

**Teknofest Doğal Dil İşleme Yarışması**

**Proje Dökümantasyonu**

HİVE HEROES

**Takım Üyeleri**

Berfin Duman- [berfiinduman@gmail.com](mailto:berfiinduman@gmail.com)

Bike Sönmez- [bikesnmz@outlook.com](mailto:bikesnmz@outlook.com)

05.04.2023

**İÇERİK**

**1. Projenin Amacı**

[**2. Problem**](#_l7s5gb4cs08n) **Tanımı**

[**3.**](#_llh17ktsx0x5)**Veri toplama ve Ön İşleme**

[**4.**](#_406xwmhpctv9)**Modeller ve Eğitim**

[**5.**](#_xzwq8z3mpkqh)**Model Sonuçlarını ve Performans Analizi**

[**6.**](#_94dyomg21zf) **Projede Karşılaştığımız Zorluklar ve Çözümlerimiz**

**7**[**.**](#_rcbmix1vddd8) **Demo Videosu**

**8. Referanslar**

#### **Projenin Amacı**

Bu projenin amacı Türkçe nefret söylemi ifadelerinin başarılı bir şekilde tespitinin ve sınıflandırılmasının yapılmasını amaçlayan, farklı doğal dil işleme yaklaşımlarını içeren bir yapay zeka modeli inşa etmek ve elde edilen çıktıların endüstride bu alanda faydalı şekillerde kullanımını sağlamaktır.

#### **Problem Tanımı**

Nefret söylemi; ırk, cinsiyet, yaş, ulus, din ya da cinsel yönelim gibi konulara dayanarak, belirli bir grup ya da kişiye yönelik nefret ifadesi içeren veya bu kişilere şiddet uygulanmasını teşvik eden aleni konuşmadır. Nefret söyleminin en önemli yıkıcı etkisi, hedef alınan kişilerin kendilerini güvensiz ve marjinalleştirilmiş hissetmelerine neden olmasıdır. Bu tür söylemler, insanların kişisel ve toplumsal kimliklerini olumsuz etkileyebilir ve hatta ciddi psikolojik sorunlara yol açabilir.

#### **Veri toplama ve Ön İşleme**

Proje için kullandığımız veri seti Teknofest tarafından sağlanan “teknofest\_train\_final.csv” veri setidir. Veri seti aşağıdaki özelliklerden oluşur ;

1. id : Veri setindeki her bir veriye ait eşsiz tanımlayıcı numaralar bütünüdür.

2. text: Türkçe metin verilerini içerir.

3. is\_offensive : Verilen metin verileri içerisinde aşağılayıcı söylem olup

olmadığını (ofansif - 1 veya ofansif olamayan - 0) gösterir .

4. Verilen metin verisi ofansif ve ofansif omaması durumundaki kategorilerini (ofansif - [INSULT, RACIST, SEXIST, PROFANITY], ofansif\_olmayan - [OTHER]) gösterir.

Veri setini analiz ettikten sonra gerekli kütüphane ve modüller ile birlikte veri setini import ettik. “**df = pd.read\_csv(DATA\_PATH,encoding="utf-8")**” komutu ile veri setini bir pandas dataframe olarak aldık ve veri Türkçe karakterler içerdiği için encoding işlermini “utf-8” ile gerçekleştirdik.

Veri setine bazı görselleştirme işlemleri uyguladık;

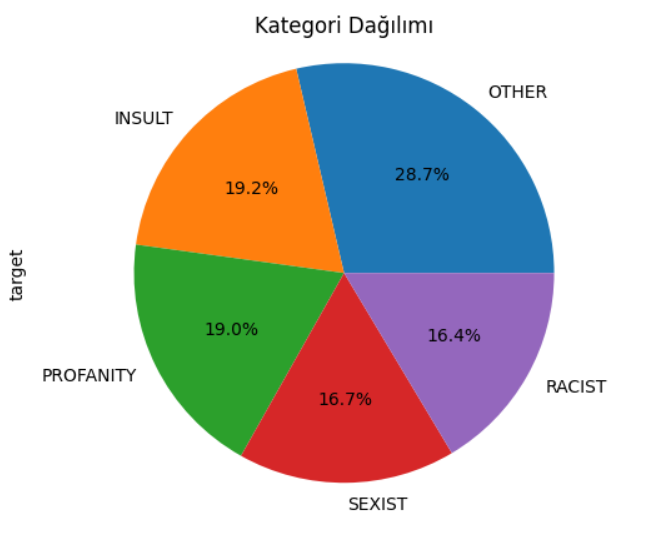
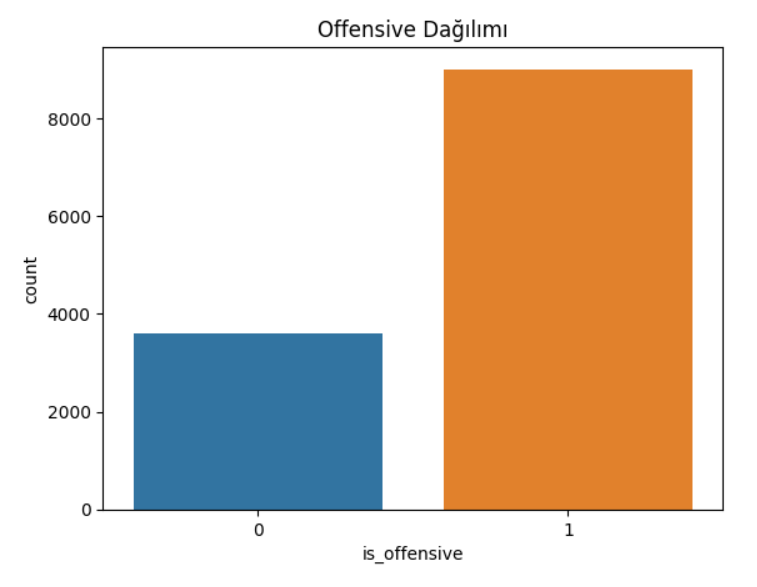


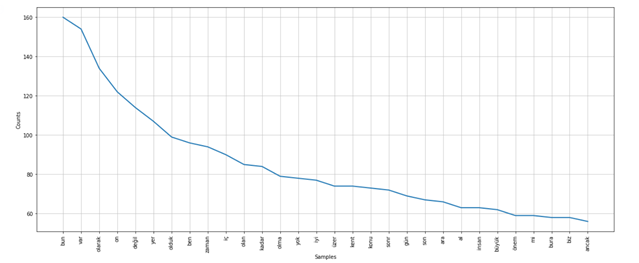
Figure 1 Train Model Ofansif-Değil Dağılımı Figure Train Model Kategorik Sınıflandırma

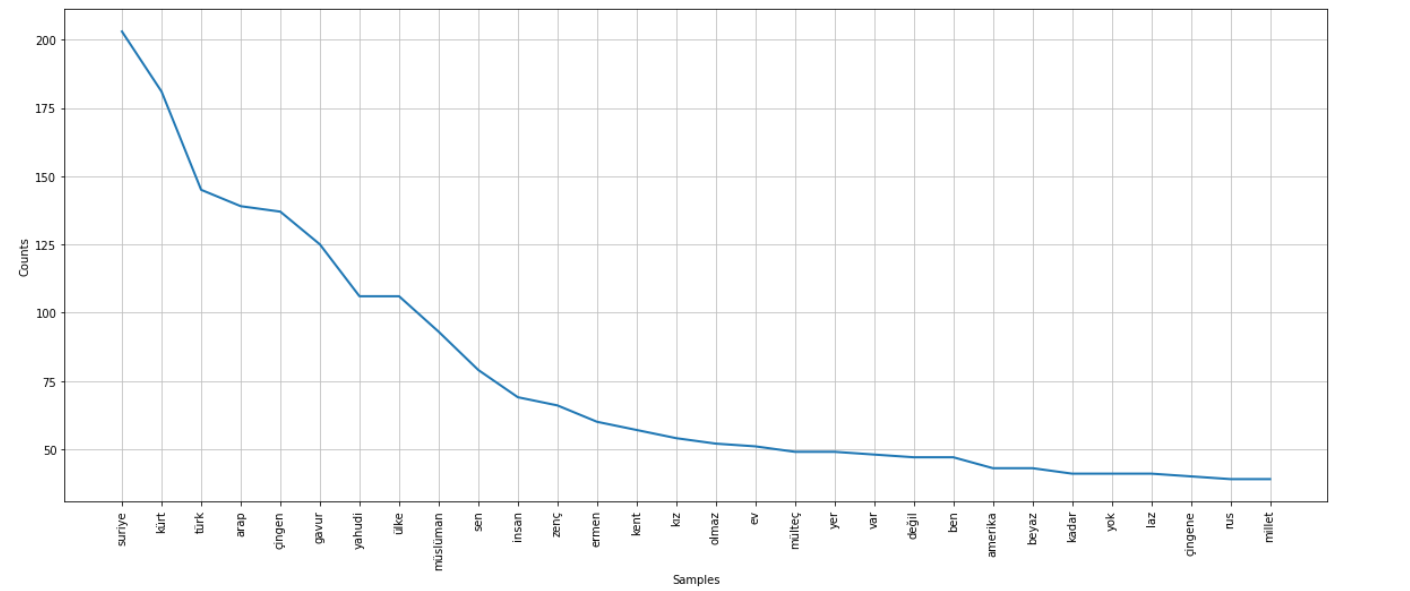


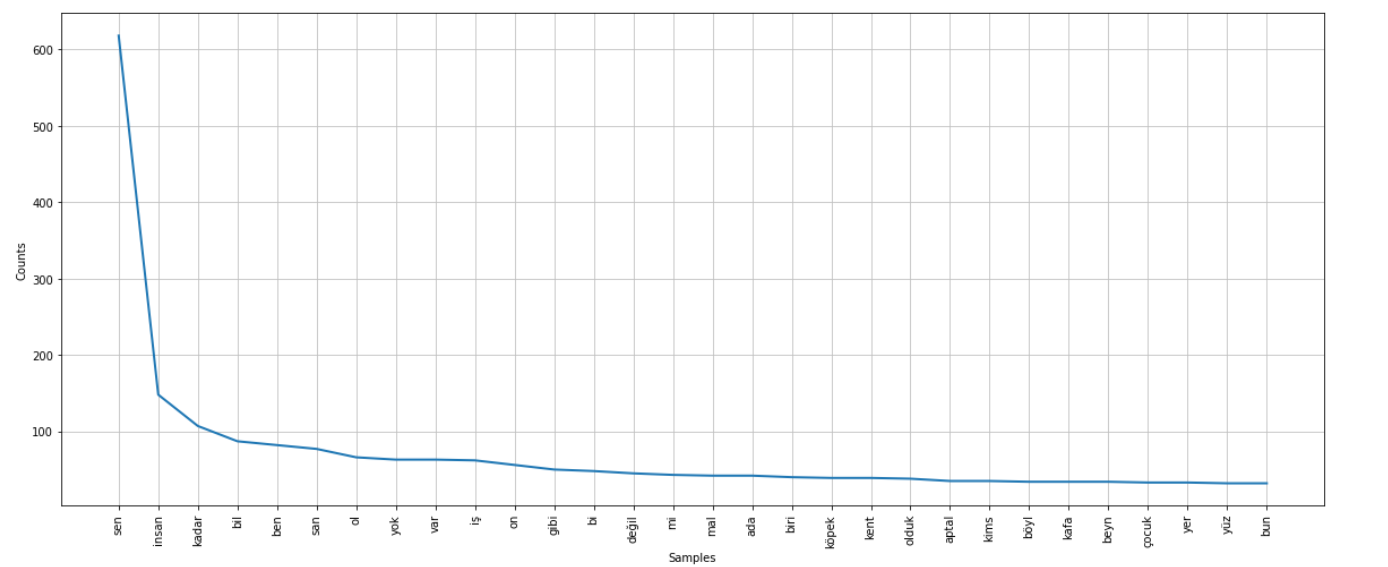
Figure Ofansif Olmayan Kategorisine Ait Kelime Frekans Dağılımı



Figure Ofansif Kategorisine Ait Kelime Frekansı Dağılımı







Veri setini aldıktan sonra veri üzerinde bazı analizler yaptık ve verilerin yapay zeka modelleri ile çalışmaya uygun hale getirilmesi için farklı ön işleme uygulamaları yaptık. Veri ön işleme adımları ise aşağıdaki adımlardan oluşur ;

1. Özel karakterlerin silinmesi ve harflerin küçültülmesi.
2. Tokenizasyon işlemi
3. Durma sözcüklerinin silinmesi
4. Lematizazsyon işlemi
5. Vektörizasyon işlemi

Öncelikle veri setindeki özel karakterleri (Türkçe haricindeki karakterler, noktamala işaretleri, sayılar, gereksiz boşluklar) sildik ve sözcük başlarındaki büyük harfleri küçülttük. Bu işlemlerin birkaçını gerçekleştiren kodlar aşağıda gösterildiği gibidir;

def preprocess\_tweet(text):

text = text.lower() # Convert to lower case

text = re.sub(r'(.)\1+', r'\1\1', text) #Replace repeating characters

text = re.sub(r'[^a-zA-ZçÇğĞıİöÖşŞüÜ\s]', '', text)

#Remove non-Turkish characters

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

return text

def cleaning\_stopwords(text,stop\_words):

return " ".join([word for word in str(text).split() if word not in stop\_words])

def tokenizer\_spacy(text):

doc = nlp(text)

# Generate the tokens

tokens = [token.text for token in doc if len(token.text) != 1]

return tokens

def lemmatize\_text(words):

global unknown\_trlemma\_words

stemmer = TurkishStemmer()

unknown\_trlemma\_words=[]

lemmatized\_words = []

for word in words:

lemmatized\_word = stemmer.stem(word)

lemmatized\_words.append(lemmatized\_word)

return lemmatized\_words

Ayrıca bir sonraki adımda kullanılmak üzere, özellikle geleneksel makine öğrenmesi sınıflandırıcısı modelleri için veri setindeki text verilerine **vektörizasyon** uygulamaları gerçekleştirdik. İki farklı vektörizasyon metodu deneyerek hangisinin daha iyi sonuç verebileceğini öğrenmeye çalıştık. Bu metodlardan ilki bir metin belgesindeki kelime sayılarını hesaplayan ve bu sayıları bir vektör olarak temsil eden **count vektörizasyonudur.** Diğer bir metod ise **TF-IDF (Term Frequency-Inverse Document Frequency)** adı verilen, bir kelimenin belgedeki sıklığına göre (term frequency) ve bu kelimenin diğer belgelerdeki sıklığına göre (inverse document frequency) hesaplanır bu sayede, nadir kullanılan kelimelelerin ağırlığı artarıran ve sıklıkla kullanılan kelimelelerin ağırlığı azaltan bir yötemdir.

Count vektörizasyonu yalnızca kelime sayılarını kullanarak belgeyi temsil ederken, TF-IDF vektörizasyonu her kelimenin önemini hesaplar. Bundan dolayı TF-IDF, genellikle daha anlamlı vektörler elde etmek için kullanılır. Bizim denemelerimize göre de TF-IDF vektörizasyonu daha anlamlı sonuçlar elde etmemizi sağladı. Aşağıda her iki vektörizastonun da kullanıldığı bir fonskiyon örneğini görebilirsiniz;

def vectorizer\_methods(X\_train, X\_test, option):

if option==0: #tf-idf

vectorizer = TfidfVectorizer()

elif option==1: #boW

vectorizer= CountVectorizer()

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

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

return X\_train,X\_test

#### **Modeller ve Eğitim**

Sınıflandırma modelleri için ilk olarak **Bernoulli Naive Bayes Classifier,** [**Support Vector Machine (SVM)**](https://www.javatpoint.com/machine-learning-support-vector-machine-algorithm), **Decision Tree Classifier, XGB Classifier** gibi geleneksel makine öğrenmesi modellerini kullandık ve sonuçlarını karşılaştırarak modellerin geliştirilmesi üzerinde araştırmalar yaptık. Aşağıda bu modellerin her birinin sonuçlarını görebileceğiniz bir fonksiyon ve sonuçlarının bazı görselleştirmeleri mevcuttur;

def sentiment\_analiz(model\_opt,X,y):

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size = 0.05, random\_state =26105111)

X\_train\_vec, X\_test\_vec = vectorizer\_methods(X\_train,X\_test, 1)

if model\_opt==1:

model = BernoulliNB()

model\_name = "Bernoulli NB"

elif model\_opt==2:

model = LinearSVC()

model\_name = "Linear SVC"

elif model\_opt==3:

model = DecisionTreeClassifier()

model\_name = "Decision Tree Classifier"

elif model\_opt==4:

model = XGBClassifier()

model\_name = "XGB Classifier"

model.fit(X\_train\_vec, y\_train)

y\_pred= model\_Evaluate(model=model,model\_name=model\_name,X\_test\_vec=X\_test\_vec,y\_test=y\_test,opt=0)

return y\_pred

def multiple\_choic(X,y,opt):

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.5, random\_state=42)

X\_train\_boW, X\_test\_boW=vectorizer\_methods(X\_train,X\_test, 0)

if opt==0: #y\_train and test are str

svc\_model = LinearSVC()

svc\_model.fit(X\_train\_boW, y\_train)

y\_pred=svc\_model.predict(X\_test\_boW)

print(classification\_report(y\_test, y\_pred, digits = 4))

conf\_matrix(y\_test,y\_pred)

elif opt==1: #y\_train and test are int

xgboost\_model = XGBClassifier()

xgboost\_model.fit(X\_train\_boW, y\_train)

y\_pred=xgboost\_model.predict(X\_test\_boW)

print(classification\_report(y\_test, y\_pred, digits = 4))

conf\_matrix(y\_test,y\_pred)

elif opt==2:

model,X\_test,y\_test=lstm\_model(X,y,num\_class=5)

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

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

print(classification\_report(y\_test, y\_pred, digits = 4))

conf\_matrix(y\_test,y\_pred)

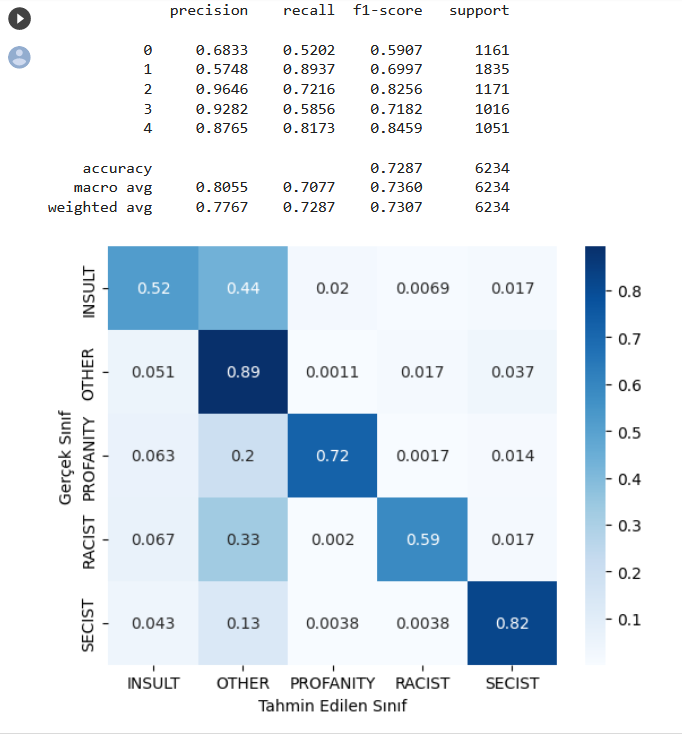
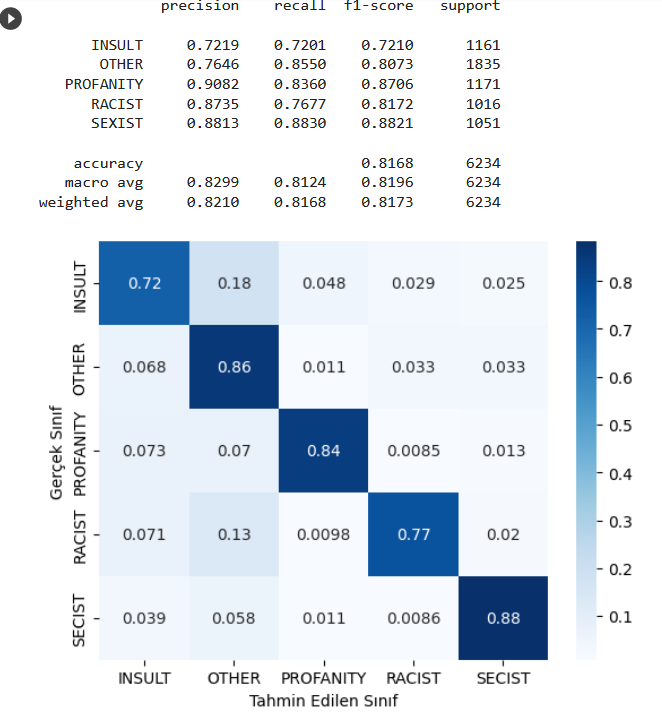
**Linear SVC XGB Classifier**

Figure 5 Kategorik Sınıflandırma Linear SVC model Figure 6 Kategorik Sınıflandırma XGB Sınıflandırıcı Modeli

Kullandığımız geleneksel yöntemlerden bir diğeri ise LSTM - BİLSTM modelleriydi. LSTM (Long Short-Term Memory) ve BiLSTM (Bidirectional Long Short-Term Memory), derin öğrenme modelleri arasında sıklıkla kullanılan yapay sinir ağlarıdır. Normal bir RNN'nin aksine, LSTM hücresi, bir hafıza hücresi ve çıkış kontrolüne sahip bir kapı mekanizması kullanır. Bu mekanizmalar, özellikle uzun süreli bağımlılıkların modellenmesi gerektiğinde, LSTM'nin standart RNN'lerden daha başarılı olmasını sağlar. BiLSTM, bir LSTM'nin iki yönlü bir varyasyonudur. Standart bir LSTM'nin aksine, bir BiLSTM, giriş sırasını hem ileri hem de geri yönde işleyerek iki farklı hafıza hücresi kullanır.

def lstm\_model(X,y,num\_class):

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_number, test\_size=0.02, random\_state=42)

tokenizer = Tokenizer(num\_words=5000, lower=True)

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

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

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

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

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

max\_len = len(max(X\_train, key=len))

X\_train = pad\_sequences(X\_train, padding='post', maxlen=max\_len)

X\_test = pad\_sequences(X\_test, padding='post', maxlen=max\_len)

model = Sequential()

model.add(Embedding(input\_dim=5000, output\_dim=50, input\_length=max\_len))

model.add(LSTM(units=128, dropout=0.2, recurrent\_dropout=0.2))

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

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

model.fit(X\_train, y\_train\_cat, validation\_data=(X\_test, y\_test\_cat), epochs=10, batch\_size=64)

return [model,X\_test,y\_test]



**LSTM BİLSTM**

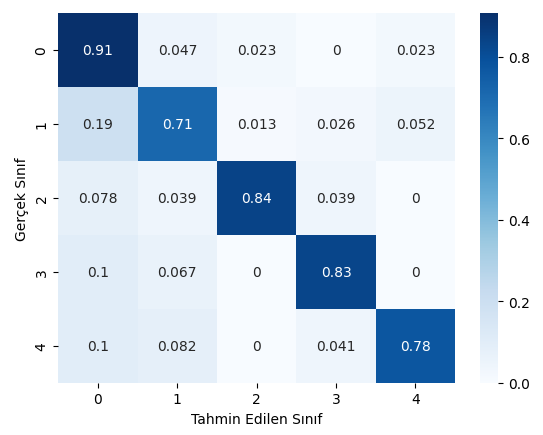
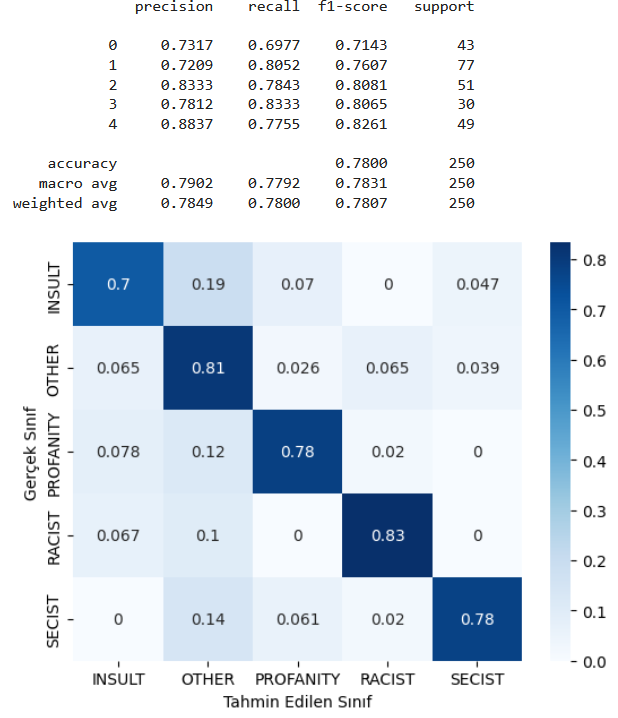
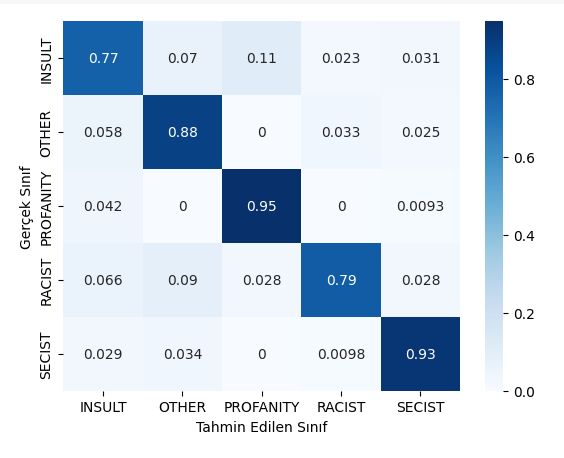


Figure 7 LSTM Sınıflandırıcı F1: 0.79 Figure BiLstm Sınıflandırıcı 0.80

Lstm modeller, traditional modellerden daha gelişmiş olsa da F1 score oranı yine de istediğimiz gibi değildi; bu nedenle transformer modelleri kullanmaya karar verdik. Bundan sonrasında huggingface üzerinden model araştırmaları yaptık

Transformer modelleri öneğitimli modellerdir, ön eğitimler genellikle ingilizce corpus üzerinden olmasına rağmen Bert, GPT, roBERTa, DistilBert kullanabileceğimiz multilingual öneğitimli modeller sunuyordu. Bu modellerden Bert ve distilBert aynı zamanda Türkçe modeller de sunuyordu. Buradan sonrasında deneysel yaklaştık ve aşağıdaki adımları izledik:

1. Bert modelini Türkçe model olan ile ikili sınıflandırma olarak ofansif olup olmaması üzerinde denedik.
2. Bert modelini 5li sınıflandırma yapacak şekilde güncelledik ve pretrained model olarak multilingual model kullandık
3. Yaptığımız 5’li sınıflandırmayı turkçe ön eğitilmiş model üzerinde kullandık.
4. Multilingual model yerine türkçe corpus ön eğitimli modeller daha iyi sonuç verdiği için var olan transformerlardan sadece türkçe ön eğitimli olanları denemeye karar verdik. Bu nedenle roBERTa kullanamadık, distilBert’i denedik yine de en iyi sonucu Bert modelimiz ile aldık.
5. Sonuçları daha da iyileştirmek ve özellikle INSULT’u daha iyi seçebilmesi amacıyla Bert modelimize Conv1d ve BiLSTM i dahil ederek ensemble bir model oluşturduk.
6. **Model Sonuçları ve Performans Analizi**



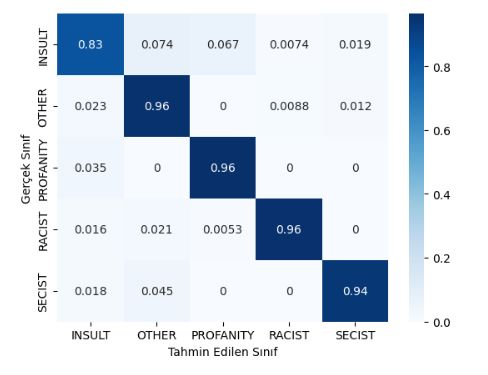


Figure DistilBert Sınıflandırıcı F1 Score= 0.86

Figure 10 Bert Sınıflandırıcı F1 Score= 0.93

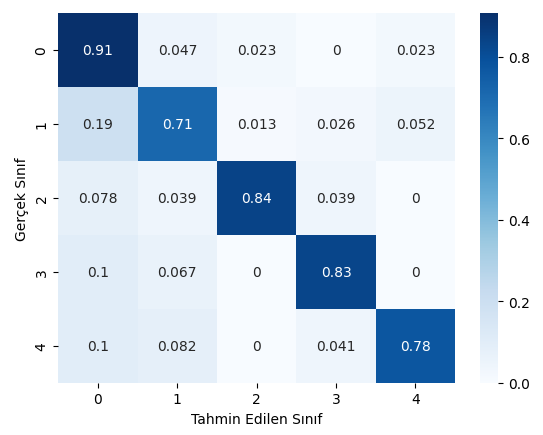


Figure Bert Ensemble Epoch :7 F1 Score=0.94

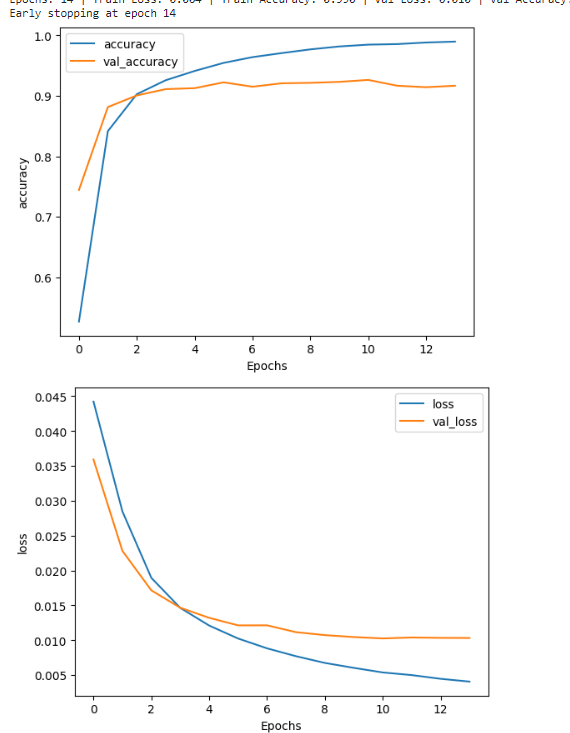
****

Figure 12 Early Stopping Epoch=14

Early stopping, bir makine öğrenimi modelinin eğitim sürecini durdurma tekniğidir. Eğitim sırasında, modelin performansı, örneğin doğruluk oranı, her epoch sonunda değerlendirilir. Eğitim verileri üzerindeki performansı artarken, doğrulama verileri üzerindeki performansı sabit kalır veya azalır. Bu durum, modelin genelleme yapamayacağı anlamına gelir ve aşırı öğrenmeye işaret eder. Biz de modelimizde en iyi epoch u seçmek için bu yöntemi kullandık ve en optimal epoch değerini 11 olarak aldık.

1. **Projede Karşılaştığımız Zorluklar ve Çözümlerimiz**

**Problem:** Proje sürecinde ilk karşılaştığımız zorluk, modellerde “INSULT” kategorisine ait verilerin sınıflandırılması konusundaki başarı düşüklüğü oldu. Hem geleneksel sınıflandırma hem de gelişmiş NLP modellerinde, modeller “INSULT” kategorisi haricindeki diğer kategorilerde yüksek bir tahmin oranına sahipken hem insult değerlerin anlamsal olarak ayırt edilebilirliğinin zor olması hem de verinin az olması nedeniyle böyle bir problemle karşılaştık.

**Çözüm:** Bu problemin çözümü için veri artırma işlemlerini gerçekleştirmeyi düşündük ve bunun için araştırmalar yaptığımızda NLP’de veri artırma işleminin karakter, sözcük, cümle ve belge düzeylerinde olacak şekilde kategorilere ayrılabileceğini gördük. Bizim planımız sözcük düzeyinde bir veri artırımı yapmaktı. Bu nedenle sözcük düzeyinde veri artırımını otomatik olarak gerçekleştirebileceğimiz NLPAug, TextDataAugmentation, TextAugment gibi bazı veri artırımı kütüphaneleri araştırdık. Fakat Türkçe dil desteklerinden dolayı bazı problemlerle karşılaştık. Bunun üzerine veri artırma işlemini manuel olarak, insult kategorisindeki verilerde az sıklıkta geçen kelimeleri tespit ederek sıklıkla geçen kelimelerin olduğu cümlelerde insult anlamı taşıyan kelimelerin yerine, tespit ettiğimiz kelimeleri “replace” metodu ile koyarak veri artırmayı denedik. Sonuç olarak insult değerlerimizin sınıflandırılması konusundaki başarısı yaklaşık 0.2 puan kadar arttı fakat diğer kategorilerin tahmin başarıları bu durumdan kötü etkilendi bu nedenle bu işlemi yapmama kararı aldık.

**Problem:** Geleneksel yöntemlerde doğruluk değerlerimizin istediğimiz düzeyde olmaması.

**Çözüm:** Bu problemin çözümü konusunda veri setine ekstra uygulayabileceğimiz herhangi bir şey yapmaya gerek duymadık, zaten çoğu parametreyi ayarlamıştık, bunun yerine daha gelişmiş dil modelleri kullanmayı düşündük. BERT başta olmak üzere CNN, GPT gibi modellerin araştırmasını yaptık ve en sonunda geleneksel yöntemler yerine gelişmiş dil modeli yaklaşımları kullanmaya karar verdik.

**Problem:** Gelişmiş dil modelleri için ilk kullandığımız model olan BERT multilingual modelinde, multilingual paketini kullandığımızda geleneksel yaklaşımlara göre daha iyi sonuç verse de bizi tatmin etmedi.

**Çözüm:** Çözüm olarak BERT’in [**bert-base-turkish-128k-cased**](https://huggingface.co/dbmdz/bert-base-turkish-128k-cased)Türkçe modelini kullandık. Ve nihai sonuçları bunun üzerinden aldık.

1. **Demo Video**

<https://drive.google.com/file/d/1R2T3hE7XAfyZMHEYYci6MJWyUory4jDu/view?usp=share_link>

1. **Referanslar**

<https://www.researchgate.net/publication/365104455_Linguistic-based_Data_Augmentation_Approach_for_Offensive_Language_Detection>

<https://huggingface.co/datasets/Toygar/turkish-offensive-language-detection>

<https://huggingface.co/dbmdz/bert-base-turkish-128k-uncased>

<https://huggingface.co/bert-base-multilingual-cased>

<https://www.tensorflow.org/text/tutorials/classify_text_with_bert?hl=tr>

<https://www.analyticsvidhya.com/blog/2021/12/text-classification-using-bert-and-tensorflow/>