

# SUNUM İÇİN HAZIRLANMIŞ SLAYTTAN EKCRAN GÖRÜNTÜLERİ

## Giriş

### Problem Tanımı

- Sosyal medya ve dijital platformlarda nefret söyleminin yaygınlığı artmaktadır.
- Nefret söylemi; bireyler ve topluluklar üzerinde ciddi psikolojik ve sosyal etkiler yaratabilir.
- Bu projede, nefret söylemini otomatik olarak tespit eden bir model geliştirilmektedir.

### Amaç

- Nefret söylemini, hakaret içeren söylemleri ve tarafsız ifadeleri ayırabilen bir sistem oluşturmak.
- Derin öğrenme yaklaşımlarıyla başarılı bir sınıflandırma modeli geliştirmek

## Kullanılan Veri Seti

- Kaynak: Kaggle - hate-speech-and-offensive-language-master
- Veri Seti İçeriği:
  - Toplam 25.000+ tweet.
  - İçerikler 3 sınıfa ayrılmıştır:
    - Nefret Söylemi (Hate Speech)
    - Hakaret (Çirkin Dil) (Offensive)
    - Tarafsız (Çoklu Sınıf Neutral)
- Sınıf Dağılımı dengelenmiştir (oversampling kullanılmıştır).

## Yöntem ve Teknolojiler

### Model

BERT (Bidirectional Encoder Representations from Transformers) modeli kullanıldı.  
Transformers kütüphanesi yardımıyla BERT tümleştirildi.

### Metin İşleme

Metin temizleme:  
-URL'ler, özel karakterler ve büyük/küçük harf farklılıkları giderildi.  
Tokenization: BERT tokenizer kullanıldı.

### Modelleme

Model, 10 epoch boyunca eğitildi(Son versiyon için(7'de zorla durduruldu)).  
AdamW optimizasyonu ve linear scheduler kullanıldı.

## Koddaki Önemli Noktalar

1

BERT Modelinin Kullanımı:  
Transformer tabanlı modellerin gücü, metinlerin anlamını anlamada ve sınıflandırmada önemli bir rol oynar. BertForSequenceClassification bu projeyi özel kılıyor.

2

Oversampling ve Ağırlıklı Kayıp Fonksiyonu:  
Veri dengesizliğini dengelemek için kullanılan oversampling ve sınıf ağırlıklarının kayıp fonksiyonuna eklenmesi, modelin nadir sınıfları öğrenmesini sağlıyor.

3

Erken Durdurma ve Model Kaydetme:  
Eğitim sırasında aşırı öğrenmenin (overfitting) önlenmesi ve en iyi modelin kaydedilmesi, uygulama açısından kritik.

## Sonuçlar

### Model Performansı

Eğitim ve Doğrulama  
Doğrulukları

En iyi epoch sonucunda  
doğrulama doğruluğu: 96.37%

Ortalama eğitim kaybı: 0.1520

### Karışıklık Matrisi

Modelin tahmin  
doğruluğu, normalize  
edilmiş karışıklık matrisi  
ile görsel olarak  
sunuldu.

### Sınıf Performansı

Precision, Recall ve F1-Score

Hate Speech: F1-Score: 0.95

Offensive: F1-Score: 0.93

Neutral: F1-Score: 0.98



# Tartışma ve Çıkarımlar

**Başarılar:**

- BERT tabanlı model, özellikle tarafsız sınıfında yüksek başarı göstermiştir.
- İlgili sınıflar arasındaki doğruluk dengesi sağlanmıştır.

**Zorluklar:**

- Nefret söylemi ve hakaret içeren dil arasındaki ince farklar bazının yanlış sınıflandırılmasına neden olmuştur.

Nefret söylemi ve hakaret içeren dil arasındaki ince farklar bazının yanlış sınıflandırılmasına neden olmuştur.

## Gelecek Çalışmalar

Veri seti daha fazla dil ve kültüre  
genelleştirilebilir.

Daha karmaşık modeller (örneğin, BERT Large  
veya RoBERTa) kullanılabilir.

Gerçek zamanlı analiz için optimizasyon  
yapılabilir.

Gerçek zamanlı analiz için optimizasyon yapılabilir.

# İlk versiyonda 8 epoch

```
==== Epoch 7/8 ====
Eğitim: 100% | 1240/1240 [07:34<00:00, 2.73it/s, accuracy=94.1, loss=0.489]
Eğitim Kaybı: 0.1566, Eğitim Doğruluğu: 94.07%
Doğrulama: 100% | 310/310 [00:34<00:00, 9.07it/s, accuracy=90.4, loss=0.219]
Doğrulama Kaybı: 0.2870, Doğrulama Doğruluğu: 90.42%
precision recall f1-score support
Hate Speech 0.45 0.48 0.47 290
Offensive 0.94 0.94 0.94 3832
Neutral 0.89 0.87 0.88 835
accuracy 0.90 4957
macro avg 0.76 0.76 0.76 4957
weighted avg 0.91 0.90 0.90 4957

==== Epoch 8/8 ====
Eğitim: 100% | 1240/1240 [07:32<00:00, 2.74it/s, accuracy=95, loss=0.0297]
Eğitim Kaybı: 0.1336, Eğitim Doğruluğu: 95.03%
Doğrulama: 100% | 310/310 [00:34<00:00, 9.09it/s, accuracy=89.7, loss=0.222]
Doğrulama Kaybı: 0.3201, Doğrulama Doğruluğu: 89.71%
precision recall f1-score support
Hate Speech 0.42 0.57 0.48 290
Offensive 0.95 0.93 0.94 3832
Neutral 0.90 0.84 0.87 835
accuracy 0.90 4957
macro avg 0.75 0.78 0.76 4957
weighted avg 0.91 0.90 0.90 4957
```

```
==== Epoch 7/8 ====
Eğitim: 100% | 1240/1240 [07:34<00:00, 2.73it/s, accuracy=94.1, loss=0.489]
Eğitim Kaybı: 0.1566, Eğitim Doğruluğu: 94.07%
Doğrulama: 100% | 310/310 [00:34<00:00, 9.07it/s, accuracy=90.4, loss=0.219]
Doğrulama Kaybı: 0.2870, Doğrulama Doğruluğu: 90.42%
precision recall f1-score support
Hate Speech      0.45    0.48    0.47     290
Offensive        0.94    0.94    0.94    3832
Neutral          0.89    0.87    0.88     835

accuracy         0.90    4957
macro avg        0.76    0.76    0.76    4957
weighted avg     0.91    0.90    0.90    4957

==== Epoch 8/8 ====
Eğitim: 100% | 1240/1240 [07:32<00:00, 2.74it/s, accuracy=95, loss=0.0297]
Eğitim Kaybı: 0.1336, Eğitim Doğruluğu: 95.03%
Doğrulama: 100% | 310/310 [00:34<00:00, 9.09it/s, accuracy=89.7, loss=0.222]
Doğrulama Kaybı: 0.3201, Doğrulama Doğruluğu: 89.71%
precision recall f1-score support
Hate Speech      0.42    0.57    0.48     290
Offensive        0.95    0.93    0.94    3832
Neutral          0.90    0.84    0.87     835

accuracy         0.90    4957
macro avg        0.75    0.78    0.76    4957
weighted avg     0.91    0.90    0.90    4957
```

## İkinci versiyonda 16 epoch (9'da durduruldu)

```
==== Epoch 8/16 ====
Eğitim: 100% | 2098/2098 [12:43:00:00, 2.75it/s, accuracy=97.4, loss=0.00462]
Eğitim Kaybı: 0.0774, Eğitim Doğruluğu: 97.44%
Doğrulama: 100% | 525/525 [00:57:00:00, 9.09it/s, accuracy=92.5, loss=0.00212]
Doğrulama Kaybı: 0.3014, Doğrulama Doğruluğu: 92.55%
precision recall f1-score support

Hate Speech    0.88    1.00    0.94    3679
Offensive      0.98    0.88    0.92    3870
Neutral        0.91    0.82    0.86    840

accuracy              0.93    8389
macro avg            0.92    0.90    0.91    8389
weighted avg         0.93    0.93    0.92    8389

==== Epoch 9/16 ====
Eğitim: 100% | 2098/2098 [12:43:00:00, 2.75it/s, accuracy=97.9, loss=0.00141]
Eğitim Kaybı: 0.0646, Eğitim Doğruluğu: 97.93%
Doğrulama: 100% | 525/525 [00:57:00:00, 9.09it/s, accuracy=93.1, loss=0.00301]
Doğrulama Kaybı: 0.2876, Doğrulama Doğruluğu: 93.09%
precision recall f1-score support

Hate Speech    0.89    1.00    0.94    3679
Offensive      0.98    0.89    0.93    3870
Neutral        0.92    0.83    0.87    840

accuracy              0.93    8389
macro avg            0.93    0.91    0.92    8389
weighted avg         0.93    0.93    0.93    8389

Erken durdurma tetiklendi.
```

## Üçüncü versiyonda 10 epoch (7'de durduruldu)

```
==== Epoch 7/10 ====
Eğitim: 100% | 2879/2879 [17:34:00:00, 2.73it/s, accuracy=97.8, loss=0.0214]
Eğitim Kaybı: 0.0670, Eğitim Doğruluğu: 97.82%
Doğrulama: 100% | 720/720 [01:20:00:00, 8.98it/s, accuracy=96.4, loss=0.00312]
Doğrulama Kaybı: 0.1520, Doğrulama Doğruluğu: 96.37%
precision recall f1-score support

Hate Speech    0.92    1.00    0.96    3876
Offensive      0.99    0.90    0.95    3838
Neutral        0.99    0.99    0.99    3800

accuracy              0.96    11514
macro avg            0.97    0.96    0.96    11514
weighted avg         0.97    0.96    0.96    11514

Erken durdurma tetiklendi.
```

## Genel Sonuç

Sonuç olarak, denenmiş 7 değiştirilmiş hiperparametrelili örneklerden sonra ulaşılan en iyi örnek;

Eğitim Kaybı: 0.0670, Eğitim Doğruluğu: 97.82%

Doğrulama Kaybı: 0.1520, Doğrulama Doğruluğu: 96.37%

değerlerine sahiptir.

# İŞLEME İÇİN KULLANILAN KODLAR VE PARÇALANMIŞ İŞ BÖLÜMLERİ

```
# Gerekli kütüphaneleri yükleyin
import pandas as pd
import numpy as np
import nltk
import re
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import torch
from torch.utils.data import DataLoader, Dataset
import torch.nn as nn
from transformers import BertTokenizer, BertForSequenceClassification, get_scheduler
from tqdm import tqdm
```

```
# NLTK'den gerekli veri setlerini indir
nltk.download('stopwords')
from nltk.corpus import stopwords

# Veri setini yükleme
data_path = "C:\\Users\\mamie\\Downloads\\hate-speech-and-offensive-language-master\\data\\labeled_data.csv"
df = pd.read_csv(data_path)

# Metin temizleme fonksiyonu
def clean_text(text):
    text = text.lower()
    text = re.sub(r'http\S+', '', text) # URL'leri kaldır
    text = re.sub(r'^a-zA-Z\s', '', text) # Özel karakterleri kaldır
    text = text.strip()
    return text

# Metin temizleme uygulama
df['tweet'] = df['tweet'].apply(clean_text)

# Sınıf dağılımını dengeleme (oversampling)
class_counts = df['class'].value_counts()
df_balanced = pd.concat([df[df['class'] == cls].sample(class_counts.max(), replace=True) for cls in class_counts.index])
df = df_balanced.reset_index(drop=True)

# Metinleri ve etiketleri ayırma
texts = df['tweet'].values
labels = df['class'].values

# Bert Tokenizer yükleme
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
```

```

# Dataset sınıfı
class HateSpeechDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len):
        self.texts = texts
        self.labels = labels
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, item):
        text = str(self.texts[item])
        label = self.labels[item]

        encoding = self.tokenizer.encode_plus(
            text,
            add_special_tokens=True,
            max_length=self.max_len,
            return_token_type_ids=False,
            padding="max_length",
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt',
        )

        return {
            'input_ids': encoding['input_ids'].squeeze(0),
            'attention_mask': encoding['attention_mask'].squeeze(0),
            'label': torch.tensor(label, dtype=torch.long)
        }

```

```

# Parametreler
EPOCHS = 10
MAX_LEN = 128
BATCH_SIZE = 16
LEARNING_RATE = 2e-5

# Veri setini eğitim ve test olarak bölme
X_train, X_test, y_train, y_test = train_test_split(texts, labels, test_size=0.2, random_state=42)

# DataLoader oluşturma
train_dataset = HateSpeechDataset(X_train, y_train, tokenizer, MAX_LEN)
test_dataset = HateSpeechDataset(X_test, y_test, tokenizer, MAX_LEN)

train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=BATCH_SIZE)

# Model yükleme
model = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=3, hidden_dropout_prob=0.4)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = model.to(device)

# Sınıf ağırlıkları
class_counts = df['class'].value_counts().sort_index().values
class_weights = torch.tensor([sum(class_counts) / c for c in class_counts]).to(device)
loss_fn = nn.CrossEntropyLoss(weight=class_weights)

# Optimizasyon
optimizer = torch.optim.AdamW(model.parameters(), lr=LEARNING_RATE, weight_decay=0.01)
scheduler = get_scheduler("linear", optimizer=optimizer, num_warmup_steps=0, num_training_steps=len(train_loader) * EPOCHS)

# Early Stopping için parametreler
best_val_loss = float('inf')
early_stopping_patience = 3
early_stopping_counter = 0

```

```

# Eğitim fonksiyonu
def train_epoch(model, data_loader, loss_fn, optimizer, device, scheduler):
    model.train()
    total_loss = 0
    correct = 0
    total = 0
    loop = tqdm(data_loader, leave=True, desc="Eğitim")

    for batch in loop:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['label'].to(device)

        optimizer.zero_grad()
        outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        total_loss += loss.item()

        preds = torch.argmax(outputs.logits, dim=1)
        correct += (preds == labels).sum().item()
        total += labels.size(0)

        loss.backward()
        optimizer.step()
        scheduler.step()

        # İlerleme çubuğunu güncelle
        loop.set_postfix(loss=loss.item(), accuracy=100 * correct / total)

    return total_loss / len(data_loader), 100 * correct / total

```

```

# Değerlendirme fonksiyonu
def eval_model(model, data_loader, loss_fn, device):
    model.eval()
    total_loss = 0
    correct = 0
    total = 0
    all_preds = []
    all_labels = []
    loop = tqdm(data_loader, leave=True, desc="Doğrulama")

    with torch.no_grad():
        for batch in loop:
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['label'].to(device)

            outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
            loss = outputs.loss
            total_loss += loss.item()

            preds = torch.argmax(outputs.logits, dim=1)
            correct += (preds == labels).sum().item()
            total += labels.size(0)

            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

            # İlerleme çubuğunu güncelle
            loop.set_postfix(loss=loss.item(), accuracy=100 * correct / total)

    return total_loss / len(data_loader), 100 * correct / total, all_preds, all_labels

```



```

# Eğitim döngüsü
accuracies = []
history = {'train_acc': [], 'val_acc': []}
for epoch in range(EPOCHS):
    print(f"\n==== Epoch {epoch + 1}/{EPOCHS} =====")
    train_loss, train_acc = train_epoch(model, train_loader, loss_fn, optimizer, device, scheduler)
    print(f"Eğitim Kaybı: {train_loss:.4f}, Eğitim Doğruluğu: {train_acc:.2f}%")

    val_loss, val_acc, val_preds, val_labels = eval_model(model, test_loader, loss_fn, device)
    print(f"Doğrulama Kaybı: {val_loss:.4f}, Doğrulama Doğruluğu: {val_acc:.2f}%")
    accuracies.append((epoch + 1, val_acc))
    history['train_acc'].append(train_acc)
    history['val_acc'].append(val_acc)

    print(classification_report(val_labels, val_preds, target_names=['Hate Speech', 'Offensive', 'Neutral']))

    # Early Stopping kontrolü
    if val_loss < best_val_loss:
        best_val_loss = val_loss
        early_stopping_counter = 0
        torch.save(model.state_dict(), "best_model.pt")
    else:
        early_stopping_counter += 1

    if early_stopping_counter >= early_stopping_patience:
        print("Erken durdurma tetiklendi.")
        break

```

```

# En iyi modeli yükleme
model.load_state_dict(torch.load("best_model.pt"))

# Karışıklık matrisi
def plot_confusion_matrix(preds, labels):
    cm = confusion_matrix(labels, preds, normalize='true')
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='.2f', cmap='Blues', xticklabels=['Hate Speech', 'Offensive', 'Neutral'], yticklabels=['Hate Speech', 'Offensive', 'Neutral'])
    plt.xlabel('Tahmin Edilen')
    plt.ylabel('Gerçek')
    plt.title('Karışıklık Matrisi')
    plt.show()

# Test sonuçları
val_loss, val_acc, val_preds, val_labels = eval_model(model, test_loader, loss_fn, device)
print(f"Test Kaybı: {val_loss:.4f}, Test Doğruluğu: {val_acc:.2f}%")
plot_confusion_matrix(val_preds, val_labels)

# Model geçmişi görselleştirme
def plot_history(history):
    plt.figure(figsize=(10, 6))
    plt.plot(history['train_acc'], label='Eğitim Doğruluğu')
    plt.plot(history['val_acc'], label='Doğrulama Doğruluğu')
    plt.xlabel('Epochs')
    plt.ylabel('Doğruluk (%)')
    plt.title('Model Eğitimi ve Doğrulama Doğruluğu')
    plt.legend()
    plt.show()

plot_history(history)

```