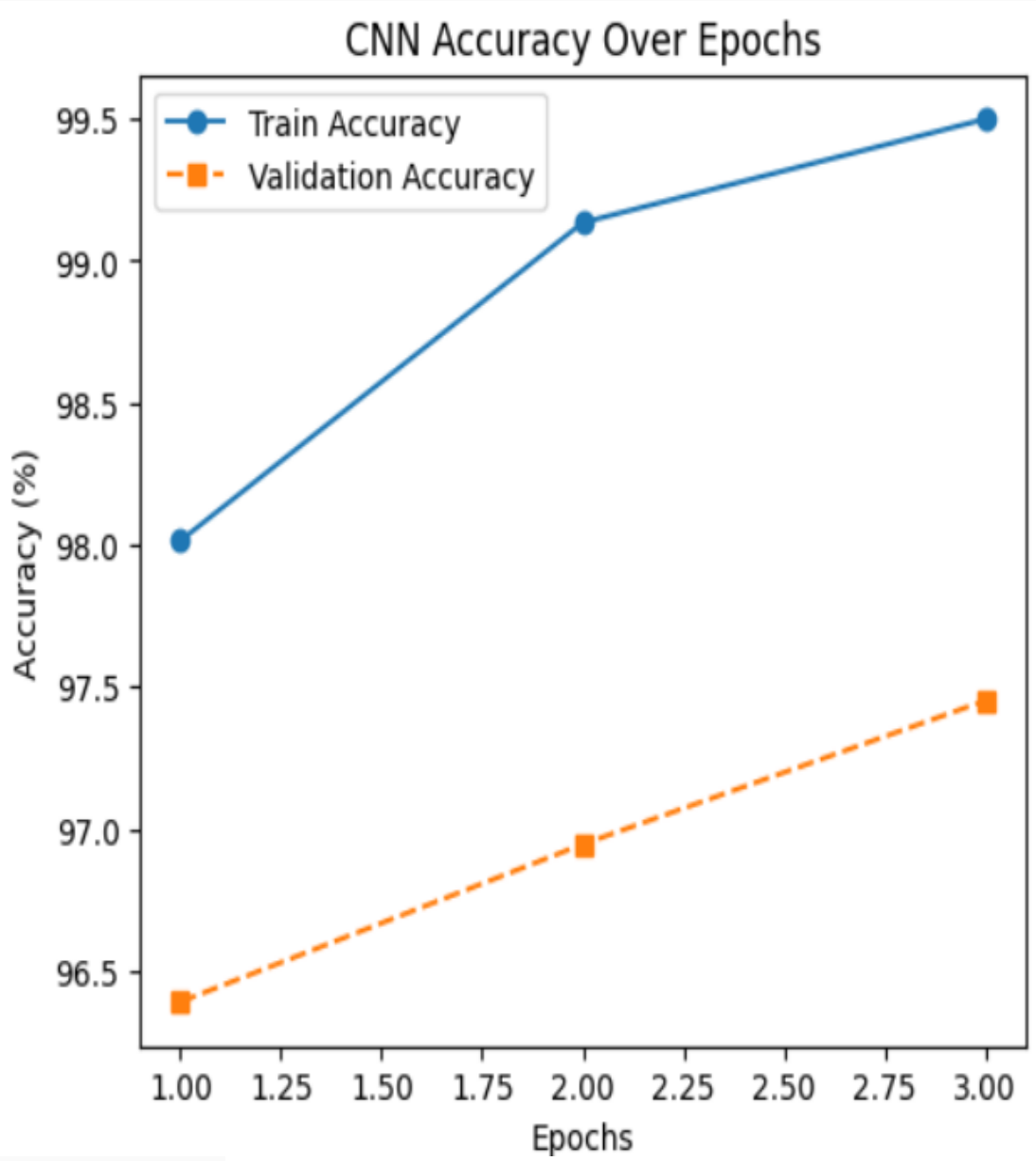
Loss Function: sparse_categorical_crossentropy is used since this is a **multi-class classification** problem.
Optimizer: adam (Adaptive Moment Estimation) is used for efficient learning.
Accuracy & loss per epoch are manually tracked.
Overfitting is prevented by monitoring validation accuracy

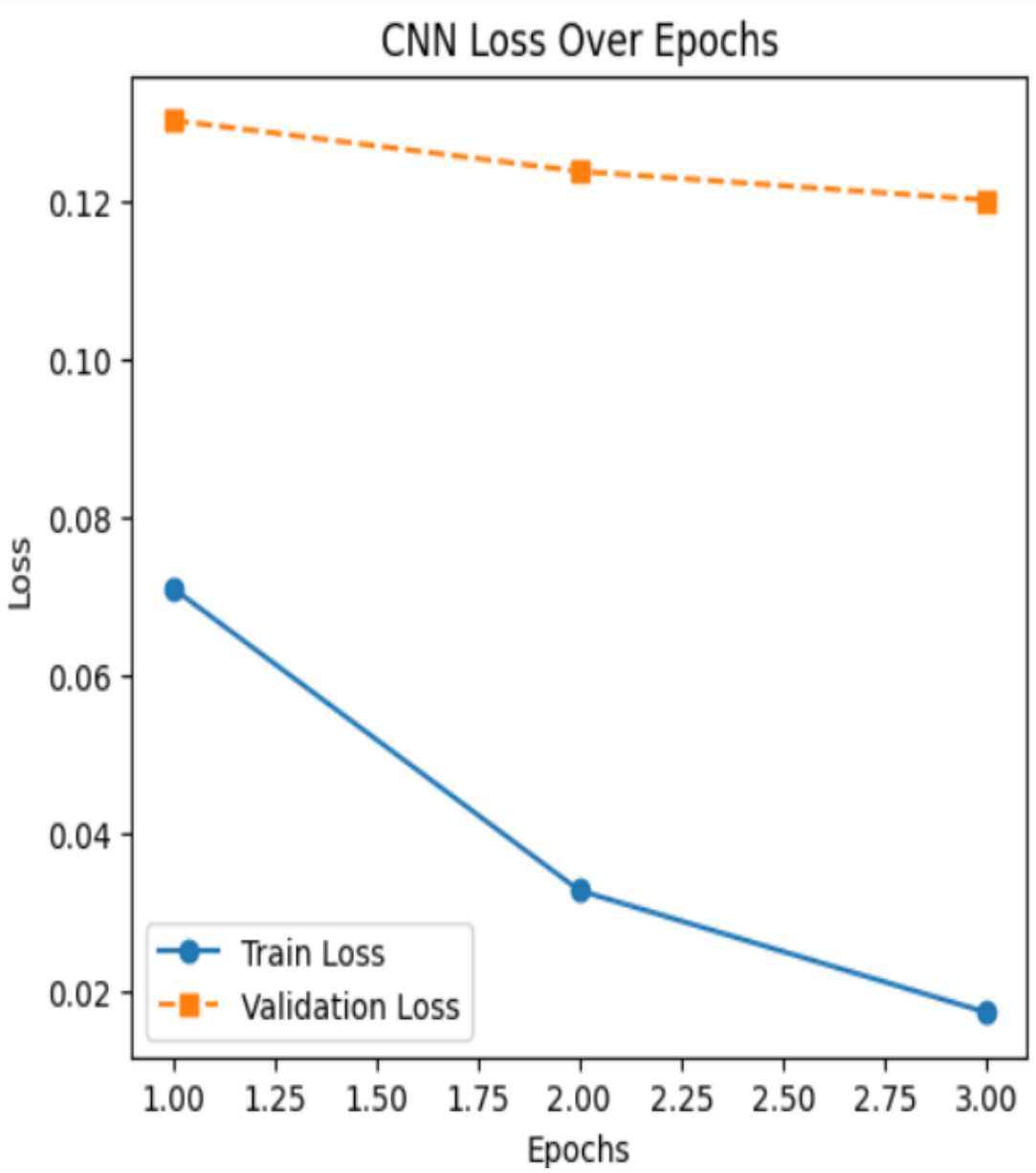
```
Epoch 1/3:
1261/1261 ━━━━━━━━━━━━━━━━━━━━ 251s 197ms/step - accuracy: 0.8044 - loss: 0.5176
Train Accuracy: 98.01%, Validation Accuracy: 96.39%

Epoch 2/3:
1261/1261 ━━━━━━━━━━━━━━━━━━━━ 249s 197ms/step - accuracy: 0.9783 - loss: 0.0884
Train Accuracy: 99.13%, Validation Accuracy: 96.95%

Epoch 3/3:
1261/1261 ━━━━━━━━━━━━━━━━━━━━ 251s 199ms/step - accuracy: 0.9900 - loss: 0.0421
Train Accuracy: 99.50%, Validation Accuracy: 97.45%
```

CNN model is highly accurate and trains efficiently.
The increasing accuracy and decreasing loss shows effective learning.


CNN Accuracy Over Epochs
Train Accuracy
Validation Accuracy
Accuracy (%)
99.5
99.0
98.5
98.0
97.5
97.0
96.5
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
Epochs


CNN Loss Over Epochs
Loss
0.12
0.10
0.08
0.06
0.04
0.02
Train Loss
Validation Loss
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
Epochs

Key Observations

Accuracy (Left Graph)

- **Train Accuracy**: Increased from **98% to 99.5%**.
- **Validation Accuracy**: Improved from **96.5% to 97.4%**.
- **Potential Overfitting**: The gap between training and validation accuracy suggests slight overfitting.

Loss (Right Graph)

- **Train Loss**: Decreased significantly (~0.06 → ~0.01).
- **Validation Loss**: Decreased slightly but remains higher than train loss.

✓ **Good learning** – accuracy improves steadily.

COMPARISON OF CNN VS. SVM

Feature	CNN (Deep Learning)	SVM (Machine Learning)
Feature Extraction	✅ **Learns automatically** (word embeddings)	❌ **Manual TF-IDF** (fixed features)
Captures Word Order?	✅ Yes (understands context & sequence)	❌ No (words treated independently)
Best for Large Datasets?	✅ Yes (scales well)	❌ No (limited with large data)
Computational Cost	❌ **High** (needs GPUs for training)	✅ **Low** (fast & efficient)
Performance on Your Data	**97.4% Accuracy**	**85.9% Accuracy**
Overfitting Risk	**Higher** (needs dropout/regularization)	✅ **Lower** (generalizes well)
Training Time	❌ **Slower** (deep learning requires epochs)	✅ **Faster** (train once, no backpropagation)

Which Model is Better?

✓ **CNN is better** for large datasets, learning deep relationships between words.
✓ **SVM works well** for smaller datasets and requires less computing power.
⚠ **CNN slightly overfits** but has **higher accuracy (97.4%)** than SVM (85.9%).

Thank You

Beyza Türk

Turkbe22@itu.edu.tr