**Gökhan ÇEKÇİ 190101041**  
**Mehmet Tarık AYVAZ 190101034**

**A Sentiment Analysis for Social Media Comments or Product Reviews**

**1. Project Objective**

The objective of this project is to perform text-based sentiment analysis. Two different methods have been used for this purpose: classical machine learning algorithms and deep learning algorithms. Both methods aim to determine whether tweets contain positive or negative sentiment.

**2. Innovations**

**1.Text Cleaning**

Text cleaning processes have been optimized to effectively remove noise from social media data.

**2. Comparison of Different Models**

The performance of classical machine learning models (SVM, Naive Bayes, Random Forest, Logistic Regression) and deep learning models (LSTM) has been compared.

**3. TF-IDF Vectorization**

Using the TF-IDF method, texts were converted into numerical data and vectorized based on the importance of words.

**4. LSTM-Based Model**

An LSTM-based model was used to achieve higher accuracy rates by considering the time series characteristics of the texts.

**5. Performance and Computation Balance**

A balanced performance evaluation was conducted between classical and deep learning methods.

**3. Dataset**

**3.1 Loading and Preparing the Dataset**

The dataset was sourced from Kaggle.com, specifically the "Sentiment140 dataset with 1.6 million tweets". Different subsets of the data were used for machine learning and deep learning algorithms: 2000 tweets for machine learning and 50000 tweets for deep learning. Increasing the dataset size for machine learning algorithms did not significantly improve success rates, whereas it did for deep learning algorithms.

In both code segments, the dataset was loaded using the pandas library. Unnecessary columns were removed, retaining only the sentiment and text columns.

**3.2 Text Cleaning**

Text cleaning was performed in both codes. URLs, usernames, hashtags, numbers, and extra spaces were removed during the cleaning process. The texts were converted to lowercase.

def clean\_text(text):

text = re.sub(r'http\S+', '', text) # Remove URLs

text = re.sub(r'@\w+', '', text) # Remove usernames

text = re.sub(r'#\w+', '', text) # Remove hashtags

text = re.sub(r'\d+', '', text) # Remove numbers

text = re.sub(r'\s+', ' ', text) # Remove extra spaces

text = text.lower().strip() # Convert to lowercase and strip spaces

return text

data['text'] = data['text'].apply(clean\_text)

**3.3 Converting Classes to Numerical Values**

In the deep learning approach, classes were converted to numerical values using LabelEncoder.

le = LabelEncoder()

data['sentiment'] = le.fit\_transform(data['sentiment'])

**4. Splitting into Training and Test Sets**

The dataset was split into training and test sets using the train\_test\_split function in both code segments.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data['text'], data['sentiment'], test\_size=0.2, random\_state=42)

**5. Classical Machine Learning Approach**

**5.1 TF-IDF Vectorization**

Texts were converted into numerical data using the TF-IDF method, with the top 1000 most frequently used words being selected.

vectorizer = TfidfVectorizer(max\_features=1000)

X\_train\_tfidf = vectorizer.fit\_transform(X\_train)

X\_test\_tfidf = vectorizer.transform(X\_test)

**5.2 Training and Evaluating Models**

Different machine learning algorithms were used to train models, which were then evaluated on the test set. The algorithms used include SVM, Naive Bayes, Random Forest, and Logistic Regression.

**5.3 Results and Visualization**

Accuracy, classification reports, and confusion matrices were calculated and visualized for each model.

**Model: Support Vector Machine**

* Accuracy: 0.7232
* Cross-validation scores: [0.65625, 0.66875, 0.6625, 0.69375, 0.75]
* Mean CV score: 0.68625

metin, ekran görüntüsü, diyagram, dikdörtgen içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Model: Naive Bayes**

* Accuracy: 0.7057
* Cross-validation scores: [0.665625, 0.6625, 0.653125, 0.69375, 0.728125]
* Mean CV score: 0.680625

metin, ekran görüntüsü, diyagram, dikdörtgen içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Model: Random Forest**

* Accuracy: 0.6933
* Cross-validation scores: [0.6125, 0.675, 0.625, 0.665625, 0.7]
* Mean CV score: 0.655625

metin, ekran görüntüsü, diyagram, dikdörtgen içeren bir resim

Açıklama otomatik olarak oluşturuldu

**Model: Logistic Regression**

* Accuracy: 0.6958
* Cross-validation scores: [0.659375, 0.665625, 0.65625, 0.69375, 0.74375]
* Mean CV score: 0.68375

metin, ekran görüntüsü, diyagram, dikdörtgen içeren bir resim

Açıklama otomatik olarak oluşturuldu

**6. Deep Learning Approach**

**6.1 Tokenization and Padding**

Texts were tokenized and padded to a fixed length. Texts were converted to numerical values using the Tokenizer, and fixed-length sequences were created with pad\_sequences.

tokenizer = Tokenizer(num\_words=10000)

tokenizer.fit\_on\_texts(X\_train)

X\_train\_seq = tokenizer.texts\_to\_sequences(X\_train)

X\_test\_seq = tokenizer.texts\_to\_sequences(X\_test)

X\_train\_pad = pad\_sequences(X\_train\_seq, maxlen=100)

X\_test\_pad = pad\_sequences(X\_test\_seq, maxlen=100)

**6.2 One-Hot Encoding**

Classes were converted using one-hot encoding:

num\_classes = 2

y\_train\_cat = to\_categorical(y\_train, num\_classes=num\_classes)

y\_test\_cat = to\_categorical(y\_test, num\_classes=num\_classes)

**6.3 Model Creation and Training**

An LSTM-based model was created and trained. The model consists of Embedding, LSTM, and Dense layers:

def create\_model():

model = Sequential()

model.add(Embedding(input\_dim=10000, output\_dim=32))

model.add(LSTM(100))

model.add(Dense(num\_classes, activation='softmax'))

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

return model

model = create\_model()

history = model.fit(X\_train\_pad, y\_train\_cat, epochs=3, batch\_size=32, validation\_data=(X\_test\_pad, y\_test\_cat), verbose=2)

**6.4 Model Performance Evaluation**

The model's performance was evaluated on the test set. Predictions were made, and accuracy was calculated. Confusion matrices and classification reports were created.

loss, accuracy = model.evaluate(X\_test\_pad, y\_test\_cat, verbose=0)

print(f'Accuracy: {accuracy}')

y\_pred\_prob = model.predict(X\_test\_pad)

y\_pred = np.argmax(y\_pred\_prob, axis=1)

y\_test\_labels = np.argmax(y\_test\_cat, axis=1)

conf\_matrix = confusion\_matrix(y\_test\_labels, y\_pred)

print("Confusion Matrix:\n", conf\_matrix)

print("Classification Report:\n", classification\_report(y\_test\_labels, y\_pred))

plt.figure(figsize=(8, 6))

sns.heatmap(conf\_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Positive'], yticklabels=['Negative', 'Positive'])

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Confusion Matrix')

plt.show()

**6.5 Cross-Validation**

The model's performance was evaluated using cross-validation. K-Fold cross-validation was used.

kf = KFold(n\_splits=3, shuffle=True, random\_state=42)

cv\_accuracy = []

for train\_index, val\_index in kf.split(X\_train\_pad):

X\_train\_kf, X\_val\_kf = X\_train\_pad[train\_index], X\_train\_pad[val\_index]

y\_train\_kf, y\_val\_kf = y\_train\_cat[train\_index], y\_train\_cat[val\_index]

model = create\_model()

model.fit(X\_train\_kf, y\_train\_kf, epochs=3, batch\_size=32, verbose=1)

loss, accuracy = model.evaluate(X\_val\_kf, y\_val\_kf, verbose=0)

cv\_accuracy.append(accuracy)

**6.6 Results and Visualization**

Epoch 1/3

1250/1250 - 45s - 36ms/step - accuracy: 0.7398 - loss: 0.5208 - val\_accuracy: 0.7829 - val\_loss: 0.4716

Epoch 2/3

1250/1250 - 44s - 35ms/step - accuracy: 0.8095 - loss: 0.4191 - val\_accuracy: 0.7846 - val\_loss: 0.4638

Epoch 3/3

1250/1250 - 43s - 34ms/step - accuracy: 0.8400 - loss: 0.3675 - val\_accuracy: 0.7772 - val\_loss: 0.4864

Accuracy: 0.7771999835968018

←[1m313/313←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m4s←[0m 13ms/step

Confusion Matrix:

[[3912 1110]

[1118 3860]]

metin, ekran görüntüsü, diyagram, dikdörtgen içeren bir resim

Açıklama otomatik olarak oluşturuldu

Classification Report:

precision recall f1-score support

0 0.78 0.78 0.78 5022

1 0.78 0.78 0.78 4978

accuracy 0.78 10000

macro avg 0.78 0.78 0.78 10000

weighted avg 0.78 0.78 0.78 10000

Epoch 1/3

←[1m834/834←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m36s←[0m 42ms/step - accuracy: 0.6657 - loss: 0.5943

Epoch 2/3

←[1m834/834←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m35s←[0m 42ms/step - accuracy: 0.8150 - loss: 0.4102

Epoch 3/3

←[1m834/834←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m35s←[0m 42ms/step - accuracy: 0.8563 - loss: 0.3447

Epoch 1/3

←[1m834/834←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m32s←[0m 37ms/step - accuracy: 0.6613 - loss: 0.5957

Epoch 2/3

←[1m834/834←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m31s←[0m 37ms/step - accuracy: 0.8131 - loss: 0.4171

Epoch 3/3

←[1m834/834←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m31s←[0m 37ms/step - accuracy: 0.8403 - loss: 0.3598

Epoch 1/3

←[1m834/834←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m32s←[0m 37ms/step - accuracy: 0.6751 - loss: 0.5887

Epoch 2/3

←[1m834/834←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m31s←[0m 37ms/step - accuracy: 0.8164 - loss: 0.4163

Epoch 3/3

←[1m834/834←[0m ←[32m━━━━━━━━━━━━━━━━━━━━←[0m←[37m←[0m ←[1m31s←[0m 37ms/step - accuracy: 0.8512 - loss: 0.3522

Cross-validation scores: [0.7624868750572205, 0.7644191384315491, 0.7554188966751099]

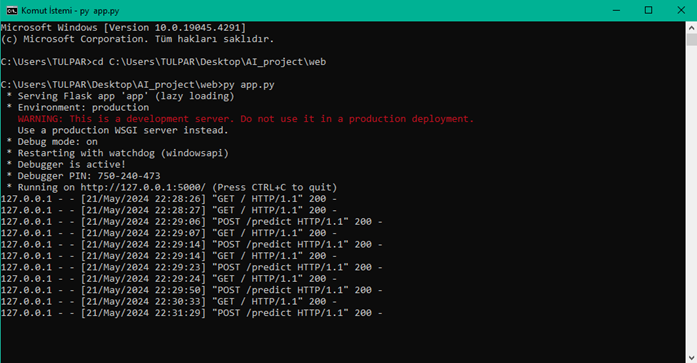
Mean CV score: 0.7607749700546265

**7. Comparison and Comments**

* **Performance Comparison:**
  + The deep learning model generally provided higher accuracy than classical machine learning models.
  + Among the classical machine learning models, Logistic Regression performed the best.
* **Challenges and Advantages:**
  + Classical machine learning methods require fewer computational resources and are faster to train.
  + Deep learning methods require more data and computational resources but generally achieve higher accuracy.

**8. Web Application**

Initially, the model was trained, and a server was created. Then, a web page was prepared and made testable. Users can now enter text messages to find out whether they contain positive or negative emotional content.



**metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu**

**9. Conclusion**

In this project, both classical machine learning and deep learning methods were applied and compared for text-based sentiment analysis. The deep learning method generally provided higher accuracy. However, classical machine learning methods are faster and require fewer computational resources, making them a suitable alternative for small datasets.