REUTERS PETROL HABERLERİNİ MAKİNE ÖĞRENMESİ ALGORİTMALARI İLE SINIFLANDIRMA

Özgenur Saygı

Bilişim Sistemleri Mühendisliği Bölümü
KOCAELİ Üniversitesi
171307024@kocaeli.edu.tr

Özet

Bu projede amaçlanan Reuters haber sitesinden toplanan petrol haberlerini veri temizleme işlemi yapıldıktan sonra sınıf etiketlerini oluşturup ilgili veri kümesinde sınıflandırma problemini makine öğrenme yöntemleriyle ele alarak metin sınıflandırması yapmaktır. Bu çalışma kapsamında metinleri algoritmaların anlaya bilecekleri verilere dönüştürme aşamasında Binary, Tf, Tf-idf , Word2vec , Glove, Fasttext gösterim modelleri kullanılmıştır. Gösterim modellerinden alınan veriler Rnn, Cnn, Lstm derin öğrenme algoritmaları kullanılarak sınıflandırma işlemi yapılmıştır.

Anahtar Kelimeler - Binary, Tf , Tf-idf , Word2vec , Glove , Fasttext, Rnn, Cnn, Lstm

Abstract

The aim of this project is to classify the oil news collected from the Reuters news site, after the data cleaning process, by creating class labels and by addressing the classification problem in the relevant data set with machine learning methods. In this study, Binary, Tf, Tf-idf, Word2vec, Glove, Fasttext representation models were used in the process of converting texts into data that algorithms can understand. Data obtained from representation models were classified using Rnn, Cnn, Lstm deep learning algorithms.

Keywords - Binary, Tf, Tf-idf, Word2vec, Glove, Fasttext, Rnn, Cnn, Lstm

I.Veri Kümesi

Veri seti Reuters haber sitesinde ki petrol ile ilgili geriye dönük 1 yıllık haberleri içermektedir.Veri seti ingilizce haberlerden oluşmaktadır. Veri setinde iki sınıfa ait haberler yer almaktadır. Bu sınıflar pozitif haberler ve negatif haberlerden oluşmaktadır. Eğitimimizde de bu haberlere bakarak kullanmış olduğum algoritmalara göre test için verdiğimiz verilerin hangi sınıfa ait olduğunu tahmin etmesi bekleniyor. Veri setinde 20219 tane haber yer almaktadır.16036 tane pozitif haber ,4183 tane negatif haberden oluşmaktadır.

Veri setindeki haberleri “ https://www.reuters.com /news/archive/OILPRD “ linkinden python’ın selenium kütüphanesi kullanarak veriler çekildi. Selenium, bilgisayarınıza yüklediğiniz bir driver yardımı ile ekrana chrome, firefox gibi bir tarayıcı açarak, gerçek bir insan gibi istediğiniz tüm işlemleri programlama dili yardımıyla çalıştırmanızı sağlayan bir araçtır. Bu projede chrome driver kullanıldı. Çekilen haberler txt uzantılı dosyaya kaydedildi.Verileri toplama işlemi tamamlandıktan sonra veri setindeki istenmeyen karakterleri ve gereksiz kelimelerin kaldırma işlemi gerçekleştirildi. Veri setini temizleme işlemini de gerçekleştirdikten sonra sınıf etiketlerini oluşturma işlemini gerçekleştirdim. Sınıf etiketlerini oluşturmak için python'ın TextBlob kütüphanesi kullanıldı. TextBlob kütüphanesinin “NaiveBayesAnalyzer” sınıfını kullanarak haberlerin pozitif mi yoksa negatif mi olduğu belirlendi ve sınıf etiketleri oluşturma işlemi de tamamlanmış oldu.

A.Genel Özellikleri

Veri Kümesi Özellikleri : Petrol ile ilgili haberlerden oluşan iki sınıflı bir veri setidir.

Nitelik Özellikleri: Real yani gerçek değerlerden oluşmaktadır.

İlgili Görevler : Sınıflandırma ,birden fazla farklı kategorik etiketler yer almaktadır.

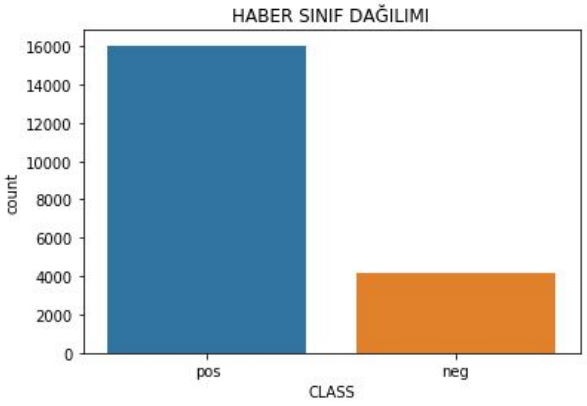
Veri Seti İçindeki Toplam Örnek Sayısı : 20219 tane örnek yer almaktadır.

Sınıf Sayısı : 2 tane sınıf etiketi yer almaktadır.

- ➔ **Negatif Sınıf Sayısı :** 16036
- ➔ **Pozitif Sınıf Sayısı :** 4183

Veri Kümesinin Olası Kullanım Alanı: Metin sınıflandırma, Duygu Analizi

Veri seti dili : İngilizce



Şekil 1: Veri seti sınıf dağılımı

B.Veri Seti Oluşturma Adımları

Adım 1 : İlk olarak chromedriver ve selenium kütüphanesini indirdim.Proje dosyasına selenium kütüphanesine ait webdriver sınıfı import ettim.Bu sınıftan driver adında bir obje oluşturdum.Bu objeyi ilgili linkteki haberleri çekebilmek için kullanıyoruz. İlk for döngüsünü sadece ilgili tarihler arasındaki haberleri çekme işlemi için kullandım.İkinci for döngüsünde o sayfada yer alan 10 haberin sadece içerik kısmını çekip oluşturmuş olduğum listeye atmak için kullandım.

```
from selenium import webdriver
liste=[]
driver=webdriver.Chrome("C:/Users/90539/Desktop/chromedriver/chromedriver.exe")
for j in range(1,517):
    path="https://www.reuters.com/news/archive/OILPRD?view=page&page="+str(j)+"&pageSize=10"
    driver.get(path)
    haber=driver.find_elements_by_xpath('//div[@class="story-content"]/p')
    for i in range(0,11):
        liste.append(haber[i].text)
        print(haber[i].text)
driver.close()
print(len(liste))
```

Şekil 2 : Veri seti kodları

Adım 2 : Bu adımda ise haberlerin yer aldığı listeyi bir metin dosyasına yazacak formata getirdim ve txt dosyasına aktardım.

```
yazi=""
for a in liste:
    yazi=yazi+a
    yazi=yazi+"\n"
import os
dosya1 = open('C:\\Users\\90539\\Desktop\\spyder_project\\oil_news2.txt',
              'w', encoding='utf-8')   # dosya erişimi
dosya1.write(yazi)  # yazdırma işlemi
dosya1.close()
```

Şekil 3 : Veri seti kodları

Adım 3 : Veri setini oluşturduktan data cleaning işlemlerini gerçekleştirildi.”isnull()” fonksiyonu ile veri setinde boş satır var mı kontrol edildi. Daha sonra veri setindeki istenmeyen kelimeleri kaldırabilmek için nltk.corpus kütüphanesinden ingilizce stopwords lerden oluşan bir liste oluşturuldu ve bu kelimeler veri setinden kaldırıldı. Daha sonra temizlemiş olduğum veri setini tekrardan txt dosyasına aktardım.

```
import pandas as pd
data=pd.read_csv("C:\\Users\\90539\\Desktop\\spyder_project\\oil_news.txt",sep="\n")
print(data)

#Veri setinde boş satır var mı kontrol ediliyor
data.isnull().sum()

#Stop word listesi oluşturuluyor
import nltk
from nltk.corpus import stopwords
import re
nltk.download('stopwords')
stop_word_list=stopwords.words('english')
stop_word_list

#İstenmeyen tek karakterler kaldırılıyor
data['NEWS']=data['NEWS'].apply(lambda x: re.sub('[,*\.!?:()"]', '', x))
print(data)

#İstenmeyen kelimeleri datasetimizden kaldırıyoruz
def token(values):
    words = nltk.tokenize.word_tokenize(values)
    filtered_words = [word for word in words if word not in stop_word_list]
    not_stopword_doc = " ".join(filtered_words)
    return not_stopword_doc
data['NEWS'] = data['NEWS'].apply(lambda x: token(x))
```

Şekil 4 : Veri seti kodları

Adım 4 : Sınıf etiketlerini oluşturmak için öncelikle textblob kütüphanesini indirdim. Textblob kütüphanesi pozitif ve negatif sınıflandırma yapmaktadır. Daha sonra metinlerin hangi sınıfa ait olduğunu belirlemek için TextBlob kütüphanesinden “NaiveBayesAnalyzer” sınıfını dahil ettim. Bir döngü aracılığıyla veri setindeki tüm metinleri sırasıyla Textblob kütüphanesinden oluşturduğum objeye parametre olarak vererek hangi sınıfa ait olduğuna dair bir geri dönüş değeri aldım ve bunlarıda bir listeye ekledim. Daha sonra hem haberleri hem de sınıf etiketlerini uygun formata getirip birlikte txt dosyasına yazdırdım.Bu işlemi de gerçekleştirdikten sonra veri seti kullanıma hazır hale gelmiş oldu.

```
from textblob import TextBlob
from textblob.sentiments import NaiveBayesAnalyzer
classifier=[]
for i in range(19021,20219):
    blob = TextBlob(data['NEWS'][i], analyzer=NaiveBayesAnalyzer())
    etiket=blob.sentiment[0]
    classifier.append(etiket)
print(classifier[0])

yazi=""
i=0
for a in data['NEWS'][19021:20219]:
    yazi=yazi+a+","+classifier[i]
    yazi=yazi+"\n"
    i=i+1
dosya1 = open('C:\\Users\\90539\\Desktop\\spyder_project\\oil_news_dataset_etiketli.txt',
              'w', encoding='utf-8')   # dosya erişimi
dosya1.write(yazi)
dosya1.close()
```

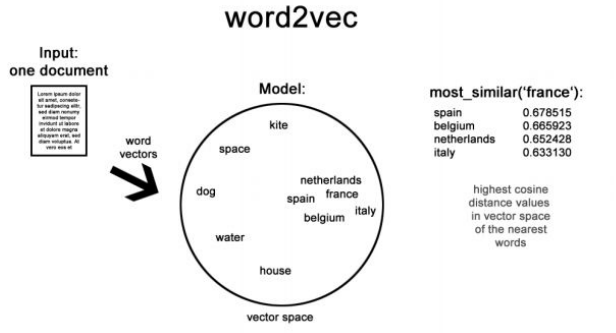
Şekil 5 : Veri seti kodları

II. Gösterim Modelleri

Gösterim modellerini kullanmamızın amacı metinsel ifadelerden oluşan veri setimizi sayısal verilere dönüştürerek yapay sinir ağı algoritmalarının anlayacağı hale getirmektir. Bu projede kullanılan gösterim modelleri şu şekildedir; Word2vec, fasttext, glove, binary, tf-idf, tf modelleridir.

A.Word2vec

Word2Vec , kelimeleri vektör uzayında ifade etmeye çalışan ***unsupervised*** (no labels) ve tahmin temelli(prediction-based) bir modeldir . Google araştırmacı Tomas Mikolov ve ekibi tarafından 2013 yılında icat edilmiştir. Word2Vec kelimeler arasındaki ilişkileri ortaya çıkarmamızı sağlayan bir çeşit algoritma aracıdır. Analiz edilen metinlerde geçen kelimelerin birbirleri ile olan uzaklık ve yakınlık ilişkilerini vektörel olarak hesaplanabilinmesini sağlar. Hesaplanan bu ilişkiler kolay bir şekilde görselleştirilebilir. Kullanılan bir kelimeye en yakın kelimeleri bularak öneri sistemleri oluşturabilirsiniz.



Şekil 6 : Word2vec gösterimi

Word2Vec kelimelerin vektörel temsili için iki farklı model mimariden birini kullanabilir: CBOW (Continuous Bag Of Words) ve skip-gram. CBOW ve Skip-Gram modelleri birbirlerinden output'u ve input'u alma açısından farklılaşmaktadır. CBOW modelinden "windows size" merkezinde olmayan kelimeleri alıp merkezde olan kelimeleri tahmin etmeye çalışırken Skip Gram modelinde ise merkezde olan kelimeleri alıp merkezde olmayan kelimeleri bulunma prensibine göre çalışır

1.Word2vec Projede Kullanımı

Adım 1:İlk olarak eğitim için kullanılacak olan veri setimizi programımıza dahil etmeye çalışıyoruz.Veri setini program içine dahil edebilmek için Pandas kütüphanesini kullandım .

```
import pandas as pd
data=pd.read_csv("oil_etiket.txt",sep=",")
print(data)
```

```
                                                   NEWS CLASS
0      Kuwait National Petroleum Co KNPC Thursday ann...   pos
1      Russia prepared possible drop oil prices OPEC ...   pos
2      The following changes OPEC 's output targets d...   pos
3      The latest list top places invest mining domin...   neg
4      Iran Thursday stood decision deny UN nuclear i...   pos
...                                                  ...   ...
20214  Pembina Pipeline Corp said Wednesday would buy...   pos
20215  In west Texas center US oil boom 3800 students...   neg
20216  Iranian Foreign Minister Mohammad Javad Zarif ...   pos
20217  Noble Group Holdings Noble Holdings plans rebu...   pos
20218  Iranian Foreign Minister Mohammad Javad Zarif ...   pos

[20219 rows x 2 columns]
```

Şekil 7 : Word2vec proje kodları

Adım 2: Yapay sinir ağını oluşturmak için ve word2vec için gerekli kütüphaneler programa dahil edildi.

```
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense,LSTM,SimpleRNN,Conv1D,MaxPool1D,GlobalMaxPool1D,Activation
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from gensim.models.word2vec import Word2Vec
from keras.utils import to_categorical
import re
import nltk
from nltk.corpus import stopwords
from sklearn import preprocessing
from gensim.parsing.preprocessing import remove_stopwords
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import matplotlib.pyplot as plt
```

Şekil 8 : Word2vec proje kodları

Adım 3 : Veri setindeki sınıf etiketi ve haberlerin bulunduğu sütunları sırasıyla x ve y değişkenlerine atıyoruz. x değişkeninde bulunan her bir cümleyi kelimelere ayırıyoruz ve words ismiyle oluşturduğumuz listeye append() fonksiyonu ile ekliyoruz. Böylelikle haberlerdeki tüm kelimeleri içeren words adında bir kelime havuzu oluşmaktadır.

Word2vec modelimizi oluşturmak için gensim adındaki kütüphaneden Word2vec sınıfını çağırdım.Bu sınıftan oluşturduğumuz obje bir takım parametreler almaktadır.Bu parametreler şunlardır :

words: Model için oluşturulmuş, ön işlemden geçirilmiş kelimeler listesidir.Modelin eğitilmesi için kullanılan veri kümesidir.

size : Opsiyonel olarak integer değer alır. Kelime vektörlerinin boyutudur. Bu çalışmada 200 değerini verdim

window: Bir cümle içindeki mevcut ve tahmin edilen kelime arasındaki maksimum mesafedir. Bu çalışmada üç değerini verdim

mincount: Parametre değerinden daha düşük frekanstaki tüm kelimeler yok sayılır

workers: Thread sayısı, modeli eğitmek için kullanılacak iş parçacıkları sayısı. Çok çekirdekli makineler için.Bu çalışmada 16 değerini verdim.

SG: Parametresini belirtmezsek default olarak cbow algoritması çalışır. Modelimize "sg" parametresini belirtmediğimiz için cbow algoritması kullanılmıştır.

```
x=data['NEWS']
```

```
words=[]
for i in x:
    words.append(i.split())
```

```
word2vec_model=Word2Vec(words,size=200,window=3,min_count=1,workers=16)
print(word2vec_model)
```

```
Word2Vec(vocab=23083, size=200, alpha=0.025)
```

Şekil 9 : Word2vec proje kodları

Adım 4 : Verilerimizi tokenizer sınıfı ile ön işleme yapmaktayız.Veri setimizi yapay sinir ağlarında eğitime uygun hale getiriyoruz. Keras kütüphanesinden Tokenizer sınıfını çağırıyoruz ve tokenizer adında bir obje oluşturuyoruz. Metin korpusunu sayısal değerlerden oluşan dizilere dönüştürüyoruz. Cümlelerimizin uzunlukları birbirinden farklı fakat YSA algoritmalarının birçoğunda farklı uzunluklarda olan cümlelerle eğitilemiyor. Farklı uzunluklarda cümleleri tahmin etmesi olanaksız oluyor. Bu nedenle veri setinde bulunan bütün cümleleri aynı boyuta getirmemiz gerekiyor.

"text_to_sequenses" fonksiyonu ile veri setindeki metinleri nümerik hale getiriyoruz.

"pad_sequences" fonksiyonu ile de cümlelerimizi aynı boyuta getiriyoruz. Daha uzun cümleleri kısaltarak ve daha kısa cümleleri de boş değeri (0) ile doldurulmaktadır.

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(x)
x= tokenizer.texts_to_sequences(x)
x=pad_sequences(x)
```

Şekil 10 : Word2vec proje kodları

B.FastText

Facebook AI Research tarafından geliştirilen, gensim yapısında bulunan bir kütüphanedir. FastText, metin sınıflandırılması için geliştirilen bir kütüphanedir. Metin veya kelimeleri herhangi bir dil (konuşma dili) ile ilgili görevde kullanılabilecek sürekli vektörlere dönüştürür. "FastText" 2016 yılında Facebook tarafından geliştirilmiş Word2Vec'in bir uzantısıdır. Tek tek kelimeleri yapay sinir ağına girdi olarak vermek yerine kelimeleri birkaç harf bazlı "n-gram" halinde parçalar. Örneğin elma sözcüğü için **tri gram:** elm, lma'dır. N-gram ifadesinde yer alan n tekrar derecesini ifade etmektedir. Yani kaçar kaçar böleceğimizi buradaki n ifadesi sağlar. Bir kelime veya harften ne kadar olduğunu anlamamızı sağlar. Elma'nın word vektörü tüm bu n-gram vektörlerinin toplamıdır. Eğitim tamamlandıktan sonra eğitim setinde verilen tüm n-gramlar için kelime vektörlerine sahip olacağız. Nadir sıkılıkta geçen kelimelerin n-gramlarının ortaya çıkma olasılığı düşük olduğu için artık bu kelimeler daha doğru bir şekilde temsil edilebilir. Aşağıdaki bölümde FastText'in Gensim ile nasıl kullanılacağını gösterilmektedir.
CBOW modelinde tüm kelimeleri kapsayan bir bakış açısı (seling, these, leather, jacket gibi) alır ve hedefleri tahmin etmek için vektör toplamı kullanır.

SKIPGRAM modelinde ise hedef kelime (fine) verildi ise hedefe rasgele bir yakın kelime kullanarak hedefi belirlemeye çalışır.

1. FastText Projede Kullanımı

FastText modelini kullanabilmek için gensim kütüphanesinden Fasttext sınıfını projeye dahil ediyoruz.Bu sınıftan oluşturduğumuz obje bir takım parametreler almaktadır.Bu parametreler şunlardır :
sentences: Gensim text verisini girdi olarak list of list olacak şekilde alır.
size: Embedding vector boyutu
window: Sağında ve solunda kaç kelime olacağı
min_count: Belli bir sayının altında geçen kelimeleri almaması
workers: Thread sayısı
sg: 0 sa Skip-Gram 1 ise CBOW.

```
from gensim.models import FastText
model_ted = FastText(words, size=100, window=3, min_count=1, workers=4,sg=1)
```

*Şekil 11 :*FastText proje kodları

C. Glove (Global Vectors for Word Representation)

Unsupervised algoritmaların temelinde verilerin istatistikleri yer almaktadır. Skip-gram, CBOW gibi modeller anlamsal bilgileri yakalar ama birlikte kullanılma istatistiklerini kullanmazlar. Matris ayrıştırma yöntemleri bu istatistikleri kullanmasına rağmen anlamsal ilişkileri yakalayamamaktadırlar. Bu tarz modellerde anlamsallık yoktur. Pennington ve diğerleri tarafından önerilen "GloVe" modeli olasılık istatistiklerinden yararlanarak yeni bir objektif fonksiyon oluşturarak bu problemi çözmeyi amaçlamaktadır.

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T w_j + b_i + b_j - \log X_{ij})^2$$

*Şekil 11 :*Glove Formül

Burada X_ij, korpustaki kelime çiftinin (i, j) birlikte geçme sayısıdır. F(x) ağırlık fonksiyonu 3 gereksinime sahiptir:

- ➔ f (0) = 0 olması durumunda bütün terimler sonsuza gitme eğilimi göstermemeli.
- ➔ Birlikte az geçen kelime çiftleri için düşük ağırlık verirken ağırlık fonksiyonu azaltıcı olmamalıdır.
- ➔ X_ij'nin büyük değerleri için bu değer biraz daha küçük olmalıdır.

1.Glove Projede Kullanımı

Adım 1 : İlk olarak önceden eğitilmiş kelime vektörlerinden oluşan txt dosyasını indirdim. İlgili dosyayı https://nlp.stanford.edu/projects/glove/ adre sine giderek projeye uygun olan dosya indirilebilir veya kendiniz de oluşturabilirsiniz.
Önceden eğitilmiş vektörleri yüklemek için, önce kelimeler arasındaki eşlemeleri ve bu kelimelerin gömme vektörlerini tutacak bir sözlük oluşturmalıyız . Bunu da dict() fonksiyonu ile gerçekleştirdim. Ardından kelime vektörlerini çekeceğimiz dosyayı open() fonksiyonu ile açıyoruz. With ifadesinin içine girdikten sonra, dosyadaki her satırdan geçmeli ve satırı bileşenlerinin her birine bölmeliyiz.Satırı böldükten sonra, kelimenin içinde boşluk olmadığını varsayarız ve onu bölünmüş çizginin ilk (veya sıfırıncı) öğesine eşit olarak belirleriz.
Ardından, satırın geri kalanını alıp bir Numpy dizisine dönüştürebiliriz. Bu, kelimenin konumunun vektörüdür.Son olarak, sözlüğümüzü yeni kelime ve ona karşılık gelen vektör ile güncelleyebiliriz.

```
import os
embeddings_index = dict()
f = open("C:\\Users\\90539\\Desktop\\BİL.SİS.MÜH\\3.Sınıf\\2.DÖNEM\\MAKİNE ÖĞRENMESİ\\yazlab_proje2\\glove.6B.100d.txt",
         "r",encoding="utf-8")
for line in f:
    values = line.split()
    word = values[0]
    coefs = asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
print('Loaded %s word vectors.' % len(embeddings_index))
```

```
Loaded 400000 word vectors.
```

Şekil 13 : Glove proje kodları

Oluşturulan vektörü ise yapay sinir ağı algoritmalarında girdi olarak kullanabilmemiz için vektörlerin boyutunu eşitlenmesi gerekiyor. Aşağıdaki kod parçacığı tokenizer sınıfıyla oluşturduğumuz kelime hazinesine göre vektörleri algoritmada kullanacağımız hale dönüştürmektedir.

```
embedding_matrix = zeros((vocab_size, 100))
for word, i in t.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

Şekil 14 : Glove proje kodları

Ç.TF-İDF (Term Frequency – Inverse Document Frequency)

İngilizcedeki Term Frequency – Inverse Document Frequency (Terim frekansı – ters metin frekansı) olarak geçen kelimelerin baş harflerinden oluşan terim basitçe bir metinde geçen terimlerin çıkarılması ve bu terimlerin geçtiği miktara göre çeşitli hesapların yapılması üzerine kuruludur.

Klasik olarak TF yani terimlerin kaç kere geçtiğinden daha iyi sonuç verir. Kısaca TF-IDF hesabı sırasında iki kritik sayı bulunmaktadır. Bunlardan birincisi o anda ele alınan dokümandaki terimin sayısı diğeri ise bu terimi külliyatta içeren toplam doküman sayısıdır.
Son olarak TF-IDF yönteminin diğer yöntemlere göre farkını açıklamaya çalışalım. TF-IDF ile bir terimin kaç kere geçtiği kadar kaç farklı dokümanda da geçtiği önem kazanır. Örneğin sadece bir dokümanda 100 kere geçen bir terimle 10 farklı dokümanda onar kere geçen terimin ikisi de aslında toplamda 100 kere geçmiştir ancak TF-IDF ikincisine yani daha fazla dokümanda geçene önem verir.
Bu yöntem, bir kelimenin/terimin belli bir dökümanda ne kadar geçtiği ve toplam döküman sayısının bu kelimenin geçtiği döküman sayısına oranlanmasına dayanarak, bu terimin dökümandaki ağırlığını gösteren bir yöntemdir.

TF = Dökümanda x teriminin kaç kere geçtiği / Dökümandaki toplam terim sayısı

IDF = ln (Toplam döküman sayısı / İçinde x terimi geçen toplam döküman sayısı)

1.Tf-idf Projede Kullanımı

Adım 1 : Veri setinde ki metinlerin tf-idf değerlerini hesaplayabilmek için kullanabileceğimiz birden fazla yöntem vardır.Bu çalışmada yapay sinir ağı algoritmaları kullanıldığı için tf-idf vektörünü bu algoritmaların embedding yani giriş katmanında kullanacak formata getirmemiz gerekiyor.

İlk olarak keras kütüphanesinin tokenizer sınıfı projeye import edildi. Tokenizer sınıfından oluşturduğum obje ile veri setimizdeki cümlelerde ki en sık geçen kelimelerden oluşan bir kelime havuzu oluşturuldu.Tokenizer sınıfının text_to_matrix() fonksiyonun mode özelliğiyle veri setimizi dönüştürmek istediğimiz vektör uzayına çeviriyoruz. Mode özelliğinin alabildiği değerler sırasıyla şu şekildedir;
Tf-idf, binary, count, freg değerlerinden herhangi birisi seçildikten sonra ilgili vektör uzayına dönüşüm yapılmaktadır. Bu parametrenin değerini belirledikten sonra text_to_matrix() fonksiyonun aldığı bir diğer parametre ise habelerimizin yani metinlerimizin olduğu dönüşüm yapılacak hedef değişkendir. Bu değeride girdikten sonra veri seti tf-idf vektörüne dönüştürme işlemi de tamamlanmış oldu. Yapay sinir ağı algoritmalarına bu vektörü direkt olarak girdi olarak veremiyoruz. pad_sequences() fonksiyonu ile vektörün tüm elemanlarını eşit uzunlukta ayarlıyoruz.

```
from keras.preprocessing.text import Tokenizer
MAX_NUM_WORDS = 650
num_words=MAX_NUM_WORDS
maxlen=650
tokenizer = Tokenizer(num_words=MAX_NUM_WORDS)
tokenizer.fit_on_texts(data['NEWS'])
X=tokenizer.texts_to_matrix(data['NEWS'],mode="tfidf")
```

Şekil 15 : Tf-idf proje kodları

D.Binary

Belgede ilgili kelimenin olması veya olmaması durumuna göre (1,0) olacak şekilde vektör ağırlıklandırılır. Bu vektörler bellekte açık halde tuttuğunuzda 0 lardan dolayı çok yer kaplayabilir. Bag of Words genelde bu yöntemi kullanmaktadır. Döküman vektörleri oluşturulurken Binary Scoring yönteminden yararlanılmıştır.

1.Binary Projede Kullanımı.

İlk olarak keras kütüphanesinin tokenizer sınıfı projeye import edildi. Tokenizer sınıfından oluşturduğum obje ile veri setimizdeki cümlelerde ki en sık geçen kelimelerden oluşan bir kelime havuzu oluşturuldu. Tokenizer sınıfının text_to_matrix() fonksiyonun mode özelliğiyle veri setimizi dönüştürmek istediğimiz vektör uzayına çeviriyoruz.

Mode özelliğinin binary olarak seçildi ve veri setinde ki metinler 1 ve 0 değerlerinden oluşan bir vektör uzayına dönüştürüldü.

```
from keras.preprocessing.text import Tokenizer
MAX_NUM_WORDS = 700
num_words=MAX_NUM_WORDS
maxlen=700
tokenizer = Tokenizer(num_words=MAX_NUM_WORDS)
tokenizer.fit_on_texts(data['NEWS'])
X=tokenizer.texts_to_matrix(data['NEWS'],mode="binary")
```

Şekil 16 : Binary proje kodları

E.TF (Term Frequency)

Bu aşamada vektör uzay modeli olan Tf-idf den sadece tf değeri kullanılmıştır. Vektör uzayları, metinlerin yapısal olmayan formdan sayısal hale getirilmesini sağlayan en geniş kabul gören yöntemdir. Bu yaklaşım aslında her metnin vektör olarak temsil edildiği bir BoW (Bag of Word) sürümü olup, her boyut ayrı bir terime karşılık gelmektedir. Yani her metin, mevcut kelimelerden oluşan MxN büyüklüğünde bir vektördür. Kısacası vektörler üst üste eklenerek döküman-terim matrisi oluşturulur. Bu matris, M adet haber ve n adet terimden oluşmaktadır. İlgili terimler haber içerisinde geçiyorsa o terimin ağırlık değeri sıfırdan farklı olur. Terimin ağırlık değerinden kast edilen aslında ilgili terimin metin üzerindeki etkisidir denilebilir.TF yaklaşımında ise ilgili terimin haber içerisindeki sıklığına bakılır.

1.TF Projede Kullanımı

Sklearn TF-IDF metodu için TdidfVectorizer sınıfını sağlamaktadır. İlk olarak TdidfVectorizer sınıfını kullanabilmek için bu ilgili kütüphaneyi dahil ediyoruz. Daha sonra TfidfVectorizer fonksiyonu çağırıyoruz. Bu fonksiyonda sadece tf değerini hesaplatmak için bir takım parametreler almaktadır. Vektör oluştururken sadece tf değerini hesaplanması için use_idf parametresinin false değerini alması gerekmektedir. Max_feauters özelliği ile de kelimelerin tf değerlerinden bir havuz oluşturuyoruz.

Daha sonra , ilgili sınıftan oluşturduğumuz objenin **fit ve transform** fonksiyonu ile cümlelerdeki kelimelerin tf değerlerini içeren bir vektör oluşması sağlandı.

```
vectorizer = TfidfVectorizer(max_features=650,use_idf=False)
vectorizer = vectorizer.fit(data['NEWS'])
```

```
df = vectorizer.transform(data['NEWS'])
```

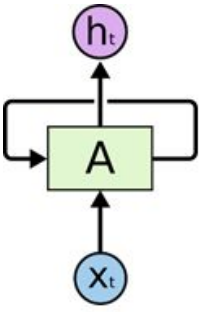
*Şekil 17 :*TF proje kodları

III. Yapay Sinir Ağı Algoritmaları

A. RNN

Tekrarlayan yapay sinir ağları (RNN), birimler arasındaki bağlantıların, yönlendirilmiş bir döngü oluşturduğu ağlardır. RNN ile dinamik zamansal davranış sergilemesine izin verilmektedir. İleri beslemeli sinir ağlarından farklı olarak, RNN'ler kendi giriş belleklerini, girdileri işlemek için kullanılabilir. Bu öznitelik RNN'leri, el yazısı tanıma ve konuşma tanımada, kullanılabilir bir yöntem yapmaktadır.

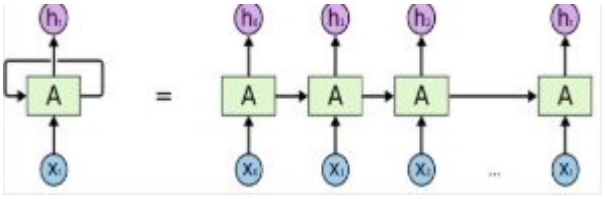
İnsanlar, yeni öğrendikleri bir sözcük için yeni anlamlar öğrenmezler. Daha önceden var olan benzer sözcüklerden yola çıkarak yeni öğrendikleri sözcüğe anlam yüklerler. Ancak geleneksel yapay sinir ağlarında insanlarda bulunan bu anlamlandırma özniteliği bulunmaz ve bu onların en büyük eksikliğidir. Örneğin, videodaki tüm karelere bakarak aktiviteler sınıflandırmak istendiğinde, geleneksel sinir ağları kareler arasında insanlar gibi anlamlandırma kurulamadığından, sınıflandırma yapamayacaktır.



*Şekil 18 :*RNN Döngü Gösterimi

Tekrarlayan Yapay Sinir Ağları (Recurrent Neural Network) ise bir döngü oluşturarak, geçmiş bilgilerin kullanılmasını sağlayacak ve böylelikle kareler arasında anlamlandırma yaparak sınıflandırma yapabilecektir. Şekilde döngü gösterilmektedir.

Üstteki şekilde basit bir tekrarlayan sinir ağ görüntülenmektedir. ‘A’ ismi verilen dikdörtgen, bir yapay sinir ağındaki hücredir. Ağın girdi değeri X’dir. Yapay sinir ağının çıktı değeri h'dır. Hücreden çıkan bir değer yine kendisine gelerek, bir döngü oluşturmaktadır. Bu döngü ile önceki zamanın bilgileri de kullanılabildiğinden yeni bilgi, eski bilgi kullanılarak anlamlandırıla bilmekte ve böylelikle sınıflandırma yapılabilmektedir.
Geleneksel yapay sinir ağlarında, hücrelerden çıkan sonuçlar tekrardan kendilerine girdi olarak gelmemektedir. RNN de ise hücreden çıkan sonuç, tekrardan kendisine girdi olarak gelmektedir. RNN açılırsa aşağıdaki şekildeki gibi bir mimari ortaya çıkmaktadır. Zaman diliminde, aynı hücre kendini birden fazla tekrar etmektedir. Böylelikle de kareler arasında anlamlandırma kurulabilmektedir.



*Şekil 19 :*RNN Döngü Gösterimi

RNN’ler, bir döngü oluşturabildiklerinden, sıralı gelişen aktiviteleri birbirleriyle anlamlandıra bilmektedir. Akış içerisindeki aktivitelerin anlamlandırılarak sınıflandırabilme sinden dolayı son yıllarda yaygın olarak kullanılmaktadır. RNN’lerin, birçok kullanım alanı bulunmaktadır ; konuşma tanımlama, dil modelleme, dil çevrimi, resim başlığı oluşturma vb.
Hali hazırda Keras kütüphanesinde 3 çeşit RNN katmanı var.

1. SimpleRNN
2. LSTM
3. GRU

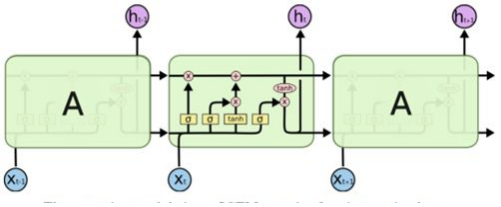
B.CNN

Konvolüsyonel Sinir Ağları (Convolution Neural Network-CNN) çok katmanlı algılayıcıların (Multi Layer Perceptron-MLP) bir türüdür. Görme merkezindeki hücreler tüm görseli kapsayacak şekilde alt bölgelere ayrılmış, basit hücreler, kenar benzeri özelliklere, karmaşık hücreler ise daha geniş alıcılarla, tüm görsele yoğunlaştığı düşünülmektedir. İleri yönlü bir sinir ağı olan CNN algoritması da, hayvanların görme merkezinden esinlenilerek ortaya atılmıştır. Buradaki matematiksel konvolüsyon işlemi, bir nöronun kendi uyarı alanından uyaranlara verdiği cevap olarak düşünülebilir . CNN, bir veya daha fazla konvolüsyonel katman, alt örnekleme (subsampling) katmanı ve bunun ardından standart çok katmanlı bir sinir ağı gibi bir veya daha fazla tamamen bağlı katmandan oluşur .CNN algoritmaları görüntü ve ses işleme alanı başta olmak üzere doğal dil işleme (NLP), biyomedikal gibi bir çok farklı alanda uygulanmaktadır.

C.LSTM (Long- Short Term Memory)

Uzun Kısa Vadeli Hafıza Ağları genellikle “LSTM’ler” olarak adlandırılır uzun vadeli bağımlılıkları öğrenebilen özel bir RNN türüdür. Bunlar Hochreiter & Schmidhuber (1997) tarafından tanıtıldı ve aşağıdaki çalışmalarda pek çok kişi tarafından atıf aldı ve yaygınlaştırıldı. Çok çeşitli sorunlar üzerine muazzam derecede çalışırlar ve şu anda yaygın olarak kullanılmaktadırlar. LSTM’ler, uzun vadeli bağımlılık sorununun önüne geçmek için açıkça tasarlanmıştır. Uzun süre bilgi hatırlamak pratikte varsayılan davranışlarıdır, öğrenmek için uğraştıkları bir şey değildir. Tüm tekrarlayan sinir ağları, tekrar eden sinir ağı modül zinciri biçimindedir. Standart RNN’lerde, bu yinelenen modül, tek bir tanh katmanı gibi çok basit bir yapıya sahip olacaktır.

LSTM’lerin de art arda birbiri takip eden yapıları vardır. Ancak bir sonraki parça farklı bir yapıya sahiptir. Tek bir sinir ağı katmanı yerine, çok özel bir şekilde etkileşimde olan dört parça var



*Şekil 20 :*RNN Döngü Gösterimi

Yukarıdaki diyagramda, her satır bir düğümün çıktısından başkalarının girişlerine kadar tüm vektörü taşır. Sarı kutular sinir ağı katmanları öğrenilirken, pembe daireler vektör eklenmesi gibi noktasal işlemleri temsil eder. Birleştirilen satırlar birleştirme hattını, çizgi çatallaştırma ise kopyalanan içeriği ve kopyaları farklı yere gideceklerini belirtir.

IV.Örnek Uygulamalar

A.Word2vec ve RNN Kullanımı

Adım 1 : İlk olarak eğitim için kullanılacak olan veri setimizi projeye dahil etmeye çalışıyoruz.Veri setini proje içine dahil edebilmek için Pandas kütüphanesini kullandım. Bu kütüphanede ki read_csv() fonksiyonu ile bilgisayarınızdan ilgili bölümden veri setini çağırıyoruz.

```
import pandas as pd
data=pd.read_csv("oil_etiket.txt",sep=",")
print(data)
```

```
                                                    NEWS CLASS
0      Kuwait National Petroleum Co KNPC Thursday ann...   pos
1      Russia prepared possible drop oil prices OPEC ...   pos
2      The following changes OPEC 's output targets d...   pos
3      The latest list top places invest mining domin...   neg
4      Iran Thursday stood decision deny UN nuclear i...   pos
...                                                  ...   ...
20214  Pembina Pipeline Corp said Wednesday would buy...   pos
20215  In west Texas center US oil boom 3800 students...   neg
20216  Iranian Foreign Minister Mohammad Javad Zarif ...   pos
20217  Noble Group Holdings Noble Holdings plans rebu...   pos
20218  Iranian Foreign Minister Mohammad Javad Zarif ...   pos

[20219 rows x 2 columns]
```

Şekil 21 : Word2vec ve RNN kullanımı

Adım 2 : Yapay sinir ağını oluşturmak için ve word2vec için gerekli kütüphaneler programa dahil edildi.

```
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense,LSTM,SimpleRNN,Conv1D,MaxPool1D,GlobalMaxPool1D,Activation
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from gensim.models.word2vec import Word2Vec
from keras.utils import to_categorical
import re
import nltk
from nltk.corpus import stopwords
from sklearn import preprocessing
from gensim.parsing.preprocessing import remove_stopwords
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import matplotlib.pyplot as plt
```

Şekil 22 : Word2vec ve RNN kullanımı

Adım 3: Bu adımda veri setindeki kelimelerin vektör uzayında ifade etmeye çalışıyoruz.Word2Vec kelimeler arasındaki ilişkileri ortaya çıkarmamızı sağlayan bir çeşit algoritma aracıdır.Bu gösterim modelinde kullanılan parametreler ile ilgili ayrıntılı bilgi raporun gösterim modelleri bölümünde yer almaktadır.

```
x=data['NEWS']
```

```
words=[]
for i in x:
    words.append(i.split())
```

```
word2vec_model=Word2Vec(words,size=200,window=3,min_count=1,workers=16)
print(word2vec_model)
```

```
Word2Vec(vocab=23083, size=200, alpha=0.025)
```

Şekil 23 : Word2vec ve RNN kullanımı

Adım 4 : Verilerimizi tokenizer sınıfı ile ön işleme yapmaktayız.Veri setimizi yapay sinir ağlarında eğitime uygun hale getiriyoruz.Keras kütüphanesinden Tokenizer sınıfını çağırıyoruz ve tokenizer adında bir obje oluşturuyoruz. Metin korpusunu sayısal değerlerden oluşan dizilere dönüştürüyoruz. Cümlelerimizin uzunlukları birbirinden farklı fakat YSA algoritmalarının birçoğunda farklı uzunluklarda olan cümlelerle eğitilemiyor. Farklı uzunluklarda cümleleri tahmin etmesi olanaksız oluyor. Bu nedenle veri setinde bulunan bütün cümleleri aynı boyuta getirmemiz gerekiyor.

“text_to_sequenses” fonksiyonu ile veri setindeki metinleri nümerik hale getiriyoruz.

“pad_sequences” fonksiyonu ile de cümlelerimizi aynı boyuta getiriyoruz. Daha uzun cümleleri kısaltarak ve daha kısa cümleleri de boş değeri (0) ile doldurulmaktadır.

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(x)
x= tokenizer.texts_to_sequences(x)
x=pad_sequences(x)
```

Şekil 24 : Word2vec ve RNN kullanımı

Adım 5: Hedef değişken yani sınıf etiketi kategorik. Onu nümerik hale getirmeliyiz. Hatta gölge değişken oluşturmalıyız..**”np_utils“** sınıfının bir fonksiyonu olan **“to_caterogical”** fonksiyonu

çıkış katmanından yani sınıflarımızın olduğu katmandan 2 farklı rakam yerine 0 veya 1 değerlerinden oluşan bir vektör elde ediyoruz.Bu sayede doğru rakama karşılık gelen etiketin indeksi 1 iken diğer tüm etiketler için bu indeks 0 değerini alır.

```
y=data['CLASS']
```

```
from sklearn.preprocessing import LabelEncoder
label_encoder =LabelEncoder()
y = label_encoder.fit_transform(y)
y=to_categorical(y)
pd.DataFrame(y).sample(10)
```

Şekil 25 : Word2vec ve RNN kullanımı

Adım 5 : İlk aşamada "train_test_split()" fonksiyonu ile veri setinin %20 'ni test için ayırdım, %80 'ni eğitim için kullanılacak.x_train ve y_train değişkenlerinde eğitim için ayırdığımız verilerin sırasıyla haber ve sınıf etiketleri yer almaktadır.x_test ve y_test değişkenlerinde ise test etmek için ayırdığımız verilerin sırasıyla haber ve sınıf etiketleri yer almaktadır.

```
x_train, x_test, y_train, y_test = train_test_split(x,y,
                                   test_size = 0.20,random_state=0)
```

Şekil 26 : Word2vec ve RNN kullanımı

Adım 6 : Yapay sinir ağı iki farklı şekilde başlatılabilir: 1. Katmanlar dizilimi (Sequential layers) olarak. 2. Graph olarak. Bu projede ilk yöntem kullanıldı. Bu yöntem için sadece yukarıda indirilen Sequential sınıfından nesne yaratıldı.Bu Sıralı katmanlardan oluşan bir modeldir.Bu aşamada modelimizi oluşturmaya başlandı.

embedding: Her bir kelimenin kaç boyutlu vektör olarak tanımlandığını belirtilir .Böylelikle giriş katmanında ki girdi yani nöron sayısını belirlemiş oluruz.

SimpleRNN: RNN algoritması katmanıdır.

Dense: Dense objesi ile bütün giriş ve çıkış katmanın özelliklerini tanımlayabiliriz.

return_sequences : Bu parametre bir sonraki katman YSA katmanlarından birisi ise true değeri almaktadır.

İlk olarak embedding ile giriş katmanını oluşturup verilerimizi bu katmandan veriyoruz.İlk gizli katmanımız ise SipmleRNN algoritması çalışmaktadır.Bu katmanda 128 tane nöron yer almaktadır.Yapay sinir ağlarında doğrusal olmayan gerçek dünya özelliklerini tanıtmak için aktivasyon fonksiyonuna ihtiyaç duyarız. Bu çalışmada aktivasyon fonksiyonu olarak "tanh" kullanıldı.Fonksiyonun aralığı bu kez (-1,+1) olarak tanımlanmaktadır. Sigmoid fonksiyonuna göre avantajı ise türevinin daha dik olması yani daha çok değer alabilmesidir.

İkinci gizli katman da SimpleRNN algoritması ile çalışmaktadır.Katmanda 64 tane nöron yer almaktadır.Bu çalışmada aktivasyon fonksiyonu olarak "tanh" kullanıldı.

Son eklediğimiz katman ise çıktı katmanıdır.Çıktı katmanındaki nöron sayısını veri setimizde ki sınıf sayısı kadar veriyoruz. Çoklu sınıflandırma problemi üzerine çalıştığımız için "softmax" aktivasyon fonksiyonunu kullanıyoruz.

Bir sonraki aşamada modelimizi compile fonksiyonu ile derlemeye başlıyoruz.

optimizer : Eniyileme algoritmaları, kayıp fonksiyonuna göre ağın nasıl güncelleneceğini belirler. Optimizer, öğrenme oranını kontrol eder.Bu projede ağırlık katsayılarının güncellenmesi için kullanılacak optimizasyon yöntemi **'adam'** dır.

loss : Gerçek değer ile tahmin edilen değer arasındaki hatayı ifade eden metrik. Temelde **"binary_crossentropy"** , ikili doğruluğu gösterir. Çok sınıflı sınıflandırma problemleri için kullanılır.

metrics: Model değerlendirme kriterleri belirlenir. Eğitim ve test sırasında değerlendirmek istediğim metrik accuracy (doğruluk) değeridir.

```
from keras.layers import Flatten, Dense, SimpleRNN, LSTM, Dropout,Activation
model = Sequential()
model.add(word2vec_model.wv.get_keras_embedding(True))
model.add( SimpleRNN(128,activation='tanh',return_sequences=True))
model.add( SimpleRNN(64,activation='tanh',return_sequences=False))
model.add(Dense(2, activation='softmax'))
model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
history = model.fit(x_train, y_train ,validation_data=(x_test, y_test), epochs=3, batch_size=512)
```

Şekil 27 : Word2vec ve RNN kullanımı

B.Glove ve Lstm Kullanımı

Adım 1 : Yapay sinir ağını oluşturmak için ve glove için gerekli kütüphaneler programa dahil edildi.

```
from numpy import array
from numpy import asarray
from numpy import zeros
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Flatten
from keras.layers import Embedding
```

Şekil 28 : Glove ve Lstm kullanımı

Adım 2 : Eğitim için kullanılacak olan veri setimizi projeye dahil etmeye çalışıyoruz.Veri setini proje içine dahil edebilmek için Pandas kütüphanesini kullandım. Bu kütüphanede ki read_csv() fonksiyonu ile bilgisayarınızdan ilgili bölümden veri setini çağırıyoruz.

```
import pandas as pd
data=pd.read_csv("oil_etiket.txt",sep=",")
print(data)
```

```
                                                 NEWS CLASS
0      Kuwait National Petroleum Co KNPC Thursday ann...   pos
1      Russia prepared possible drop oil prices OPEC ...   pos
2      The following changes OPEC 's output targets d...   pos
3      The latest list top places invest mining domin...   neg
4      Iran Thursday stood decision deny UN nuclear i...   pos
...                                                  ...   ...
20214  Pembina Pipeline Corp said Wednesday would buy...   pos
20215  In west Texas center US oil boom 3800 students...   neg
20216  Iranian Foreign Minister Mohammad Javad Zarif ...   pos
20217  Noble Group Holdings Noble Holdings plans rebu...   pos
20218  Iranian Foreign Minister Mohammad Javad Zarif ...   pos

[20219 rows x 2 columns]
```

Şekil 29: Glove ve Lstm kullanımı

Adım 3 : Hedef değişken yani sınıf etiketi kategorik. Onu nümerik hale getirmeliyiz. Hatta gölge değişken oluşturmalıyız..**"np_utils"** sınıfının bir fonksiyonu olan **"to_caterogical"** fonksiyonu çıkış katmanından yani sınıflarımızın olduğu katmandan 2 farklı rakam yerine 0 veya 1 değerlerinden oluşan bir vektör elde ediyoruz.Bu sayede doğru rakama karşılık gelen etiketin indeksi 1 iken diğer tüm etiketler için bu indeks 0 değerini alır.

```
x=data['NEWS']
y=data['CLASS']
```

```
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical
label_encoder =LabelEncoder()
y= label_encoder.fit_transform(y)
y=to_categorical(y)
pd.DataFrame(y).sample(10)
```

Şekil 30: Glove ve Lstm kullanımı

Adım 4 : Tokenizer sınıfından bir obje oluşturuldu ve fit fonksiyonu ile veri setinde ki metinlerden bir kelime havuzu oluşturuldu. Metinleri "text_to_ sequences" fonksiyonu ile nümerik değerlerden oluşan bir vektöre çeviriyoruz.

```
# prepare tokenizer
t = Tokenizer()
t.fit_on_texts(x)
vocab_size = len(t.word_index) + 1
# integer encode the documents
encoded_docs = t.texts_to_sequences(x)
print(encoded_docs)
```

Şekil 31: Glove ve Lstm kullanımı

Adım 5 : Bir önceki adımda nümerik değerlerden oluşturulan vektörün ysa algoritmalarında kullanabilmek için eşit uzunlukta ki vektörlere dönüştürüyoruz. Bu işlemi pad_sequences() fonksiyonu ile yapıyoruz.

```
max_length = 60
padded_docs = pad_sequences(encoded_docs, maxlen=max_length, padding='post')
print(padded_docs)
```

```
[[ 741  124   76 ...    0    0    0]
 [  70 1848  565 ...    0    0    0]
 [  10   80  528 ...    0    0    0]
 ...
 [ 135  185   30 ...    0    0    0]
 [2537   79 1386 ...    0    0    0]
 [ 135  185   30 ...    0    0    0]]
```

Şekil 32: Glove ve Lstm kullanımı

Adım 6 : Veri setini eğitim ve test için ikiye bölüyoruz. Bu aşamada veri setinin %20 sini test için %80 ni eğitim için ayırdık.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(padded_docs,y,test_size = 0.20,random_state=0)
```

Şekil 33: Glove ve Lstm kullanımı

Adım 7: Bu adımda veri setindeki kelimelerin vektör uzayında ifade etmeye çalışıyoruz."GloVe" modeli olasılık istatistiklerinden yararlanarak yeni bir objektif fonksiyon oluşturarak bu problemi çözmeyi amaçlamaktadır.Bu gösterim modelinde kullanılan parametreler ile ilgili ayrıntılı bilgi raporun gösterim modelleri bölümünde yer almaktadır.

```
import os
embeddings_index = dict()
f = open("C:\\Users\\90539\\Desktop\\BİL.SİS.MÜH\\3.Sınıf\\2.DÖNEM\\MAKİNE ÖĞRENMESİ\\yazlab_proje2\\glove.6B.100d.txt",
         "r",encoding='utf-8')
for line in f:
    values = line.split()
    word = values[0]
    coefs = asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
print('Loaded %s word vectors.' % len(embeddings_index))
```

```
Loaded 400000 word vectors.
```

```
embedding_matrix = zeros((vocab_size, 100))
for word, i in t.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
```

Şekil 34: Glove ve Lstm kullanımı

Adım 8: Yapay sinir ağı iki farklı şekilde başlatılabilir: 1. Katmanlar dizilimi (Sequential layers) olarak. 2. Graph olarak. Bu projede ilk yöntem kullanıldı. Bu yöntem için sadece yukarıda indirilen Sequential sınıfından nesne yaratıldı.Bu Sıralı katmanlardan oluşan bir modeldir.Bu aşamada modelimizi oluşturmaya başlandı.

embedding: Her bir kelimenin kaç boyutlu vektör olarak tanımlandığını belirtilir .Böylelikle giriş katmanında ki girdi yani nöron sayısını belirlemiş oluruz.

vocab_size: Veri setinden oluşturulan kelime havuzunun boyutudur.

weights: Her bir girdinin (özelliğin) ağırlık(weight) değeri vardır. Bu girdiler gizli katmandaki düğümler ile bu ağırlıklar aracılığı ile bağlıdır. Girdiler için belirlenen ağırlık değerleri yapay sinir ağında o özelliğin önemini, ağırlığını belirtmektedir.

trainable : Eğitim sırasında ağırlıkların güncellen mesini önlemek için trainable = False ayarlanmaktadır.

dropout : Dropout katmanı her adımda belirtilen orandaki girdiyi rassal olarak sıfıra eşitleyerek modelin veriye aşırı uyum sağlamasının (aşırı öğrenmenin) önüne geçer.

Dense: Dense objesi ile bütün giriş ve çıkış katmanın özelliklerini tanımlayabiliriz.

return_sequences : Bu parametre bir sonraki katman YSA katmanlarından birisi ise true değeri almaktadır.

İlk olarak embedding ile giriş katmanını oluşturup verilerimizi bu katmandan veriyoruz. Bu katmandan sonra ezberlemenin önüne geçmesi için dropout katmanı kullanılmıştır. İlk gizli katmanımız ise Lstm algoritması çalışmaktadır.Bu katmanda 128 tane nöron yer almaktadır.Yapay sinir ağlarında doğrusal olmayan gerçek dünya özelliklerini tanıtmak için aktivasyon fonksiyonuna ihtiyaç duyarız. Bu çalışmada aktivasyon fonksiyonu olarak "tanh" kullanıldı.Fonksiyonun aralığı bu kez (-1,+1) olarak tanımlanmaktadır. Sigmoid fonksiyonuna göre avantajı ise türevinin daha dik olması yani daha çok değer alabilmesidir.

Lstm katmanından sonra bir tane daha dropout katmanı kullanılmıştır.

Son eklediğimiz katman ise çıktı katmanıdır.Çıktı katmanındaki nöron sayısını veri setimizde ki sınıf sayısı kadar veriyoruz. Çoklu sınıflandırma problemi üzerine çalıştığımız için "softmax" aktivasyon fonksiyonunu kullanıyoruz.

```
# define model
from keras.layers import Dropout,LSTM
model = Sequential()
e = Embedding(vocab_size, 100, weights=[embedding_matrix], input_length=60, trainable=False)
model.add(e)
model.add(Dropout(0.5))
model.add(LSTM(128,activation='tanh',return_sequences=False))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))
# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# summarize the model
print(model.summary())
```

Şekil 35: Glove ve Lstm kullanımı

Bir sonraki aşamada modelimizi compile fonksiyonu ile derlemeye başlıyoruz.

optimizer : En iyileme algoritmaları, kayıp fonksiyonuna göre ağın nasıl güncelleneceğini belirler. Optimizer, öğrenme oranını kontrol eder.Bu projede ağırlık katsayılarının güncellenmesi için kullanılacak optimizasyon yöntemi **'adam'** dır.

loss : Gerçek değer ile tahmin edilen değer arasındaki hatayı ifade eden metrik. Temelde **"binary_crossentropy"** , ikili doğruluğu gösterir. Çok sınıflı sınıflandırma problemleri için kullanılır.

metrics: Model değerlendirme kriterleri belirlenir. Eğitim ve test sırasında değerlendirmek istediğim metrik accuracy (doğruluk) değeridir.

Adım 9 : Model yapılandırıldıktan sonra eğitim sürecine başlayabiliriz. Bu, Keras'taki fit () fonksiyonu kullanılarak yapılabilir.Bu fonksiyon da bir takım parametreler almaktadır.

batch_size: Bu parametre modelin eğitilmesi aşamasında aynı anda kaç adet verinin işleneceği anlamına gelir.Burada 512 olarak belirttim.

epoch : Bu değer, model eğitilirken verilerin modelden kaç kez geçiş yapacağını belirtir.Bu projede epoch değeri 3 olarak belirlendi.

validation_data : Doğrulama seti olarak bilinir. Bu parametreye veri setimizden test için ayırdığımız değerleri veriyoruz.

```
history = model.fit(x_train, y_train ,validation_data=(x_test, y_test),
                    epochs=3, batch_size=512)
```

Şekil 36: Glove ve Lstm kullanımı

C. FastText ve CNN Kullanımı

Adım 1 : Eğitim için kullanılacak olan veri setimizi projeye dahil etmeye çalışıyoruz.Veri setini proje içine dahil edebilmek için Pandas kütüphanesini kullandım. Bu kütüphanede ki read_csv() fonksiyonu ile bilgisayarınızdan ilgili bölümden veri setini çağırıyoruz.

```
import pandas as pd
data=pd.read_csv("oil_etiket.txt",sep=",")
print(data)
```

```
                                                    NEWS CLASS
0      Kuwait National Petroleum Co KNPC Thursday ann...   pos
1      Russia prepared possible drop oil prices OPEC ...   pos
2      The following changes OPEC 's output targets d...   pos
3      The latest list top places invest mining domin...   neg
4      Iran Thursday stood decision deny UN nuclear i...   pos
...                                                  ...   ...
20214  Pembina Pipeline Corp said Wednesday would buy...   pos
20215  In west Texas center US oil boom 3800 students...   neg
20216  Iranian Foreign Minister Mohammad Javad Zarif ...   pos
20217  Noble Group Holdings Noble Holdings plans rebu...   pos
20218  Iranian Foreign Minister Mohammad Javad Zarif ...   pos
```

Şekil 37: FastText ve CNN kullanımı

Adım 2 : Yapay sinir ağını oluşturmak için ve fasttext için gerekli kütüphaneler programa dahil edildi.

```
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense,LSTM,SimpleRNN,Conv1D,MaxPool1D,GlobalMaxPool1D,Activation
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from gensim.models.word2vec import Word2Vec
from keras.utils import to_categorical
import re
import nltk
from nltk.corpus import stopwords
from sklearn import preprocessing
from gensim.parsing.preprocessing import remove_stopwords
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import matplotlib.pyplot as plt
```

Şekil 38: FastText ve CNN kullanımı

Adım 3 : FastText modelini kullanabilmek için gensim kütüphanesinden Fasttext sınıfını projeye dahil ediyoruz.Bu sınıftan oluşturduğumuz obje bir takım parametreler almaktadır.Bu parametreler şunlardır :

words: Gensim text verisini girdi olarak list of list olacak şekilde alır.
size: Embedding vector boyutu.Bu çalışmada 100 olarak belirlendi.
window: Sağında ve solunda kaç kelime olacağı.Bu çalışmada 3 olarak belirlendi.
min_count: Belli bir sayının altında geçen kelimeleri almaması
workers: Thread sayısı.Aynı anda çalışacak iş parçacığı sayısıdır.
sg: 0 sa Skip-Gram 1 ise CBOW. Bu çalışmada cbow kullanılmıştır.

```
x=data['NEWS']
words=[]
for i in x:
    words.append(i.split())
```

```
from gensim.models import FastText
model_ted = FastText(words, size=100, window=3, min_count=1, workers=4,sg=1)
```

Şekil 39: FastText ve CNN kullanımı

Adım 4 : Tokenizer sınıfından bir obje oluşturuldu ve fit fonksiyonu ile veri setinde ki metinlerden bir kelime havuzu oluşturuldu. Metinleri "text_to_ sequences" fonksiyonu ile nümerik değerlerden oluşan bir vektöre çeviriyoruz.Nümerik değerlerden oluşturulan vektörün ysa algoritmalarında kullanabilmek için eşit uzunlukta ki vektörlere dönüştürüyoruz. Bu işlemi pad_sequences() fonksiyonu ile yapıyoruz.

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(x)
x= tokenizer.texts_to_sequences(x)
x=pad_sequences(x)
```

Şekil 40: FastText ve CNN kullanımı

Adım 5 : Hedef değişken yani sınıf etiketi kategorik. Onu nümerik hale getirmeliyiz. Hatta gölge değişken oluşturmalıyız..**"np_utils"** sınıfının bir fonksiyonu olan **"to_caterogical"** fonksiyonu çıkış katmanından yani sınıflarımızın olduğu katmandan 2 farklı rakam yerine 0 veya 1 değerlerinden oluşan bir vektör elde ediyoruz.Bu sayede doğru rakama karşılık gelen etiketin indeksi 1 iken diğer tüm etiketler için bu indeks 0 değerini alır.

```
y=data['CLASS']
from sklearn.preprocessing import LabelEncoder
label_encoder =LabelEncoder()
y = label_encoder.fit_transform(y)
y=to_categorical(y)
pd.DataFrame(y).sample(10)
```

Şekil 41: FastText ve CNN kullanımı

Adım 6 : Veri setini eğitim ve test için ikiye bölüyoruz. Bu aşamada veri setinin %20 sini test için %80 ni eğitim için ayırdık.

```
x_train, x_test, y_train, y_test = train_test_split(x,y,
                                   test_size = 0.20,random_state=0)
```

Şekil 42: FastText ve CNN kullanımı

Adım 7 : Yapay sinir ağı iki farklı şekilde başlatılabilir: 1. Katmanlar dizilimi (Sequential layers) olarak. 2. Graph olarak. Bu projede ilk yöntem kullanıldı. Bu yöntem için sadece yukarıda indirilen Sequential sınıfından nesne yaratıldı.Bu Sıralı katmanlardan oluşan bir modeldir.Bu aşamada modelimizi oluşturmaya başlandı.

embedding: Her bir kelimenin kaç boyutlu vektör olarak tanımlandığını belirtilir .Böylelikle giriş katmanında ki girdi yani nöron sayısını belirlemiş oluruz.

Conv1D: Bu katman, bir çıkış nöronu üretmek için tek bir uzamsal (veya zamansal) boyut üzerinde katman girdisiyle kıvrılan bir evrişim çekirdeği oluşturur.

GlobalMaxPooling1D: Maksimum havuz oluşturma katmanı, filtrenin kapsadığı özellik haritasının bölgesinden maksimum öğeyi seçen bir havuzlama işlemidir.

Dense: Dense objesi ile bütün giriş ve çıkış katmanın özelliklerini tanımlayabiliriz.

İlk olarak embedding ile giriş katmanını oluşturup verilerimizi bu katmandan veriyoruz.. İlk gizli katmanımız ise Conv1D katmanıdır. .Bu katmanda 128 tane nöron yer almaktadır.Yapay sinir ağlarında doğrusal olmayan gerçek dünya özelliklerini tanıtmak için aktivasyon fonksiyonuna ihtiyaç duyarız. Bu çalışmada aktivasyon fonksiyonu olarak "relu" kullanıldı. Bu katmanından sonra bir tane GlobalMaxPooling1D katmanı eklendi. Son eklediğimiz katman ise çıktı katmanıdır.Çıktı katmanındaki nöron sayısını veri setimizde ki sınıf sayısı kadar veriyoruz. Aktivasyon fonksiyonu olarak sigmoid kullanıldı.

```
from keras import layers
model = Sequential()
model.add(model_ted.wv.get_keras_embedding(True))
model.add(layers.Conv1D(128, 5, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(2, activation='sigmoid'))
model.compile(optimizer='adam',loss='binary_crossentropy', metrics=['accuracy'])
print(model.summary())
history = model.fit(x_train, y_train,
                    epochs=3,
                    verbose=True,
                    validation_data=(x_test, y_test),
                    batch_size=512)
```

Şekil 43: FastText ve CNN kullanımı

Model yapılandırıldıktan sonra eğitim sürecine başlayabiliriz. Bu, Keras'taki fit () fonksiyonu kullanılarak yapılabilir.Bu fonksiyon da bir takım parametreler almaktadır.

batch_size: Bu parametre modelin eğitilmesi aşamasında aynı anda kaç adet verinin işleneceği anlamına gelir.Burada 512 olarak belirttim.

epoch : Bu değer, model eğitilirken verilerin modelden kaç kez geçiş yapacağını belirtir.Bu projede epoch değeri 3 olarak belirlendi.

validation_data : Doğrulama seti olarak bilinir. Bu parametreye veri setimizden test için ayırdığımız değerleri veriyoruz.

D.Tf-idf ve RNN Kullanımı

Adım 1 : Eğitim için kullanılacak olan veri setimizi projeye dahil etmeye çalışıyoruz.Veri setini proje içine dahil edebilmek için Pandas kütüphanesini kullandım. Bu kütüphanede ki read_csv() fonksiyonu ile bilgisayarınızdan ilgili bölümden veri setini çağırıyoruz.

```
import pandas as pd
data=pd.read_csv("oil_etiket.txt",sep=",")
print(data)
```

```
                                                    NEWS CLASS
0      Kuwait National Petroleum Co KNPC Thursday ann...   pos
1      Russia prepared possible drop oil prices OPEC ...   pos
2      The following changes OPEC 's output targets d...   pos
3      The latest list top places invest mining domin...   neg
4      Iran Thursday stood decision deny UN nuclear i...   pos
...                                                  ...   ...
20214  Pembina Pipeline Corp said Wednesday would buy...   pos
20215  In west Texas center US oil boom 3800 students...   neg
20216  Iranian Foreign Minister Mohammad Javad Zarif ...   pos
20217  Noble Group Holdings Noble Holdings plans rebu...   pos
20218  Iranian Foreign Minister Mohammad Javad Zarif ...   pos

[20219 rows x 2 columns]
```

Şekil 44:Tf-idf ve RNN kullanımı

Adım 2 : İlk olarak keras kütüphanesinin tokenizer sınıfı projeye import edildi. Tokenizer sınıfından oluşturduğum obje ile veri setimizdeki cümlelerde ki en sık geçen kelimelerden oluşan bir kelime havuzu oluşturuldu. Num_words parametresi kelime havuzunun boyutudur. Tokenizer sınıfının text_to_matrix() fonksiyonun mode özelliğiyle veri setimizi dönüştürmek istediğimiz vektör uzayına çeviriyoruz. Mode özelliği tf-idf değeri seçildi. Bu parametrenin değerini belirledikten sonra text_to_matrix() fonksiyonun aldığı bir diğer parametre ise habelerimizin yani metinlerimizin olduğu dönüşüm yapılacak hedef değişkendir. Bu değeride girdikten sonra veri seti tf-idf vektörüne dönüştürme işlemi de tamamlanmış oldu. Yapay sinir ağı algoritmalarına bu vektörü direkt olarak girdi olarak veremiyoruz. pad_sequences() fonksiyonu ile vektörün tüm elemanlarını eşit uzunlukta ayarlıyoruz.

```
from keras.preprocessing.text import Tokenizer
MAX_NUM_WORDS = 650
num_words=MAX_NUM_WORDS
maxlen=650
tokenizer = Tokenizer(num_words=MAX_NUM_WORDS)
tokenizer.fit_on_texts(data['NEWS'])
X=tokenizer.texts_to_matrix(data['NEWS'],mode="tfidf")
```

```
from keras.preprocessing.sequence import pad_sequences
X = pad_sequences(X)
```

Şekil 45:Tf-idf ve RNN kullanımı

Adım 3 : Hedef değişken yani sınıf etiketi kategorik. Onu nümerik hale getirmeliyiz. Hatta gölge değişken oluşturmalıyız..**"np_utils"** sınıfının bir fonksiyonu olan **"to_caterogical"** fonksiyonu çıkış katmanından yani sınıflarımızın olduğu katmandan 2 farklı rakam yerine 0 veya 1 değerlerinden oluşan bir vektör elde ediyoruz.Bu sayede doğru rakama karşılık gelen etiketin indeksi 1 iken diğer tüm etiketler için bu indeks 0 değerini alır.

```
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
label_encoder =LabelEncoder()
Y = label_encoder.fit_transform(data['CLASS'])
Y=to_categorical(Y)
pd.DataFrame(Y).sample(10)
```

Şekil 46:Tf-idf ve RNN kullanımı

Adım 4 : Veri setini eğitim ve test için ikiye bölüyoruz. Bu aşamada veri setinin %20 sini test için %80 ni eğitim için ayırdık.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X,Y,test_size = 0.20,random_state=0)
```

Şekil 47:Tf-idf ve RNN kullanımı

Adım 5 : Yapay sinir ağı iki farklı şekilde başlatılabilir: 1. Katmanlar dizilimi (Sequential layers) olarak. 2. Graph olarak. Bu projede ilk yöntem kullanıldı. Bu yöntem için sadece yukarıda indirilen Sequential sınıfından nesne yaratıldı.Bu Sıralı katmanlardan oluşan bir modeldir.Bu aşamada modelimizi oluşturmaya başlandı.

embedding: Her bir kelimenin kaç boyutlu vektör olarak tanımlandığını belirtilir .Böylelikle giriş katmanında ki girdi yani nöron sayısını belirlemiş oluruz. num_words 650 kelimeden oluşan sözlüğümüzdür. input_length girişi uzunluğudur.

SimpleRNN: RNN algoritması katmanıdır.

Dense: Dense objesi ile bütün giriş ve çıkış katmanın özelliklerini tanımlayabiliriz.

return_sequences : Bu parametre bir sonraki katman YSA katmanlarından birisi ise true değeri almaktadır.

İlk olarak embedding ile giriş katmanını oluşturup verilerimizi bu katmandan veriyoruz.İlk gizli katmanımız ise SimpleRNN algoritması çalışmaktadır.Bu katmanda 128 tane nöron yer

almaktadır.Yapay sinir ağlarında doğrusal olmayan gerçek dünya özelliklerini tanıtmak için aktivasyon fonksiyonuna ihtiyaç duyarız. Bu çalışmada aktivasyon fonksiyonu olarak "tanh" kullanıldı.Fonksiyonun aralığı bu kez (-1,+1) olarak tanımlanmaktadır. Sigmoid fonksiyonuna göre avantajı ise türevinin daha dik olması yani daha çok değer alabilmesidir.

İkinci gizli katman da SimpleRNN algoritması ile çalışmaktadır.Katmanda 64 tane nöron yer almaktadır.Bu çalışmada aktivasyon fonksiyonu olarak "tanh" kullanıldı.

Son eklediğimiz katman ise çıktı katmanıdır. Çıktı katmanındaki nöron sayısını veri setimizde ki sınıf sayısı kadar veriyoruz. Çoklu sınıflandırma problemi üzerine çalıştığımız için "softmax" aktivasyon fonksiyonunu kullanıyoruz.

```
from keras.layers import Flatten, Dense, SimpleRNN, LSTM, Dropout,Activation
model = Sequential()
model.add(Embedding(num_words,16,input_length=X.shape[1]))
model.add( SimpleRNN(128,activation='tanh',return_sequences=True))
model.add( SimpleRNN(64,activation='tanh',return_sequences=False))
model.add(Dense(2, activation='softmax'))
model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
```

Şekil 48:Tf-idf ve RNN kullanımı

Bir sonraki aşamada modelimizi compile fonksiyonu ile derlemeye başlıyoruz.

optimizer: En iyileme algoritmaları, kayıp fonksiyonuna göre ağın nasıl güncelleneceğini belirler. Optimizer, öğrenme oranını kontrol eder.Bu projede ağırlık katsayılarının güncellenmesi için kullanılacak optimizasyon yöntemi 'adam' dır.

loss: Gerçek değer ile tahmin edilen değer arasındaki hatayı ifade eden metrik. Temelde "binary_crossentropy" , ikili doğruluğu gösterir. Çok sınıflı sınıflandırma problemleri için kullanılır.

metrics: Model değerlendirme kriterleri belirlenir. Eğitim ve test sırasında değerlendirmek istediğim metrik accuracy (doğruluk) değeridir.

Adım 6 : Model yapılandırıldıktan sonra eğitim sürecine başlayabiliriz. Bu, Keras'taki fit () fonksiyonu kullanılarak yapılabilir.Bu fonksiyon da bir takım parametreler almaktadır.

batch_size: Bu parametre modelin eğitilmesi aşamasında aynı anda kaç adet verinin işleneceği anlamına gelir. Burada 512 olarak belirttim.

epoch : Bu değer, model eğitilirken verilerin modelden kaç kez geçiş yapacağını belirtir.Bu projede epoch değeri 3 olarak belirlendi.

validation_data : Doğrulama seti olarak bilinir. Bu parametreye veri setimizden test için ayırdığımız değerleri veriyoruz.

```
history = model.fit(x_train, y_train ,validation_data=(x_test, y_test), epochs=3, batch_size=512)
```

Şekil 49:Tf-idf ve RNN kullanımı

E. Binary ve LSTM Kullanımı

Adım 1: Eğitim için kullanılacak olan veri setimizi projeye dahil etmeye çalışıyoruz.Veri setini proje içine dahil edebilmek için Pandas kütüphanesini kullandım. Bu kütüphanede ki read_csv() fonksiyonu ile bilgisayarınızdan ilgili bölümden veri setini çağırıyoruz.

```
import pandas as pd
data=pd.read_csv("oil_etiket.txt",sep=",")
print(data)
```

```
                                                  NEWS CLASS
0      Kuwait National Petroleum Co KNPC Thursday ann...   pos
1      Russia prepared possible drop oil prices OPEC ...   pos
2      The following changes OPEC 's output targets d...   pos
3      The latest list top places invest mining domin...   neg
4      Iran Thursday stood decision deny UN nuclear i...   pos
...                                                  ...   ...
20214  Pembina Pipeline Corp said Wednesday would buy...   pos
20215  In west Texas center US oil boom 3800 students...   neg
20216  Iranian Foreign Minister Mohammad Javad Zarif ...   pos
20217  Noble Group Holdings Noble Holdings plans rebu...   pos
20218  Iranian Foreign Minister Mohammad Javad Zarif ...   pos

[20219 rows x 2 columns]
```

Şekil 50:Binary ve Lstm kullanımı

Adım 2: İlk olarak keras kütüphanesinin tokenizer sınıfı projeye import edildi. Tokenizer sınıfından oluşturduğum obje ile veri setimizdeki cümlelerde ki en sık geçen kelimelerden oluşan bir kelime havuzu oluşturuldu. Num_words parametresi kelime havuzunun boyutudur.
Tokenizer sınıfının text_to_matrix() fonksiyonun mode özelliğiyle veri setimizi dönüştürmek istediğimiz vektör uzayına çeviriyoruz. Mode özelliğinin binary olarak seçildi ve veri setinde ki metinler 1 ve 0 değerlerinden oluşan bir vektör uzayına dönüştürüldü. Yapay sinir ağı algoritmalarına bu vektörü direkt olarak girdi olarak veremiyoruz. pad_sequences() fonksiyonu ile

vektörün tüm elemanlarını eşit uzunlukta ayarlıyoruz.

```
from keras.preprocessing.text import Tokenizer
MAX_NUM_WORDS = 700
num_words=MAX_NUM_WORDS
maxlen=700
tokenizer = Tokenizer(num_words=MAX_NUM_WORDS)
tokenizer.fit_on_texts(data['NEWS'])
X=tokenizer.texts_to_matrix(data['NEWS'],mode="binary")
```

```
from keras.preprocessing.sequence import pad_sequences
X = pad_sequences(X)
```

Şekil 51:Binary ve Lstm kullanımı

Adım 3: Hedef değişken yani sınıf etiketi kategorik. Onu nümerik hale getirmeliyiz. Hatta gölge değişken oluşturmalıyız..”np_utils“ sınıfının bir fonksiyonu olan “to_caterogical” fonksiyonu çıkış katmanından yani sınıflarımızın olduğu katmandan 2 farklı rakam yerine 0 veya 1 değerlerinden oluşan bir vektör elde ediyoruz.Bu sayede doğru rakama karşılık gelen etiketin indeksi 1 iken diğer tüm etiketler için bu indeks 0 değerini alır.

```
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
label_encoder =LabelEncoder()
Y = label_encoder.fit_transform(data['CLASS'])
Y=to_categorical(Y)
pd.DataFrame(Y).sample(10)
```

Şekil 52 : Binary ve Lstm kullanımı

Adım 4: Veri setini eğitim ve test için ikiye bölüyoruz. Bu aşamada veri setinin %20 sini test için %80 ni eğitim için ayırdık.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X,Y,test_size = 0.20,random_state=0)
```

Şekil 53: Binary ve Lstm kullanımı

Adım 5: Yapay sinir ağı iki farklı şekilde başlatılabilir: 1. Katmanlar dizilimi (Sequential layers) olarak. 2. Graph olarak. Bu projede ilk yöntem kullanıldı. Bu yöntem için sadece yukarıda indirilen Sequential sınıfından nesne yaratıldı.Bu Sıralı katmanlardan oluşan bir modeldir.Bu aşamada modelimizi oluşturmaya başlandı.

embedding: Her bir kelimenin kaç boyutlu vektör olarak tanımlandığını belirtilir .Böylelikle giriş katmanında ki girdi yani nöron sayısını belirlemiş oluruz. Num_words 650 kelimeden oluşan sözlüğümüzdür. Input_length girişi uzunluğudur.

dropout : Dropout katmanı her adımda belirtilen orandaki girdiyi rassal olarak sıfıra eşitleyerek modelin veriye aşırı uyum sağlamasının (aşırı öğrenmenin) önüne geçer.

Dense: Dense objesi ile bütün giriş ve çıkış katmanın özelliklerini tanımlayabiliriz.

return_sequences : Bu parametre bir sonraki katman YSA katmanlarından birisi ise true değeri almaktadır.

İlk olarak embedding ile giriş katmanını oluşturup verilerimizi bu katmandan veriyoruz. Bu katmandan sonra ezberlemenin önüne geçmesi için dropout katmanı kullanılmıştır. İlk gizli katmanımız ise Lstm algoritması çalışmaktadır.Bu katmanda 128 tane nöron yer almaktadır.Yapay sinir ağlarında doğrusal olmayan gerçek dünya özelliklerini tanıtmak için aktivasyon fonksiyonuna ihtiyaç duyarız. Bu çalışmada aktivasyon fonksiyonu olarak “tanh” kullanıldı.Fonksiyonun aralığı bu kez (-1,+1) olarak tanımlanmaktadır. Sigmoid fonksiyonuna göre avantajı ise türevinin daha dik olması yani daha çok değer alabilmesidir.

Lstm katmanından sonra bir tane daha dropout katmanı kullanılmıştır.

Son eklediğimiz katman ise çıktı katmanıdır.Çıktı katmanındaki nöron sayısını veri setimizde ki sınıf sayısı kadar veriyoruz. Çoklu sınıflandırma problemi üzerine çalıştığımız için "softmax" aktivasyon fonksiyonunu kullanıyoruz.

```
#LSTM ALGORİTMASI
from keras.models import Sequential
from keras.layers import Flatten, Dense, SimpleRNN, LSTM, Dropout,Activation
from keras.layers.embeddings import Embedding
model = Sequential()
model.add(Embedding(num_words,24,input_length=X.shape[1]))
model.add(Dropout(0.5))
model.add(LSTM(128,activation='tanh',return_sequences=False))
model.add(Dropout(0.5))
model.add(Dense(2, activation='softmax'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])
model.summary()

history = model.fit(x_train, y_train,
                    epochs=3,
                    batch_size=512,
                    validation_data=(x_test, y_test))
```

Şekil 54: Binary ve Lstm kullanımı

Bir sonraki aşamada modelimizi compile fonksiyonu ile derlemeye başlıyoruz.

optimizer : En iyileme algoritmaları, kayıp fonksiyonuna göre ağın nasıl güncelleneceğini

belirler. Optimizer, öğrenme oranını kontrol eder.Bu projede ağırlık katsayılarının güncellenmesi için kullanılacak optimizasyon yöntemi **'adam'** dır.

loss : Gerçek değer ile tahmin edilen değer arasındaki hatayı ifade eden metrik. Temelde **"binary_crossentropy"** , ikili doğruluğu gösterir. Çok sınıflı sınıflandırma problemleri için kullanılır.

metrics: Model değerlendirme kriterleri belirlenir. Eğitim ve test sırasında değerlendirmek istediğim metrik accuracy (doğruluk) değeridir.

Model yapılandırıldıktan sonra eğitim sürecine başlayabiliriz. Bu, Keras'taki fit () fonksiyonu kullanılarak yapılabilir.Bu fonksiyon da bir takım parametreler almaktadır.

batch_size: Bu parametre modelin eğitilmesi aşamasında aynı anda kaç adet verinin işleneceği anlamına gelir. Burada 512 olarak belirttim.

epoch : Bu değer, model eğitilirken verilerin modelden kaç kez geçiş yapacağını belirtir.Bu projede epoch değeri 3 olarak belirlendi.

validation_data : Doğrulama seti olarak bilinir. Bu parametreye veri setimizden test için ayırdığımız değerleri veriyoruz.

F. TF ve CNN Kullanımı

Adım 1 : Eğitim için kullanılacak olan veri setimizi projeye dahil etmeye çalışıyoruz.Veri setini proje içine dahil edebilmek için Pandas kütüphanesini kullandım. Bu kütüphanede ki read_csv() fonksiyonu ile bilgisayarınızdan ilgili bölümden veri setini çağırıyoruz.

```
import pandas as pd
data=pd.read_csv("oil_etiket.txt",sep=",")
print(data)
```

```
                                                    NEWS CLASS
0      Kuwait National Petroleum Co KNPC Thursday ann...   pos
1      Russia prepared possible drop oil prices OPEC ...   pos
2      The following changes OPEC 's output targets d...   pos
3      The latest list top places invest mining domin...   neg
4      Iran Thursday stood decision deny UN nuclear i...   pos
...                                                  ...   ...
20214  Pembina Pipeline Corp said Wednesday would buy...   pos
20215  In west Texas center US oil boom 3800 students...   neg
20216  Iranian Foreign Minister Mohammad Javad Zarif ...   pos
20217  Noble Group Holdings Noble Holdings plans rebu...   pos
20218  Iranian Foreign Minister Mohammad Javad Zarif ...   pos

[20219 rows x 2 columns]
```

Şekil 55: TF ve CNN kullanımı

Adım 2 : Sklearn TF-IDF metodu için TdidfVectorizer sınıfını sağlamaktadır. İlk olarak TdidfVectorizer sınıfını kullanabilmek için bu ilgili kütüphaneyi dahil ediyoruz. Daha sonra TfidfVectorizer fonksiyonu çağırıyoruz. Bu fonksiyonda sadece tf değerini hesaplatmak için bir takım parametreler almaktadır. Vektör oluştururken sadece tf değerini hesaplanması için use_idf parametresinin false değerini alması gerekmektedir. Max_feauters özelliği ile de kelimelerin tf değerlerinden bir havuz oluşturuyoruz. Daha sonra , ilgili sınıftan oluşturduğumuz objenin fit ve transform fonksiyonu ile cümlelerdeki kelimelerin tf değerlerini içeren bir vektör oluşması sağlandı.

```
vectorizer = TfidfVectorizer(max_features=650,use_idf=False)
vectorizer = vectorizer.fit(data['NEWS'])
```

```
df = vectorizer.transform(data['NEWS'])
```

Şekil 56: TF ve CNN kullanımı

Adım 3: Hedef değişken yani sınıf etiketi kategorik. Onu nümerik hale getirmeliyiz. Hatta gölge değişken oluşturmalıyız.."np_utils" sınıfının bir fonksiyonu olan "to_caterogical" fonksiyonu çıkış katmanından yani sınıflarımızın olduğu katmandan 2 farklı rakam yerine 0 veya 1 değerlerinden oluşan bir vektör elde ediyoruz.Bu sayede doğru rakama karşılık gelen etiketin indeksi 1 iken diğer tüm etiketler için bu indeks 0 değerini alır.

```
from keras.utils import to_categorical
from sklearn.preprocessing import LabelEncoder
label_encoder =LabelEncoder()
Y = label_encoder.fit_transform(data['CLASS'])
Y=to_categorical(Y)
pd.DataFrame(Y).sample(10)
```

Şekil 57: TF ve CNN kullanımı

Adım 4: Veri setini eğitim ve test için ikiye bölüyoruz. Bu aşamada veri setinin %20 sini test için %80 ni eğitim için ayırdık.

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X,Y,test_size = 0.20,random_state=0)
```

Şekil 58: TF ve CNN kullanımı

Adım 5: Yapay sinir ağı iki farklı şekilde başlatılabilir: 1. Katmanlar dizilimi (Sequential layers) olarak. 2. Graph olarak. Bu projede ilk yöntem kullanıldı. Bu yöntem için sadece yukarıda

indirilen Sequential sınıfından nesne yaratıldı.Bu Sıralı katmanlardan oluşan bir modeldir.Bu aşamada modelimizi oluşturmaya başlandı.
embedding: Her bir kelimenin kaç boyutlu vektör olarak tanımlandığını belirtilir .Böylelikle giriş katmanında ki girdi yani nöron sayısını belirlemiş oluruz. İlk parametre kelime havuzunun uzunluğudur. Input_length giriş vektörünün boyutudur..

Conv1D: Bu katman, bir çıkış nöronu üretmek için tek bir uzamsal (veya zamansal) boyut üzerinde katman girdisiyle kıvrılan bir evrişim çekirdeği oluşturur.

GlobalMaxPooling1D: Maksimum havuz oluşturma katmanı, filtrenin kapsadığı özellik haritasının bölgesinden maksimum öğeyi seçen bir havuzlama işlemidir.

Dense: Dense objesi ile bütün giriş ve çıkış katmanın özelliklerini tanımlayabiliriz.

İlk olarak embedding ile giriş katmanını oluşturup verilerimizi bu katmandan veriyoruz.. İlk gizli katmanımız ise Conv1D katmanıdır. .Bu katmanda 128 tane nöron yer almaktadır.Yapay sinir ağlarında doğrusal olmayan gerçek dünya özelliklerini tanıtmak için aktivasyon fonksiyonuna ihtiyaç duyarız. Bu çalışmada aktivasyon fonksiyonu olarak "relu" kullanıldı. Bu katmanından sonra bir tane GlobalMaxPooling1D katmanı eklendi. Son eklediğimiz katman ise çıktı katmanıdır.Çıktı katmanındaki nöron sayısını veri setimizde ki sınıf sayısı kadar veriyoruz. Aktivasyon fonksiyonu olarak sigmoid kullanıldı.

```
from keras import layers
model = Sequential()
model.add(layers.Embedding(num_words, 100, input_length=X.shape[1]))
model.add(layers.Conv1D(128, 5, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(10, activation='relu'))
model.add(layers.Dense(2, activation='sigmoid'))
model.compile(optimizer='adam',loss='binary_crossentropy', metrics=['accuracy'])
print(model.summary())
history = model.fit(x_train, y_train,
                    epochs=3,
                    verbose=True,
                    validation_data=(x_test, y_test),
                    batch_size=512)
```

Şekil 59: TF ve CNN kullanımı

Model yapılandırıldıktan sonra eğitim sürecine başlayabiliriz. Bu, Keras'taki fit () fonksiyonu kullanılarak yapılabilir.Bu fonksiyon da bir takım parametreler almaktadır.

batch_size: Bu parametre modelin eğitilmesi aşamasında aynı anda kaç adet verinin işleneceği anlamına gelir. Burada 512 olarak belirttim.

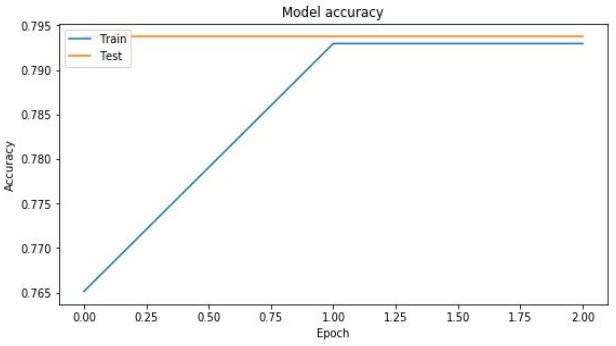
epoch : Bu değer, model eğitilirken verilerin modelden kaç kez geçiş yapacağını belirtir.Bu projede epoch değeri 3 olarak belirlendi.

validation_data : Doğrulama seti olarak bilinir. Bu parametreye veri setimizden test için ayırdığımız değerleri veriyoruz.

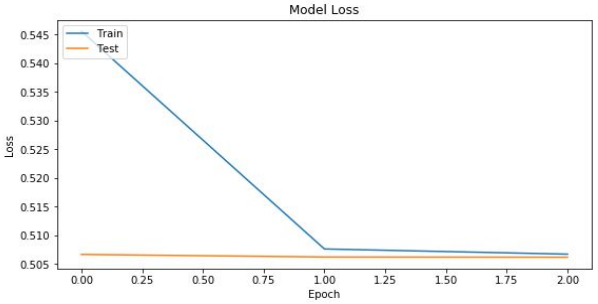
V. Sonuçlar

Gösterim Modeli: FASTTEXT Algoritma : CNN
Epoch : 3 Batch size : 512

TRAİN : %80 TEST: %20			
Accuracy	0.7836	**Validation Accuracy**	0.7938
Loss	0.5208	**Validation Loss**	0.5076



Şekil 60: FastText-CNN-TS:%20-ACC

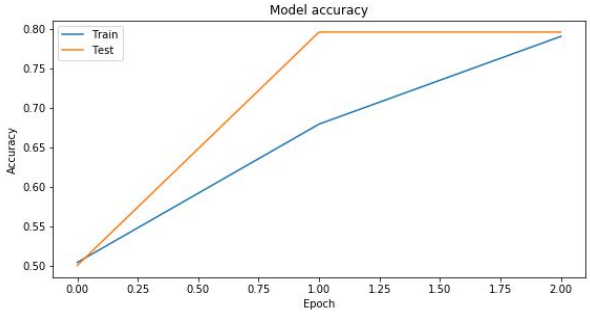


Şekil 61: FastText-CNN-TS:%20-LOSS

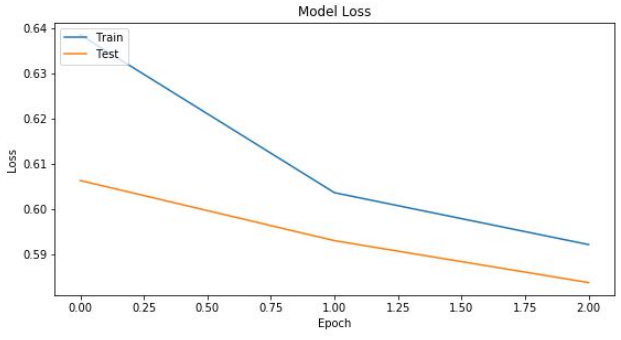
Precision	0.7937
Recall	0.7937
F1-Score	0.7937

Gösterim Modeli: FASTTEXT Algoritma : CNN
Epoch : 3 Batch size : 512

TRAİN : %50 TEST: %50			
Accuracy	0.7202	**Validation Accuracy**	0.7958
Loss	0.5606	**Validation Loss**	0.5057



Şekil 62: FastText-CNN-TS:%50-ACC

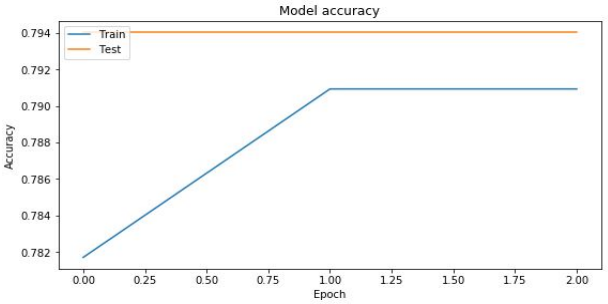


Şekil 63: FastText-CNN-TS:%50-LOSS

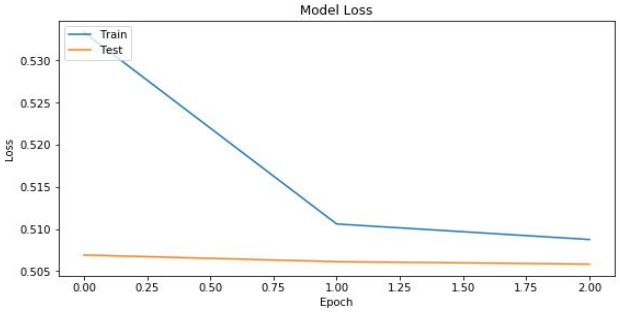
Precision	0.7958
Recall	0.7958
F1-Score	0.7958

Gösterim Modeli: FASTTEXT Algoritma : CNN
Epoch : 3 Batch size : 512

TRAİN : %30 TEST: %70			
Accuracy	0.7365	**Validation Accuracy**	0.7941
Loss	0.5612	**Validation Loss**	0.5270



Şekil 64: FastText-CNN-TS:%70-ACC

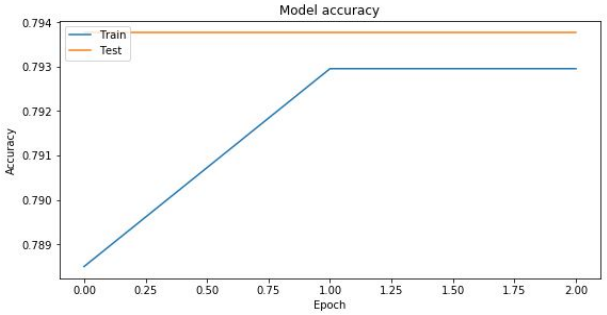


Şekil 65: FastText-CNN-TS:%70-LOSS

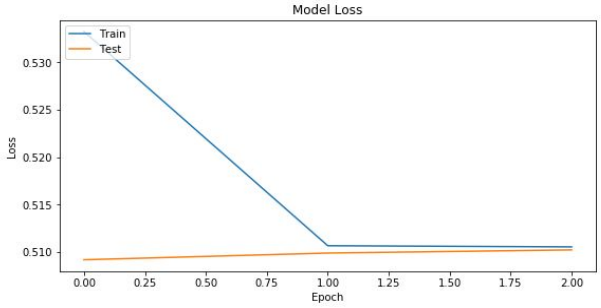
Precision	0.7940
Recall	0.7940
F1-Score	0.7940

Gösterim Modeli: FASTTEXT Algoritma : RNN
Epoch : 3 Batch size : 512

TRAİN : %80 TEST: %20			
Accuracy	0.7779	**Validation Accuracy**	0.7938
Loss	0.5300	**Validation Loss**	0.5100



Şekil 66: FastText-RNN-TS:%20-ACC

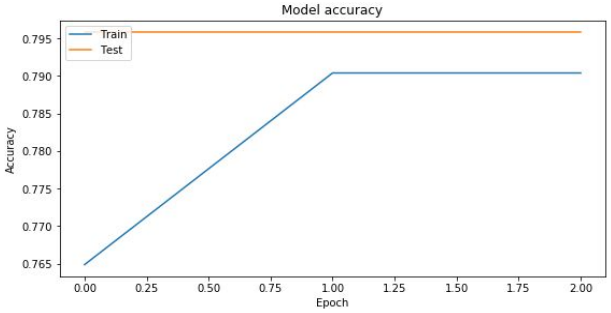


Şekil 67: FastText-RNN-TS:%20-LOSS

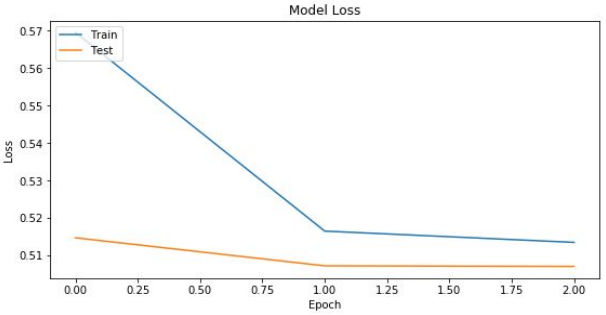
Precision	0.7937
Recall	0.7937
F1-Score	0.7937

Gösterim Modeli: FASTTEXT Algoritma : RNN Epoch : 3 Batch size : 512

TRAİN : %50 TEST: %50			
Accuracy	0.7904	**Validation Accuracy**	0.7958
Loss	0.5180	**Validation Loss**	0.5100



Şekil 68: FastText-RNN-TS:%50-ACC

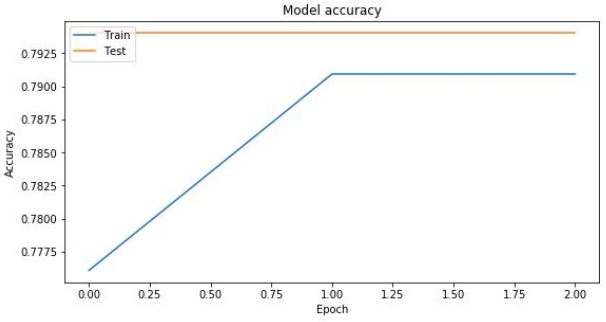


Şekil 69: FastText-RNN-TS:%50-LOSS

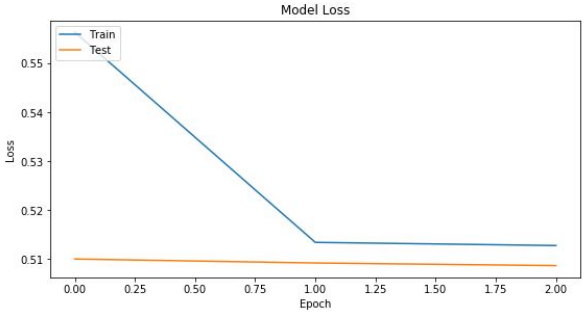
Precision	0.7958
Recall	0.7958
F1-Score	0.7958

Gösterim Modeli: FASTTEXT Algoritma : RNN Epoch : 3 Batch size : 512

TRAİN : %30 TEST: %70			
Accuracy	0.7909	**Validation Accuracy**	0.7941
Loss	0.5128	**Validation Loss**	0.5092



Şekil 70: FastText-RNN-TS:%70-ACC

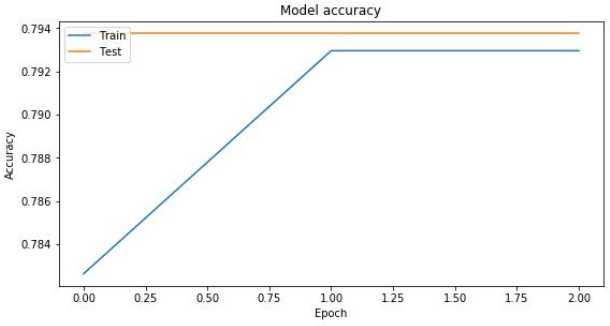


Şekil 71: FastText-RNN-TS:%70-LOSS

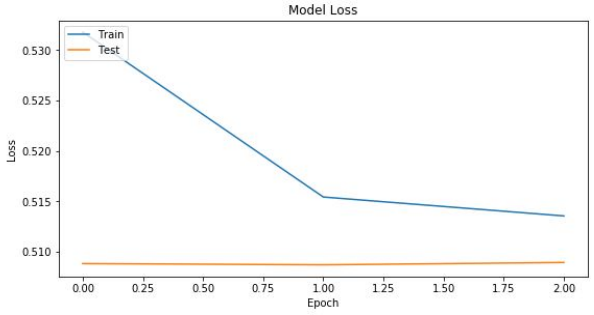
Precision	0.7940
Recall	0.7940
F1-Score	0.7940

Gösterim Modeli: FASTTEXT Algoritma: LSTM
Epoch : 3 Batch size : 512

TRAİN : %80 TEST: %20			
Accuracy	0.7930	**Validation Accuracy**	0.7938
Loss	0.5136	**Validation Loss**	0.5091



Şekil 72: FastText-Lstm-TS:%20-ACC

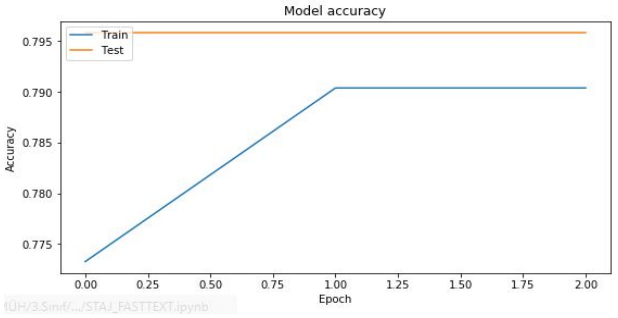


Şekil 73: FastText-Lstm-TS:%20-LOSS

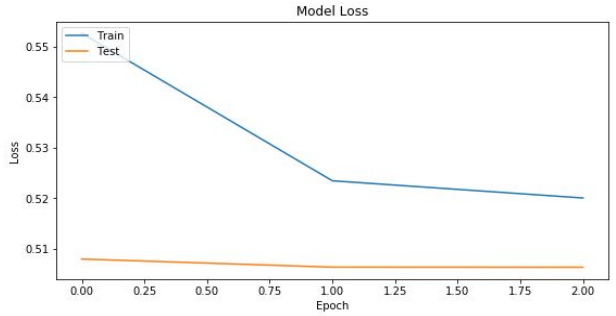
Precision	0.7937
Recall	0.7937
F1-Score	0.7937

Gösterim Modeli: FASTTEXT Algoritma: LSTM
Epoch : 3 Batch size : 512

TRAİN : %50 TEST: %50			
Accuracy	0.7904	**Validation Accuracy**	0.7958
Loss	0.5196	**Validation Loss**	0.5088



Şekil 74: FastText-Lstm-TS:%50-ACC

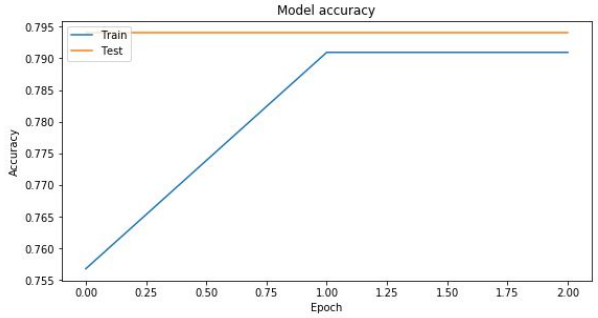


Şekil 75: FastText-Lstm-TS:%50-LOSS

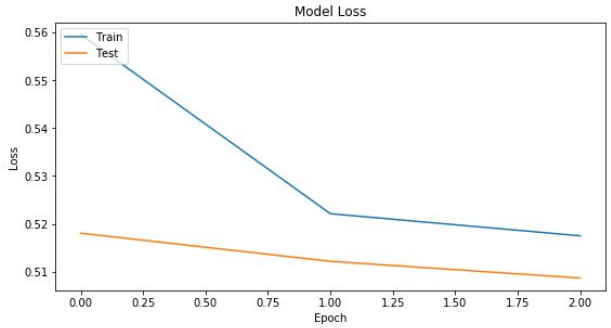
Precision	0.7958
Recall	0.7958
F1-Score	0.7958

Gösterim Modeli: FASTTEXT Algoritma: LSTM
Epoch : 3 Batch size : 512

TRAİN : %30 TEST: %70			
Accuracy	0.7909	**Validation Accuracy**	0.7941
Loss	0.5185	**Validation Loss**	0.5085



Şekil 76: FastText-Lstm-TS:%70-ACC

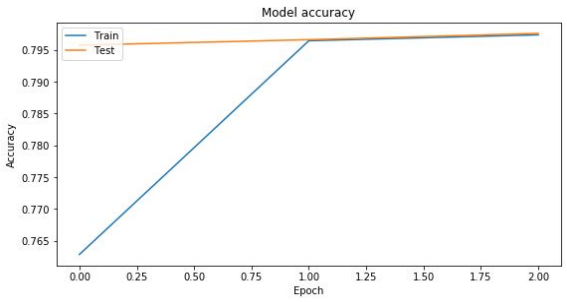


Şekil 77: FastText-Lstm-TS:%70-LOSS

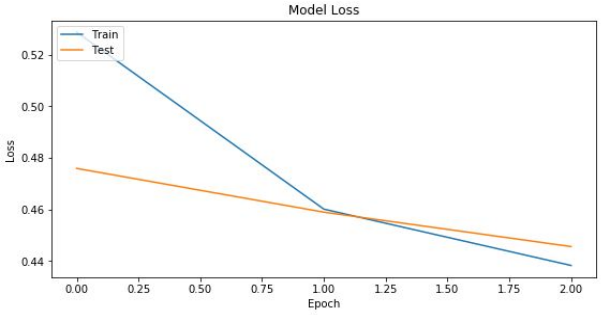
Precision	0.7940
Recall	0.7940
F1-Score	0.7940

Gösterim Modeli: GLOVE Algoritma: CNN
Epoch : 3 Batch size : 512

TRAİN : %80 TEST: %20			
Accuracy	0.8007	**Validation Accuracy**	0.4350
Loss	0.4237	**Validation Loss**	0.7993



Şekil 78: Glove-CNN-TS:%20-ACC

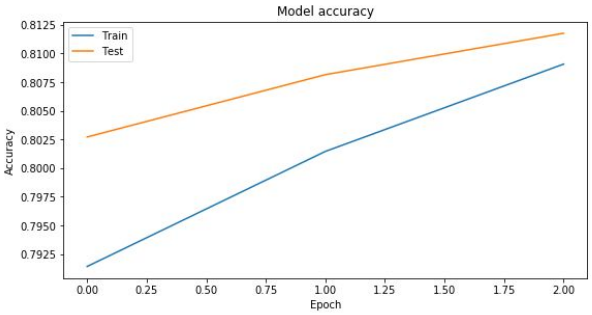


Şekil 79: Glove-CNN-TS:%20-LOSS

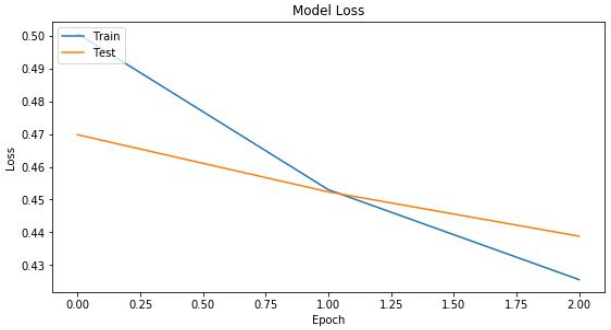
Precision	0.7937
Recall	0.8088
F1-Score	0.8012

Gösterim Modeli: GLOVE Algoritma: CNN
Epoch : 3 Batch size : 512

TRAİN : %50 TEST: %50			
Accuracy	0.7906	**Validation Accuracy**	0.7962
Loss	0.4484	**Validation Loss**	0.4529



Şekil 80: Glove-CNN-TS:%50-ACC

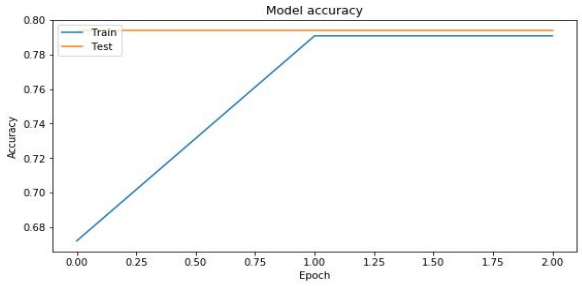


Şekil 81: Glove-CNN-TS:%50-LOSS

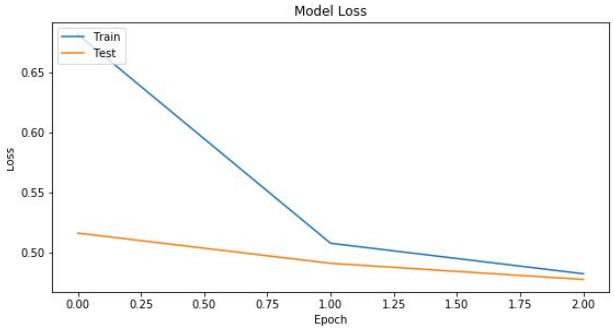
Precision	0.7961
Recall	0.7962
F1-Score	0.7962

Gösterim Modeli: GLOVE Algoritma: CNN
Epoch : 3 Batch size : 512

TRAİN : %30 TEST: %70			
Accuracy	0.7951	**Validation Accuracy**	0.7987
Loss	0.4755	**Validation Loss**	0.4725



Şekil 82: Glove-CNN-TS:%70-ACC

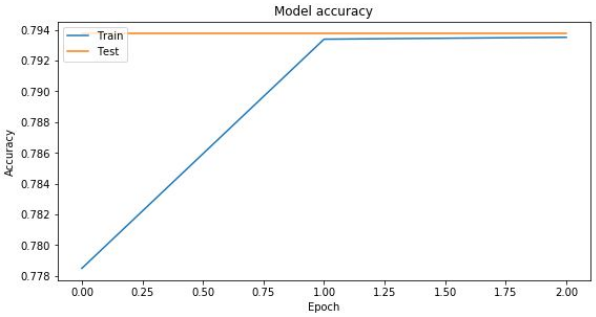


Şekil 83: Glove-CNN-TS:%70-LOSS

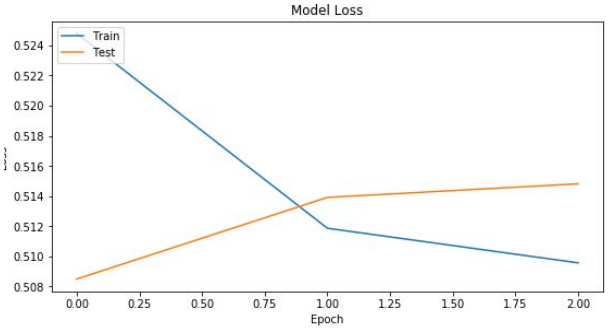
Precision	0.8036
Recall	0.7906
F1-Score	0.7970

Gösterim Modeli: GLOVE Algoritma: RNN
Epoch : 3 Batch size : 512

TRAİN : %80 TEST: %20			
Accuracy	0.7930	**Validation Accuracy**	0.7938
Loss	0.5092	**Validation Loss**	0.5046



Şekil 84: Glove-RNN-TS:%20-ACC

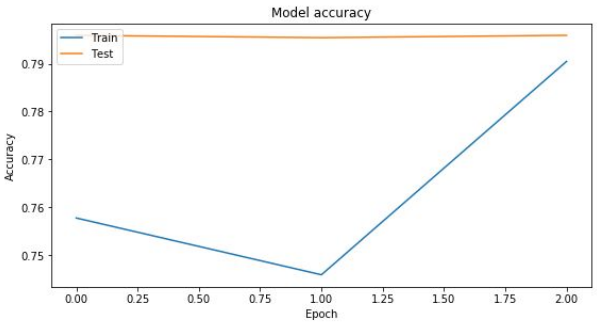


Şekil 85: Glove-RNN-TS:%20-LOSS

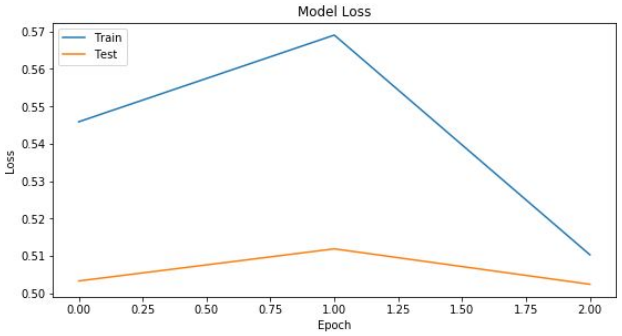
Precision	0.7937
Recall	0.7937
F1-Score	0.7937

Gösterim Modeli: GLOVE Algoritma: RNN
Epoch : 3 Batch size : 512

TRAİN : %50 TEST: %50			
Accuracy	0.7906	**Validation Accuracy**	0.7958
Loss	0.5106	**Validation Loss**	0.5021



Şekil 86: Glove-RNN-TS:%50-ACC

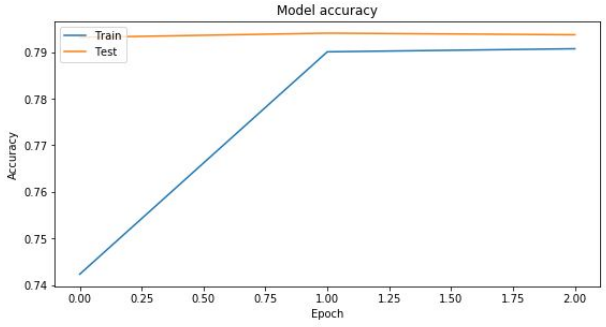


Şekil 87: Glove-RNN-TS:%50-LOSS

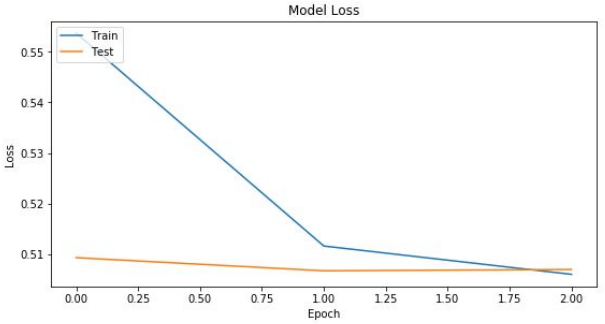
Precision	0.7958
Recall	0.7958
F1-Score	0.7958

Gösterim Modeli: GLOVE Algoritma: RNN
Epoch : 3 Batch size : 512

TRAİN : %30 TEST: %70			
Accuracy	0.7909	**Validation Accuracy**	0.7941
Loss	0.5128	**Validation Loss**	0.5241



Şekil 88: Glove-RNN-TS:%70-ACC

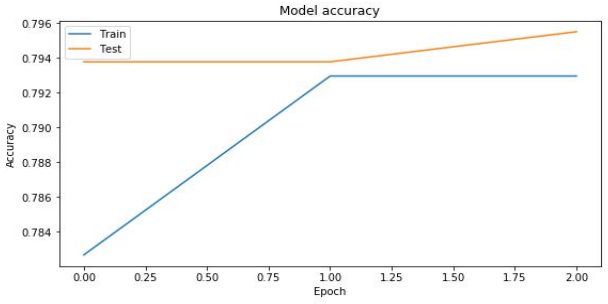


Şekil 89: Glove-RNN-TS:%70-LOSS

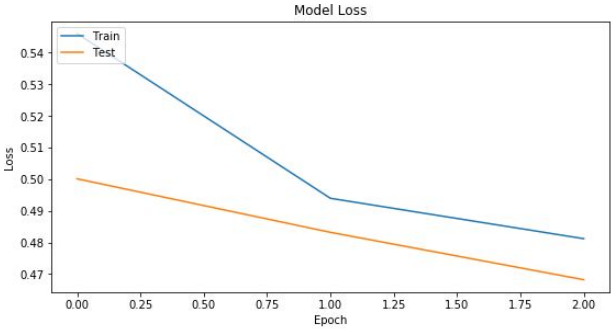
Precision	0.7941
Recall	0.7941
F1-Score	0.7941

Gösterim Modeli: GLOVE Algoritma: LSTM
Epoch : 3 Batch size : 512

TRAİN : %80 TEST: %20			
Accuracy	0.7926	**Validation Accuracy**	0.7938
Loss	0.4798	**Validation Loss**	0.4700



Şekil 90: Glove-Lstm-TS:%20-ACC

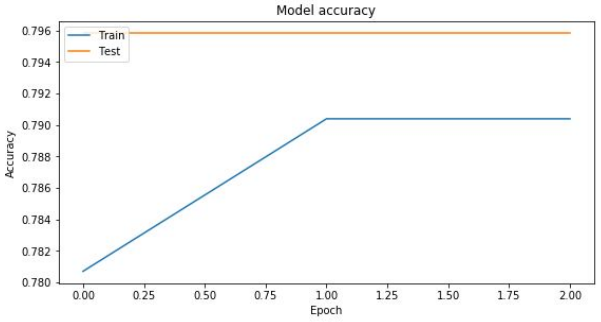


Şekil 91: Glove-Lstm-TS:%20-LOSS

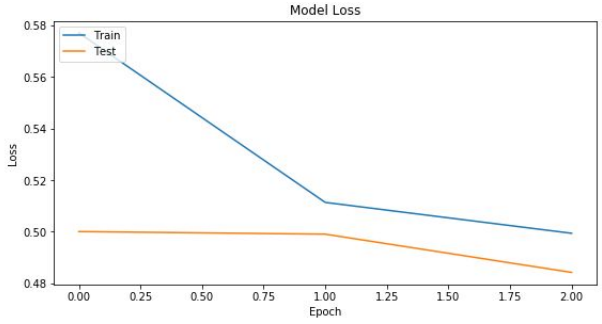
Precision	0.7937
Recall	0.7937
F1-Score	0.7937

Gösterim Modeli: GLOVE Algoritma: LSTM
Epoch : 3 Batch size : 512

TRAİN : %50 TEST: %50			
Accuracy	0.7904	**Validation Accuracy**	0.7958
Loss	0.4958	**Validation Loss**	0.4815



Şekil 92: Glove-Lstm-TS:%50-ACC

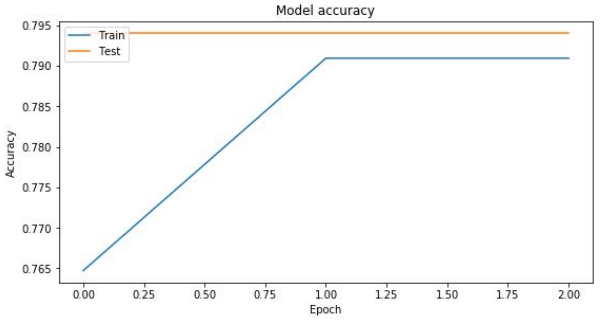


Şekil 93: Glove-Lstm-TS:%50-LOSS

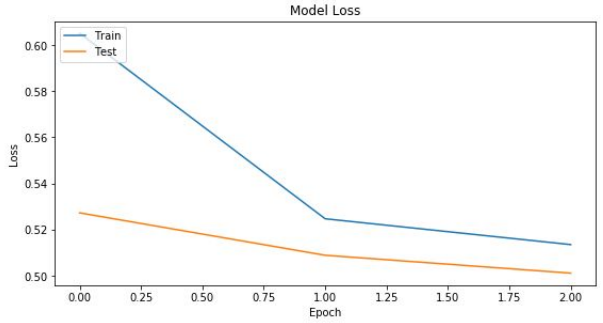
Precision	0.7958
Recall	0.7958
F1-Score	0.7958

Gösterim Modeli: GLOVE Algoritma: LSTM
Epoch : 3 Batch size : 512

TRAİN : %30 TEST: %70			
Accuracy	0.7909	**Validation Accuracy**	0.7941
Loss	0.5089	**Validation Loss**	0.5018



Şekil 94: Glove-Lstm-TS:%70-ACC

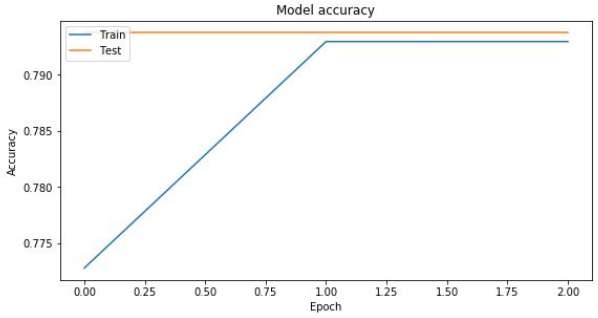


Şekil 95: Glove-Lstm-TS:%70-LOSS

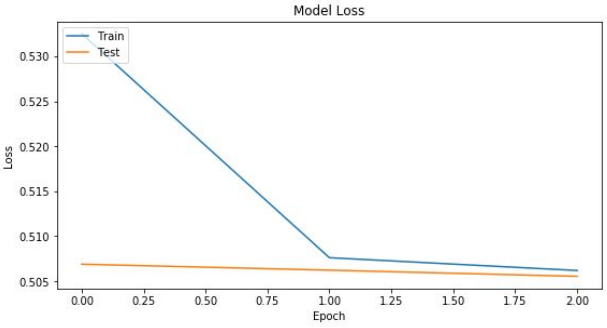
Precision	0.7940
Recall	0.7940
F1-Score	0.7940

Gösterim Modeli: Word2Vec Algoritma: CNN
Epoch : 3 Batch size : 512

TRAİN : %80 TEST: %20			
Accuracy	0.7930	**Validation Accuracy**	0.7938
Loss	0.5111	**Validation Loss**	0.5106



Şekil 96: Word2vec-CNN-TS:%20-ACC

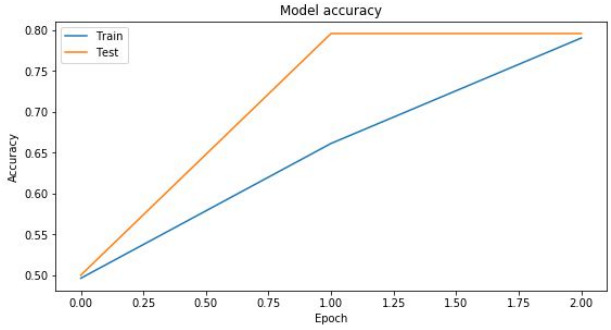


Şekil 97: Word2vec-CNN-TS:%20-LOSS

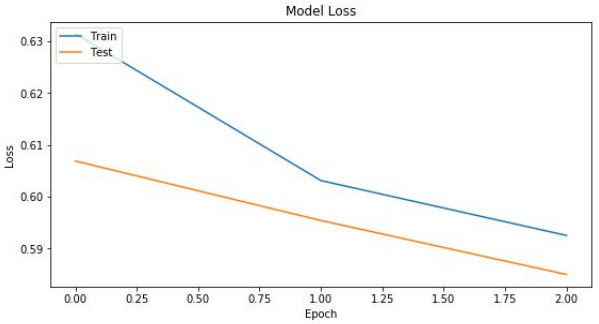
Precision	0.7937
Recall	0.7937
F1-Score	0.7937

Gösterim Modeli: Word2Vec Algoritma: CNN
Epoch : 3 Batch size : 512

TRAİN : %50 TEST: %50			
Accuracy	0.7904	**Validation Accuracy**	0.7958
Loss	0.5100	**Validation Loss**	0.5039



Şekil 98: Word2vec-CNN-TS:%50-ACC

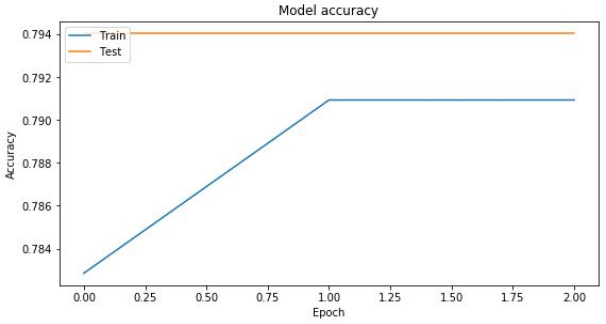


Şekil 99: Word2vec-CNN-TS:%50-LOSS

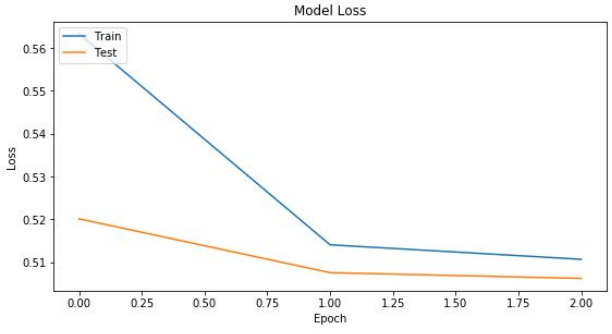
Precision	0.7958
Recall	0.7958
F1-Score	0.7958

Gösterim Modeli: Word2Vec Algoritma: CNN
Epoch : 3 Batch size : 512

TRAİN : %30 TEST: %70			
Accuracy	0.7909	**Validation Accuracy**	0.7941
Loss	0.5764	**Validation Loss**	0.5703



Şekil 100: Word2vec-CNN-TS:%70-ACC

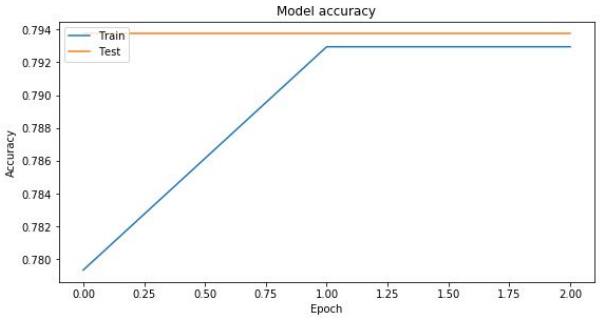


Şekil 101: Word2vec-CNN-TS:%70-LOSS

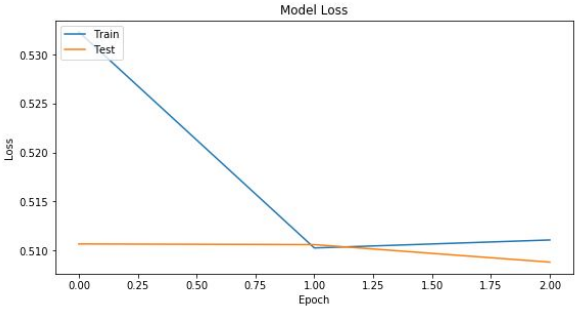
Precision	0.7958
Recall	0.7958
F1-Score	0.7958

Gösterim Modeli: Word2Vec Algoritma: RNN
Epoch : 3 Batch size : 512

TRAİN : %80 TEST: %20			
Accuracy	0.7759	**Validation Accuracy**	0.7938
Loss	0.5380	**Validation Loss**	0.5096



Şekil 102: Word2vec-RNN-TS:%20-ACC

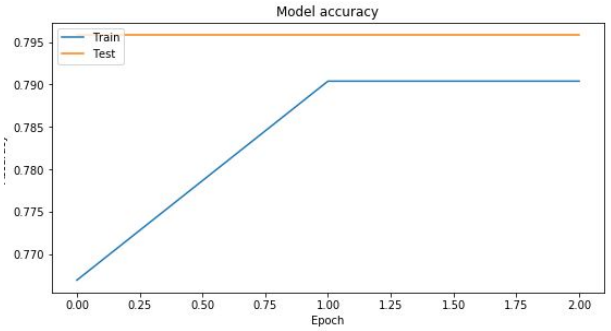


Şekil 103: Word2vec-RNN-TS:%20-LOSS

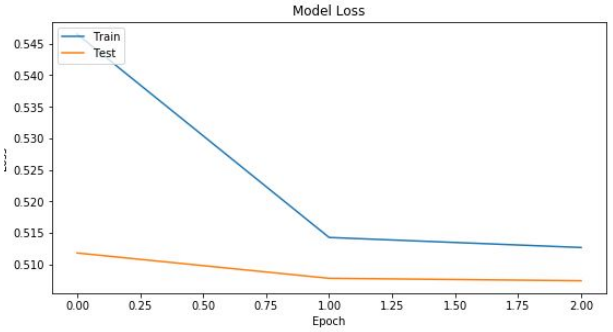
Precision	0.7937
Recall	0.7937
F1-Score	0.7937

Gösterim Modeli: Word2Vec Algoritma: RNN
Epoch : 3 Batch size : 512

TRAİN : %50 TEST: %50			
Accuracy	0.7904	**Validation Accuracy**	0.7958
Loss	0.5133	**Validation Loss**	0.5081



Şekil 104: Word2vec-RNN-TS:%50-ACC

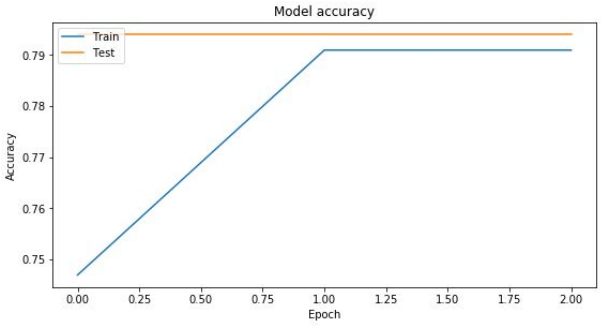


Şekil 105: Word2vec-RNN-TS:%50-LOSS

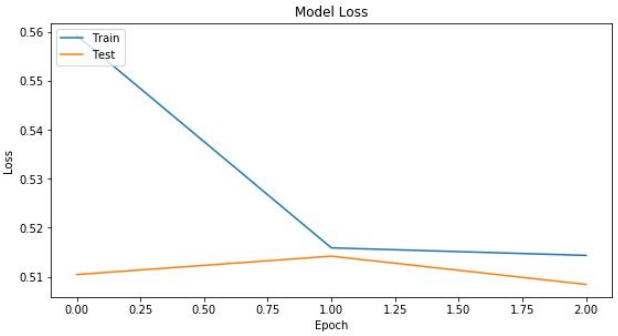
Precision	0.7937
Recall	0.7937
F1-Score	0.7937

Gösterim Modeli: Word2Vec Algoritma: RNN
Epoch : 3 Batch size : 512

TRAİN : %30 TEST: %70			
Accuracy	0.7909	**Validation Accuracy**	0.7941
Loss	0.5131	**Validation Loss**	0.5087



Şekil 106: Word2vec-RNN-TS:%70-ACC

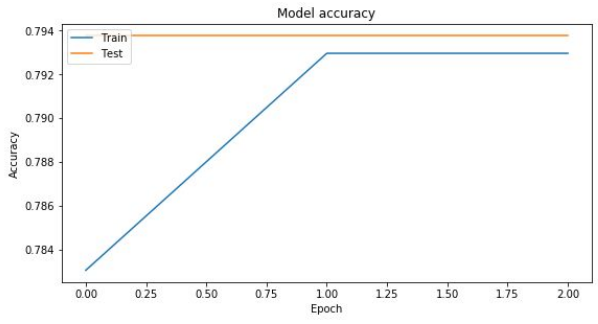


Şekil 107: Word2vec-RNN-TS:%70-LOSS

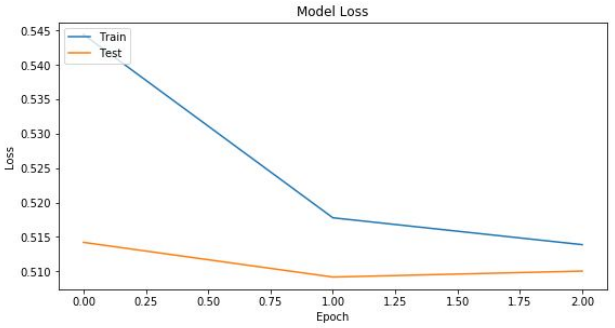
Precision	0.7937
Recall	0.7937
F1-Score	0.7937

Gösterim Modeli: Word2Vec Algoritma: LSTM
Epoch : 3 Batch size : 512

TRAİN : %80 TEST: %20			
Accuracy	0.7930	**Validation Accuracy**	0.7938
Loss	0.5147	**Validation Loss**	0.5103



Şekil 108: Word2vec-Lstm-TS:%20-ACC

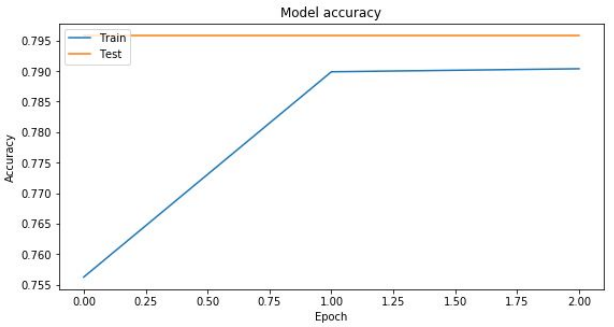


Şekil 109: Word2vec-Lstm-TS:%20-LOSS

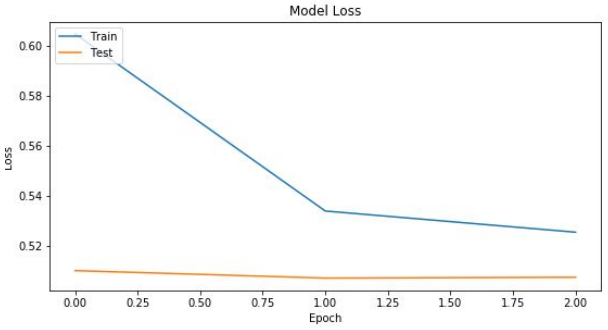
Precision	0.7937
Recall	0.7937
F1-Score	0.7937

Gösterim Modeli: Word2Vec Algoritma: LSTM
Epoch : 3 Batch size : 512

TRAİN : %50 TEST: %50			
Accuracy	0.7904	**Validation Accuracy**	0.7958
Loss	0.5226	**Validation Loss**	0.5069



Şekil 110: Word2vec-Lstm-TS:%50-ACC

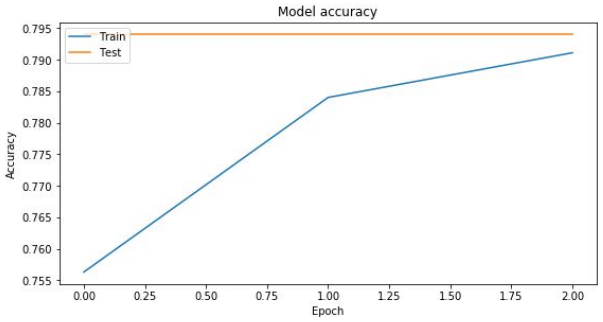


Şekil 111: Word2vec-Lstm-TS:%50-LOSS

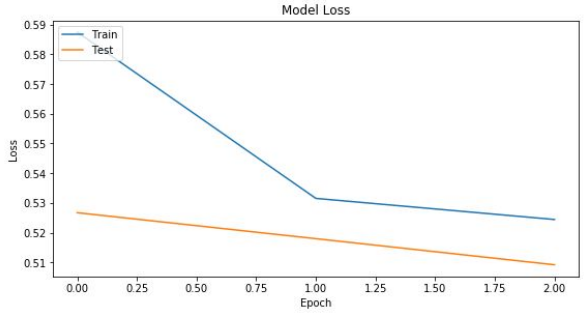
Precision	0.7958
Recall	0.7958
F1-Score	0.7958

Gösterim Modeli: Word2Vec Algoritma: LSTM
Epoch : 3 Batch size : 512

TRAİN : %30 TEST: %70			
Accuracy	0.7908	**Validation Accuracy**	0.7941
Loss	0.5270	**Validation Loss**	0.5097



Şekil 112: Word2vec-Lstm-TS:%70-ACC

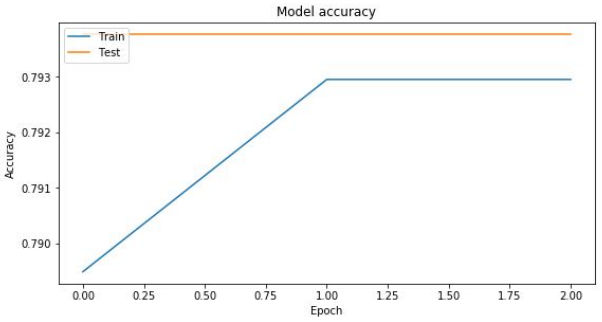


Şekil 113: Word2vec-Lstm-TS:%70-LOSS

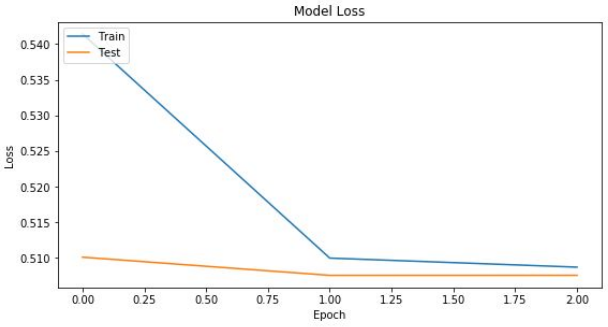
Precision	0.7940
Recall	0.7940
F1-Score	0.7940

Gösterim Modeli: Binary Algoritma: CNN
Epoch : 3 Batch size : 512

TRAİN : %80 TEST: %20			
Accuracy	0.7929	**Validation Accuracy**	0.7938
Loss	0.5582	**Validation Loss**	0.5113



Şekil 114: Binary-CNN-TS:%20-ACC

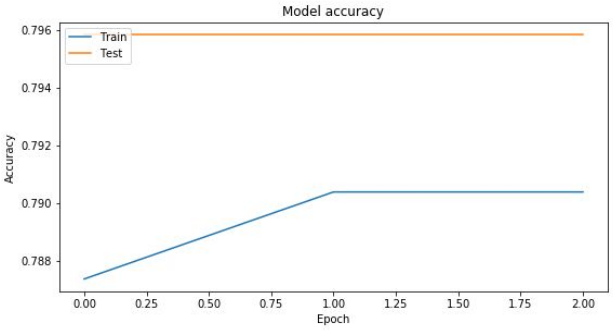


Şekil 115: Binary-CNN-TS:%20-LOSS

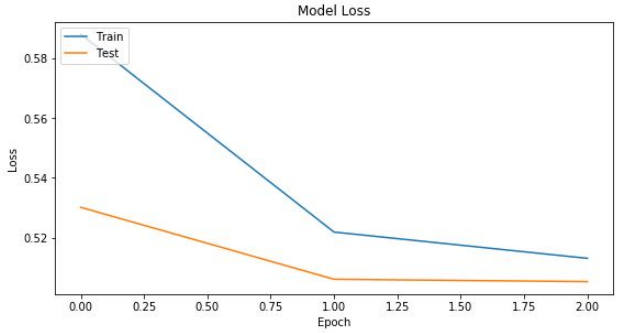
Precision	0.7937
Recall	0.7937
F1-Score	0.7937

Gösterim Modeli: Binary Algoritma: CNN
Epoch : 3 Batch size : 512

TRAİN : %50 TEST: %50			
Accuracy	0.7754	**Validation Accuracy**	0.7958
Loss	0.5882	**Validation Loss**	0.5061



Şekil 116: Binary-CNN-TS:%50-ACC

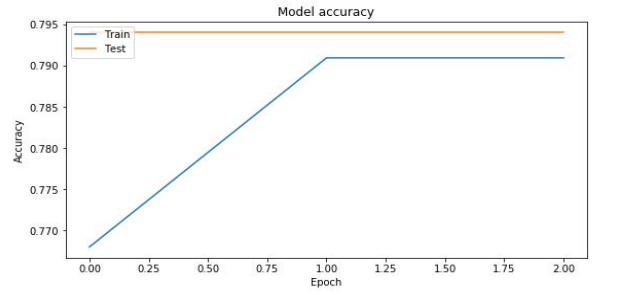


Şekil 117: Binary-CNN-TS:%50-LOSS

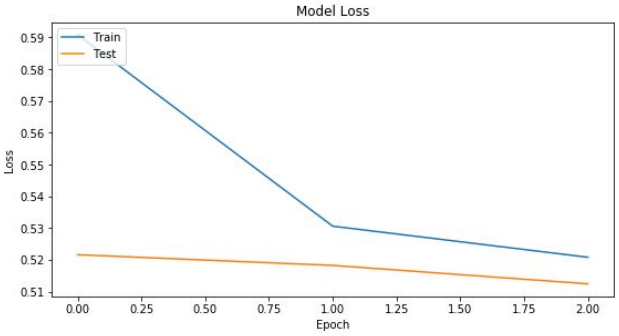
Precision	0.7958
Recall	0.7958
F1-Score	0.7958

Gösterim Modeli: Binary Algoritma: CNN
Epoch : 3 Batch size : 512

TRAİN : %30 TEST: %70			
Accuracy	0.5800	**Validation Accuracy**	0.7941
Loss	0.6561	**Validation Loss**	0.6123



Şekil 118: Binary-CNN-TS:%70-ACC

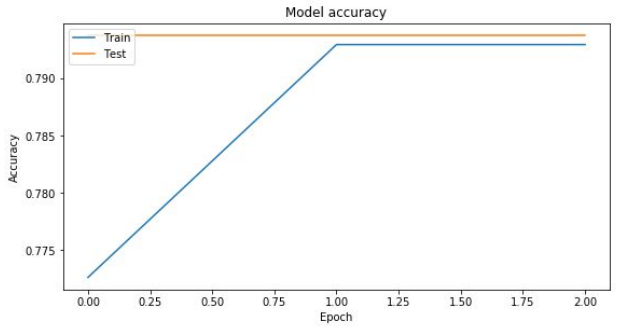


Şekil 119: Binary-CNN-TS:%70-LOSS

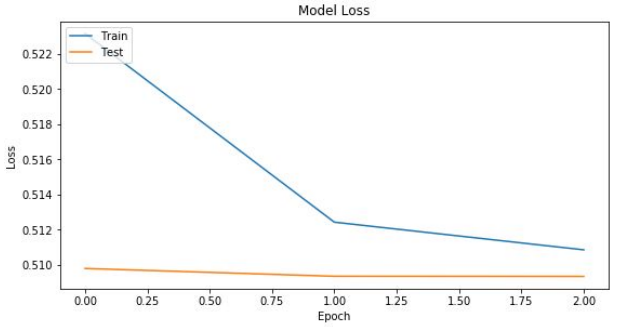
Precision	0.7940
Recall	0.7940
F1-Score	0.7940

Gösterim Modeli: Binary Algoritma: RNN
Epoch : 3 Batch size : 512

TRAİN : %80 TEST: %20			
Accuracy	0.7704	**Validation Accuracy**	0.7938
Loss	0.5298	**Validation Loss**	0.5247



Şekil 120: Binary-RNN-TS:%20-ACC

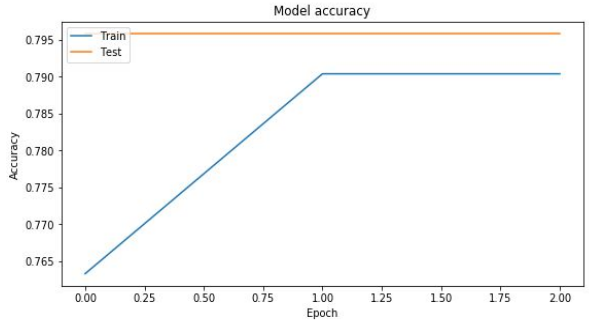


Şekil 121: Binary-RNN-TS:%20-LOSS

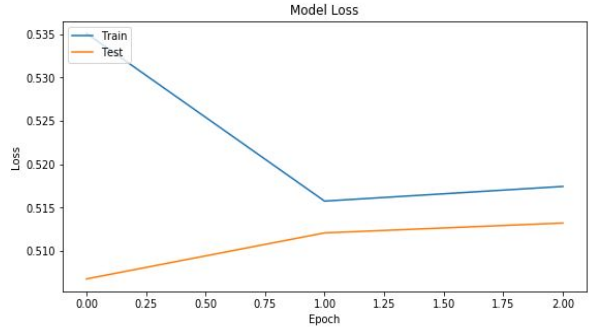
Precision	0.7937
Recall	0.7937
F1-Score	0.7937

Gösterim Modeli: Binary Algoritma: RNN
Epoch : 3 Batch size : 512

TRAİN : %50 TEST: %50			
Accuracy	0.7904	**Validation Accuracy**	0.7958
Loss	0.5186	**Validation Loss**	0.5070



Şekil 122: Binary-RNN-TS:%50-ACC

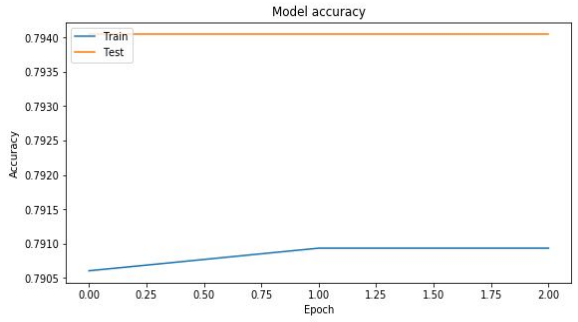


Şekil 123: Binary-RNN-TS:%50-LOSS

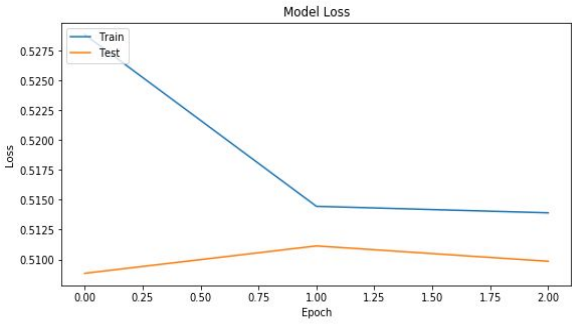
Precision	0.7937
Recall	0.7937
F1-Score	0.7937

Gösterim Modeli: Binary Algoritma: RNN
Epoch : 3 Batch size : 512

TRAİN : %30 TEST: %70			
Accuracy	0.6465	**Validation Accuracy**	0.7941
Loss	0.5988	**Validation Loss**	0.6153



Şekil 124: Binary-RNN-TS:%70-ACC

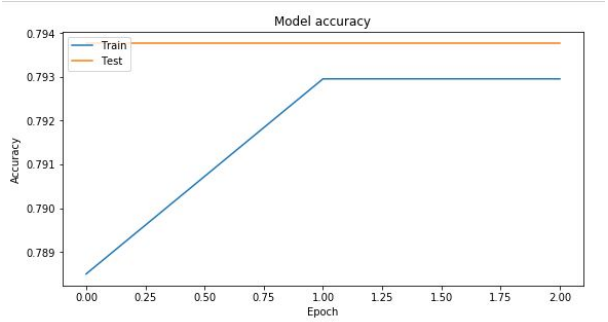


Şekil 125: Binary-RNN-TS:%70-LOSS

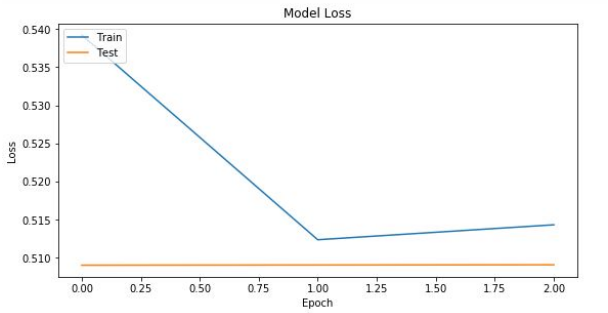
Precision	0.7940
Recall	0.7940
F1-Score	0.7940

Gösterim Modeli: Binary Algoritma: LSTM
Epoch : 3 Batch size : 512

TRAİN : %80 TEST: %20			
Accuracy	0.7798	**Validation Accuracy**	0.7938
Loss	0.5457	**Validation Loss**	0.5091



Şekil 126: Binary-Lstm-TS:%20-ACC

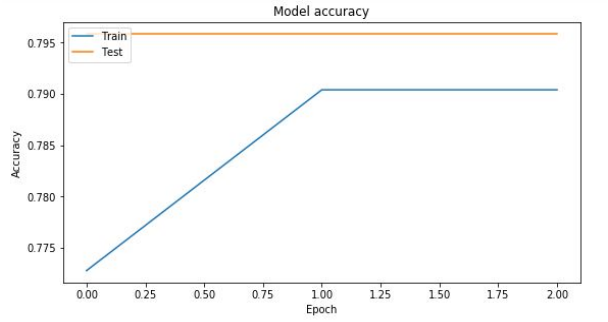


Şekil 127: Binary-Lstm-TS:%20-LOSS

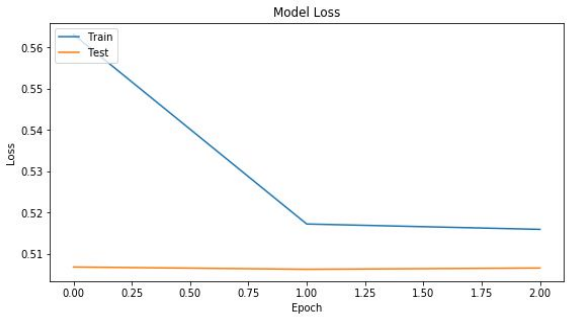
Precision	0.7937
Recall	0.7937
F1-Score	0.7937

Gösterim Modeli: Binary Algoritma: LSTM
Epoch : 3 Batch size : 512

TRAİN : %50 TEST: %50			
Accuracy	0.7624	**Validation Accuracy**	0.7958
Loss	0.5769	**Validation Loss**	0.5144



Şekil 128: Binary-Lstm-TS:%50-ACC

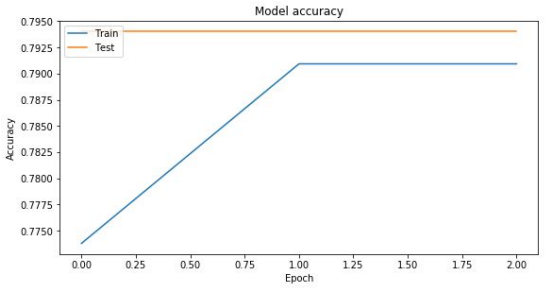


Şekil 129: Binary-Lstm-TS:%50-LOSS

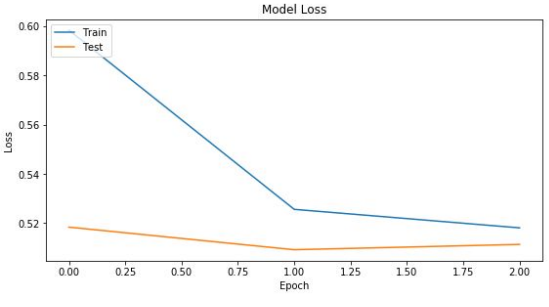
Precision	0.7937
Recall	0.7937
F1-Score	0.7937

Gösterim Modeli: Binary Algoritma: LSTM
Epoch : 3 Batch size : 512

TRAİN : %30 TEST: %70			
Accuracy	0.7571	**Validation Accuracy**	0.7941
Loss	0.5984	**Validation Loss**	0.5087



Şekil 130: Binary-Lstm-TS:%70-ACC

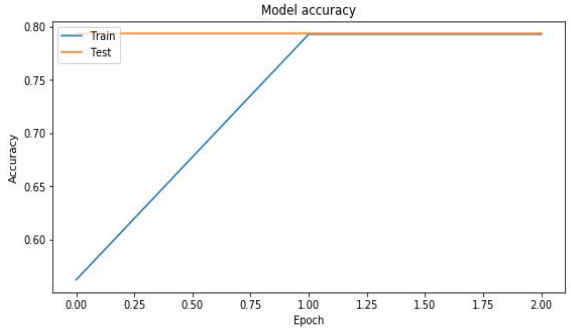


Şekil 131: Binary-Lstm-TS:%70-LOSS

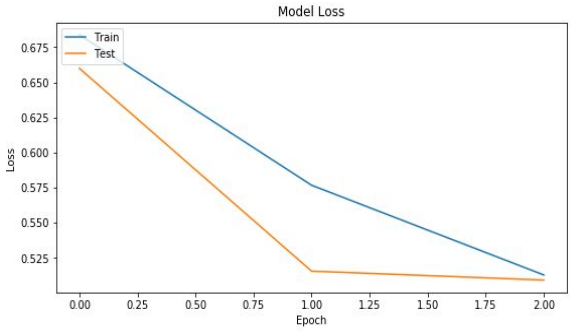
Precision	0.7940
Recall	0.7940
F1-Score	0.7940

Gösterim Modeli: Tf Algoritma: CNN
Epoch : 3 Batch size : 512

TRAİN : %80 TEST: %20			
Accuracy	0.7749	**Validation Accuracy**	0.7938
Loss	0.6172	**Validation Loss**	0.5161



Şekil 132: TF-CNN-TS:%20-ACC

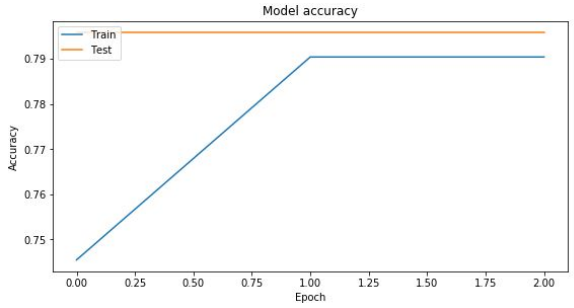


Şekil 133: TF-CNN-TS:%20-LOSS

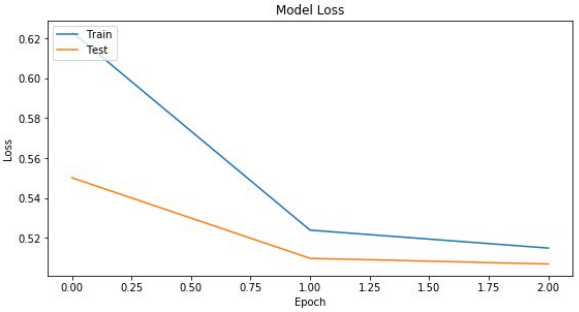
Precision	0.7937
Recall	0.7937
F1-Score	0.7937

Gösterim Modeli: Tf Algoritma: CNN
Epoch : 3 Batch size : 512

TRAİN : %50 TEST: %50			
Accuracy	0.7904	**Validation Accuracy**	0.7958
Loss	0.6095	**Validation Loss**	0.5203



Şekil 134: TF-CNN-TS:%50-ACC

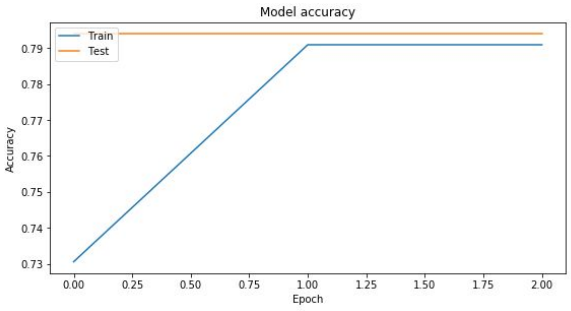


Şekil 135: TF-CNN-TS:%50-LOSS

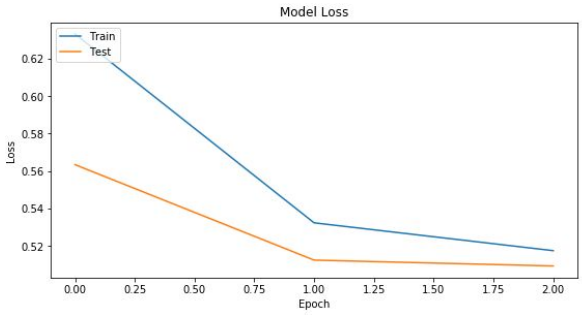
Precision	0.7958
Recall	0.7958
F1-Score	0.7958

Gösterim Modeli: Tf Algoritma: CNN
Epoch : 3 Batch size : 512

TRAİN : %30 TEST: %70			
Accuracy	0.7406	**Validation Accuracy**	0.7941
Loss	0.6495	**Validation Loss**	0.5665



Şekil 136: TF-CNN-TS:%70-ACC

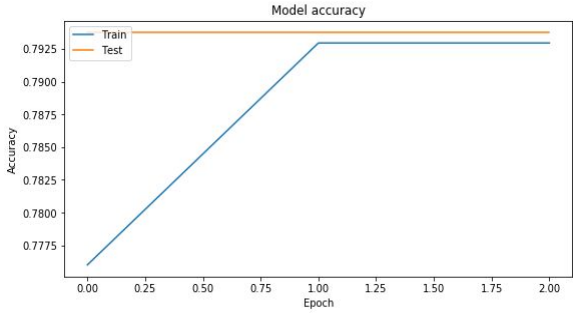


Şekil 137: TF-CNN-TS:%70-LOSS

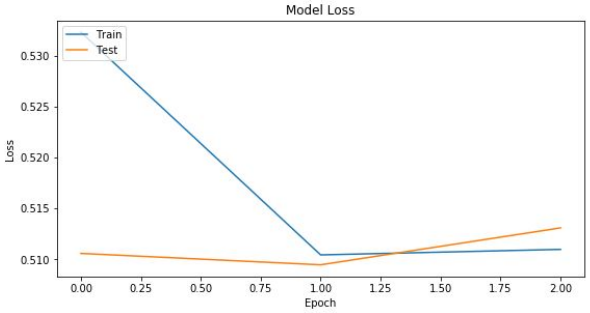
Precision	0.7940
Recall	0.7940
F1-Score	0.7940

Gösterim Modeli: Tf Algoritma: RNN
Epoch : 3 Batch size : 512

TRAİN : %80 TEST: %20			
Accuracy	0.7734	**Validation Accuracy**	0.7938
Loss	0.5227	**Validation Loss**	0.5102



Şekil 138: TF-RNN-TS:%20-ACC

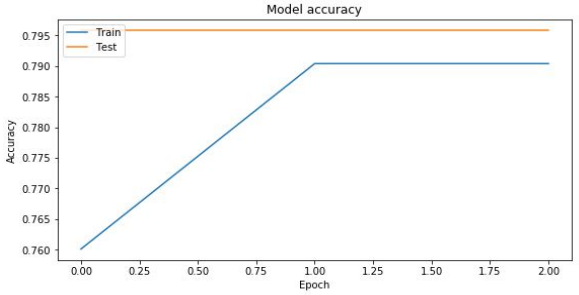


Şekil 139: TF-RNN-TS:%20-LOSS

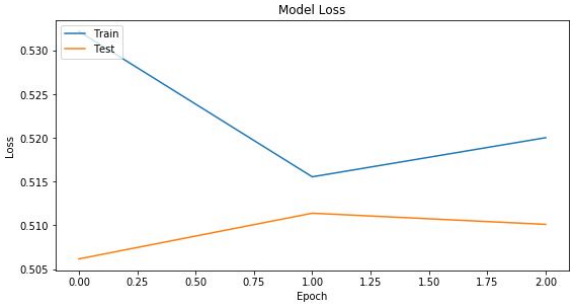
Precision	0.7937
Recall	0.7937
F1-Score	0.7937

Gösterim Modeli: Tf Algoritma: RNN
Epoch : 3 Batch size : 512

TRAİN : %50 TEST: %50			
Accuracy	0.7904	**Validation Accuracy**	0.7958
Loss	0.5177	**Validation Loss**	0.5068



Şekil 140: TF-RNN-TS:%50-ACC

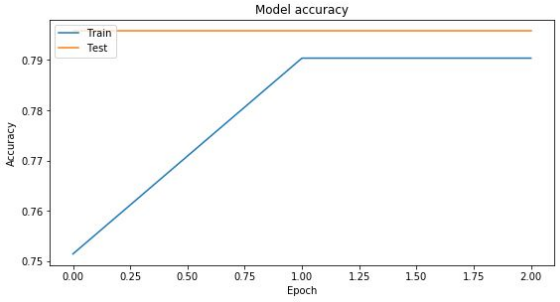


Şekil 141: TF-RNN-TS:%50-LOSS

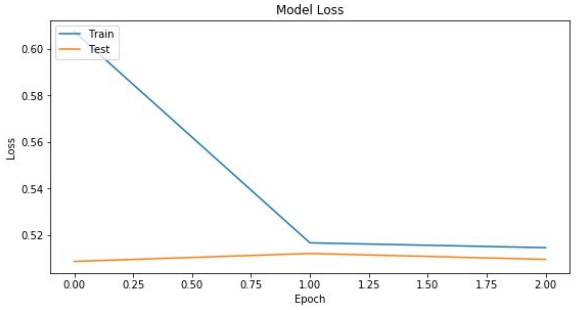
Precision	0.7958
Recall	0.7958
F1-Score	0.7958

Gösterim Modeli: Tf Algoritma: RNN
Epoch : 3 Batch size : 512

TRAİN : %30 TEST: %70			
Accuracy	0.7471	**Validation Accuracy**	0.7941
Loss	0.5606	**Validation Loss**	0.5163



Şekil 142: TF-RNN-TS:%70-ACC

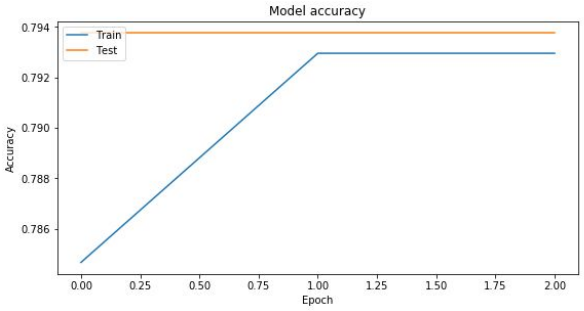


Şekil 143: TF-RNN-TS:%70-LOSS

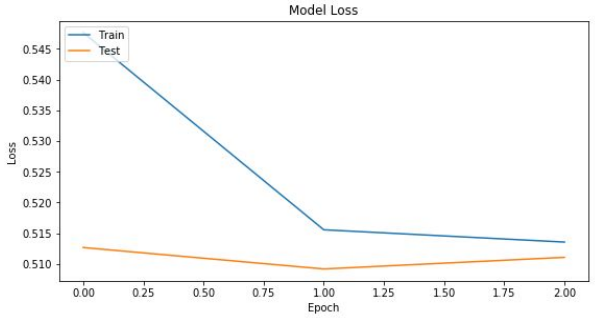
Precision	0.7940
Recall	0.7940
F1-Score	0.7940

Gösterim Modeli: Tf Algoritma: LSTM
Epoch : 3 Batch size : 512

TRAİN : %80 TEST: %20			
Accuracy	0.7851	**Validation Accuracy**	0.7938
Loss	0.5432	**Validation Loss**	0.5101



Şekil 144: TF-Lstm-TS:%20-ACC

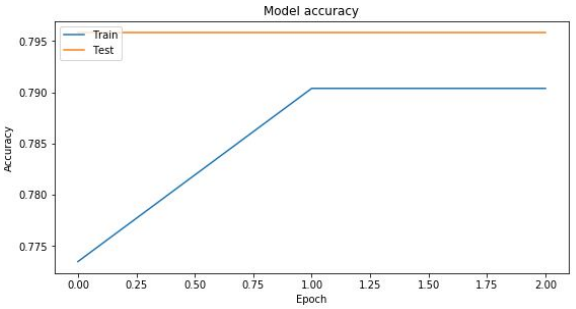


Şekil 145: TF-Lstm-TS:%20-LOSS

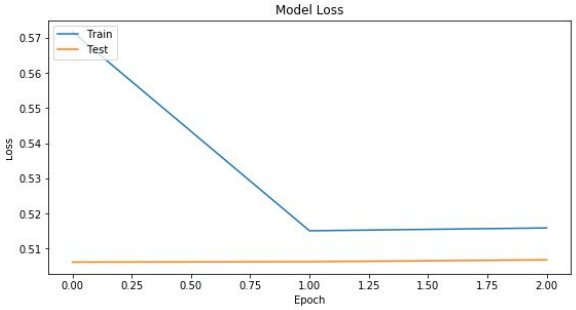
Precision	0.7937
Recall	0.7937
F1-Score	0.7937

Gösterim Modeli: Tf Algoritma: LSTM
Epoch : 3 Batch size : 512

TRAİN : %50 TEST: %50			
Accuracy	0.7899	**Validation Accuracy**	0.7958
Loss	0.5710	**Validation Loss**	0.5110



Şekil 146: TF-LSTM-TS:%50-ACC

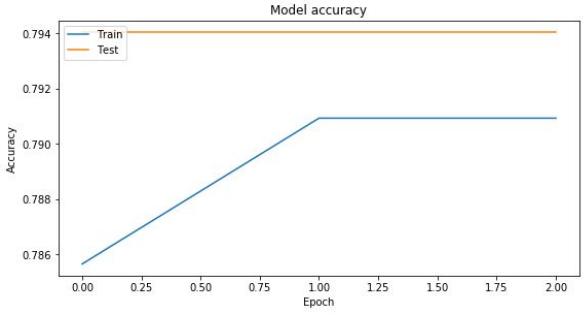


Şekil 147: TF-LSTM-TS:%50-LOSS

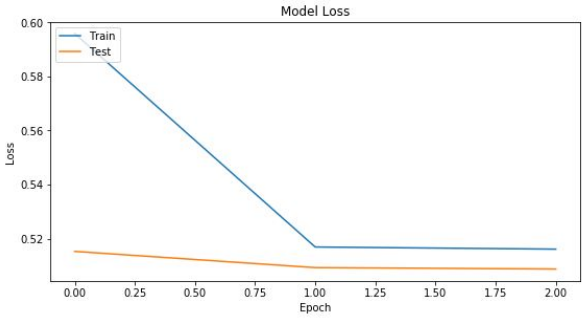
Precision	0.7958
Recall	0.7958
F1-Score	0.7958

Gösterim Modeli: Tf Algoritma: LSTM
Epoch : 3 Batch size : 512

TRAİN : %30 TEST: %70			
Accuracy	0.7779	**Validation Accuracy**	0.7941
Loss	0.6035	**Validation Loss**	0.5190



Şekil 148: TF-LSTM-TS:%70-ACC

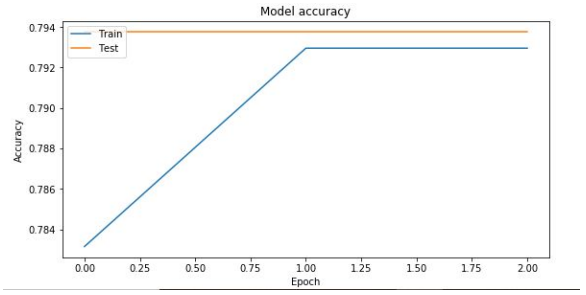


Şekil 149: TF-LSTM-TS:%70-LOSS

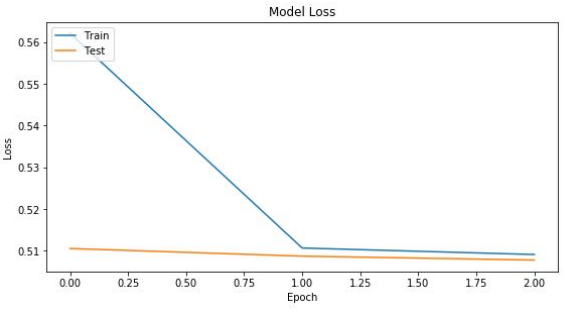
Precision	0.7940
Recall	0.7940
F1-Score	0.7940

Gösterim Modeli: Tf-idf Algoritma: CNN
Epoch : 3 Batch size : 512

TRAİN : %80 TEST: %20			
Accuracy	0.7930	**Validation Accuracy**	0.7938
Loss	0.5170	**Validation Loss**	0.5133



Şekil 150: Tf-idf-CNN-TS:%20-ACC

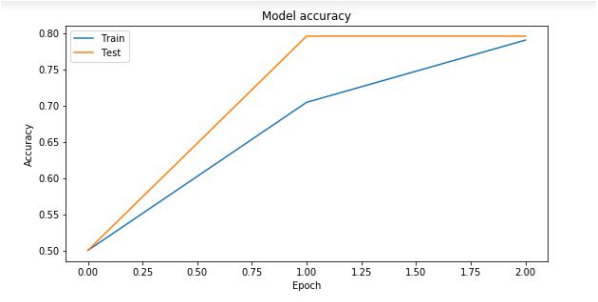


Şekil 151: Tf-idf-CNN-TS:%20-LOSS

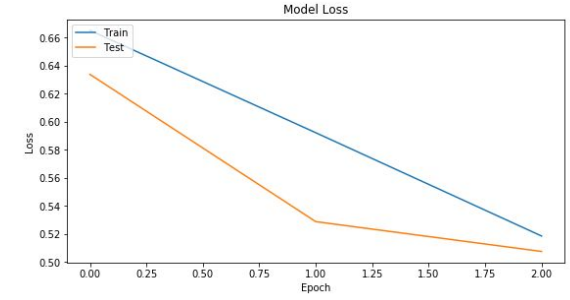
Precision	0.7937
Recall	0.7937
F1-Score	0.7937

Gösterim Modeli: Tf-idf Algoritma: CNN
Epoch : 3 Batch size : 512

TRAİN : %50 TEST: %50			
Accuracy	0.7904	**Validation Accuracy**	0.7958
Loss	0.6792	**Validation Loss**	0.6760



Şekil 152: Tf-idf-CNN-TS:%50-ACC

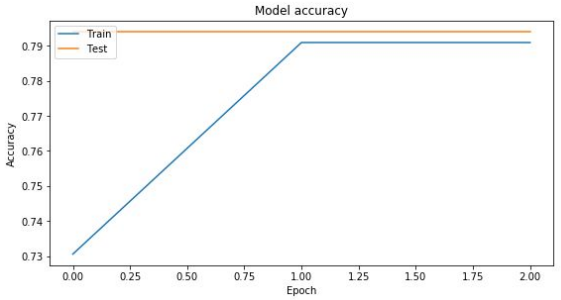


Şekil 153: Tf-idf-CNN-TS:%50-LOSS

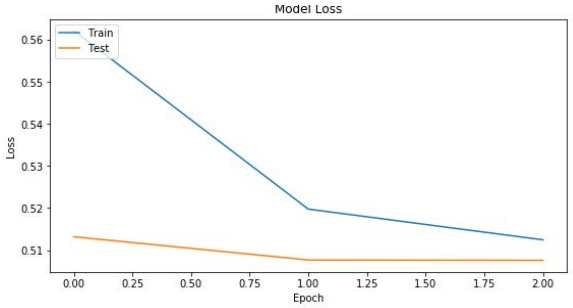
Precision	0.7958
Recall	0.7958
F1-Score	0.7958

Gösterim Modeli: Tf-idf Algoritma: CNN
Epoch : 3 Batch size : 512

TRAİN : %30 TEST: %70			
Accuracy	0.7909	**Validation Accuracy**	0.7941
Loss	0.5194	**Validation Loss**	0.5079



Şekil 154: Tf-idf-CNN-TS:%70-ACC

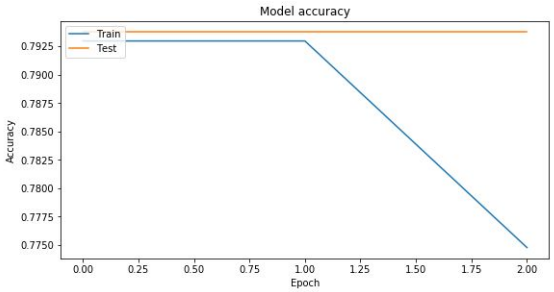


Şekil 155: Tf-idf-CNN-TS:%70-LOSS

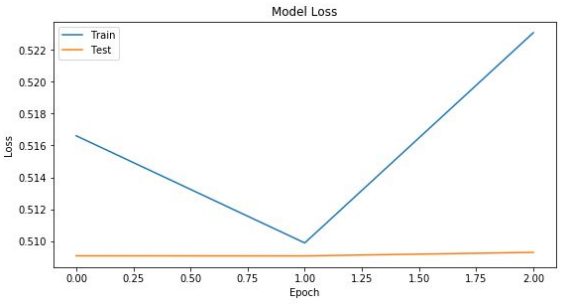
Precision	0.7940
Recall	0.7940
F1-Score	0.7940

Gösterim Modeli: Tf-idf Algoritma: RNN
Epoch : 3 Batch size : 512

TRAİN : %80 TEST: %20			
Accuracy	0.7764	**Validation Accuracy**	0.7938
Loss	0.5223	**Validation Loss**	0.5100



Şekil 156: Tf-idf-RNN-TS:%20-ACC

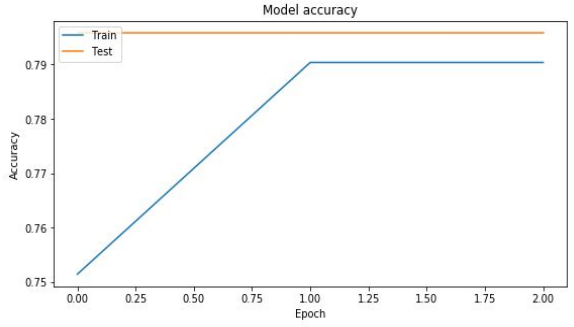


Şekil 157: Tf-idf-RNN-TS:%20-LOSS

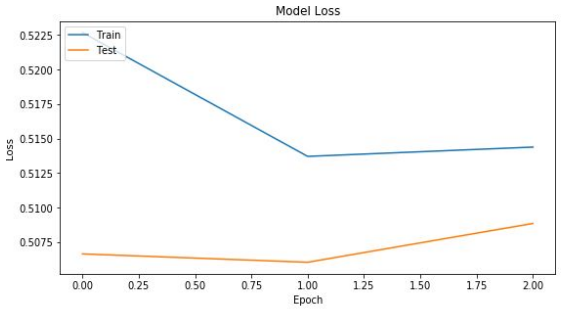
Precision	0.7937
Recall	0.7937
F1-Score	0.7937

Gösterim Modeli: Tf-idf Algoritma: RNN
Epoch : 3 Batch size : 512

TRAİN : %50 TEST: %50			
Accuracy	0.7064	**Validation Accuracy**	0.7958
Loss	0.5997	**Validation Loss**	0.5072



Şekil 158: Tf-idf-RNN-TS:%50-ACC

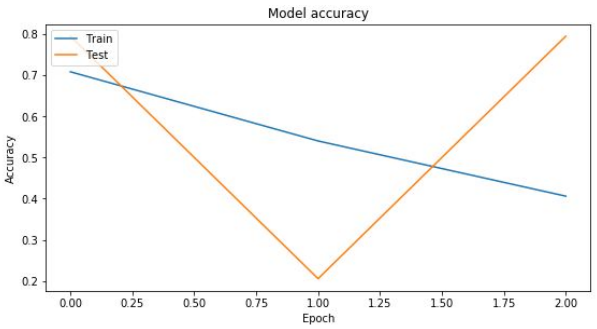


Şekil 159: Tf-idf-RNN-TS:%50-LOSS

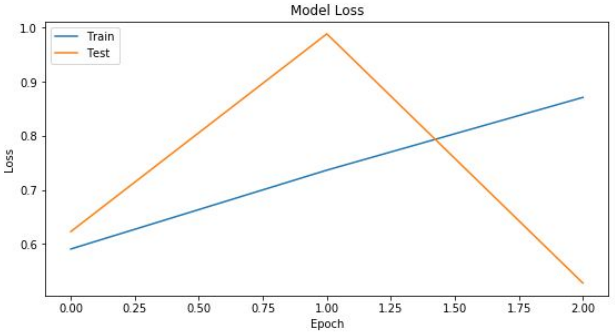
Precision	0.7958
Recall	0.7958
F1-Score	0.7958

Gösterim Modeli: Tf-idf Algoritma: RNN
Epoch : 3 Batch size : 512

TRAİN : %30 TEST: %70			
Accuracy	0.7411	**Validation Accuracy**	0.7941
Loss	0.5551	**Validation Loss**	0.5162



Şekil 160: Tf-idf-RNN-TS:%70-ACC

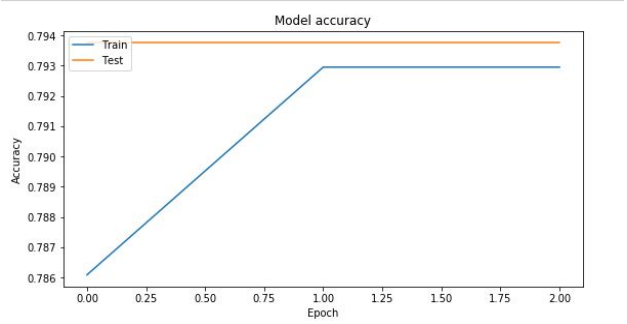


Şekil 161: Tf-idf-RNN-TS:%70-LOSS

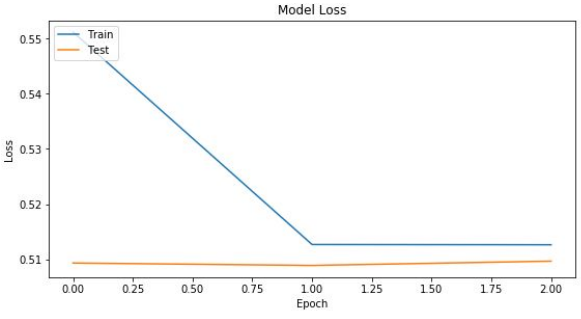
Precision	0.7940
Recall	0.7940
F1-Score	0.7940

Gösterim Modeli: Tf-idf Algoritma: LSTM
Epoch : 3 Batch size : 512

TRAİN : %80 TEST: %20			
Accuracy	0.7830	**Validation Accuracy**	0.7938
Loss	0.5452	**Validation Loss**	0.5103



Şekil 162: Tf-idf-LSTM-TS:%20-ACC

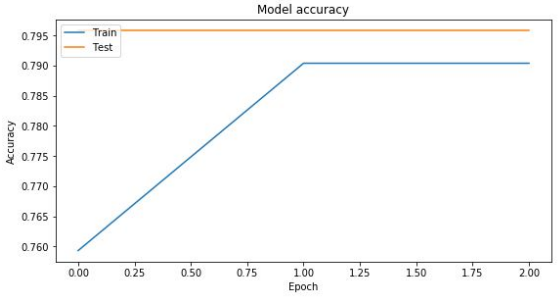


Şekil 163: Tf-idf-LSTM-TS:%20-LOSS

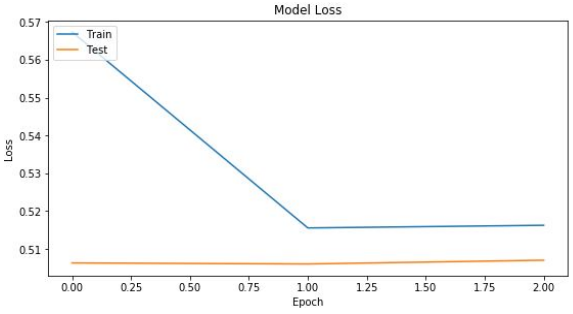
Precision	0.7937
Recall	0.7937
F1-Score	0.7937

Gösterim Modeli: Tf-idf Algoritma: LSTM
Epoch : 3 Batch size : 512

TRAİN : %50 TEST: %50			
Accuracy	0.7869	**Validation Accuracy**	0.7958
Loss	0.5853	**Validation Loss**	0.5369



Şekil 164: Tf-idf-LSTM-TS:%50-ACC

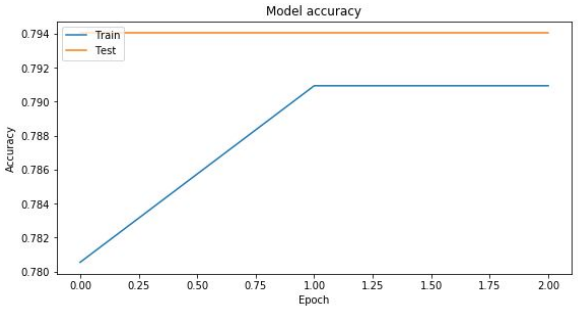


Şekil 165: Tf-idf-LSTM-TS:%50-LOSS

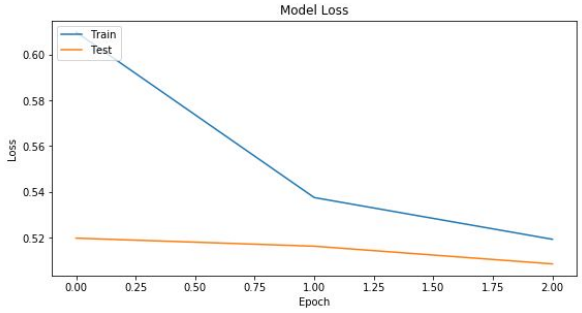
Precision	0.7958
Recall	0.7958
F1-Score	0.7958

Gösterim Modeli: Tf-idf Algoritma: LSTM
Epoch : 3 Batch size : 512

TRAİN : %30 TEST: %70			
Accuracy	0.7743	**Validation Accuracy**	0.7941
Loss	0.5950	**Validation Loss**	0.5173



Şekil 166: Tf-idf-LSTM-TS:%70-ACC



Şekil 167: Tf-idf-LSTM-TS:%70-LOSS

Precision	0.7940
Recall	0.7940
F1-Score	0.7940

VI. Kaynakça

[1]https://machinelearningmastery.com/develop-word-embeddings-python-gensim/

[2]https://towardsdatascience.com/word-embedding-with-word2vec-and-fasttext-a209c1d3e12c

[3]https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/

[4]https://realpython.com/python-keras-text-classification/

[5]https://www.toptal.com/python/twitter-data-mining-using-python

[6]https://medium.com/codable/word2vec-fasttext-glove-d4402fa8cce0

[7]https://medium.com/jatana/report-on-text-classification-using-cnn-rnn-han-f0e887214d5f

[8]https://textblob.readthedocs.io/en/dev/

[9]https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/

[10]http://acikerisim.fsm.edu.tr:8080/xmlui/bitstream/handle/11352/2702/%C3%87al%C4%B1%C5%9Fkan.pdf?sequence=1&isAllowed=y

[11]http://elitcenkalp.blogspot.com/2018/04/recurrent-neural-network.html

[12]https://medium.com/@tuncerergin/convolutional-neural-network-convnet-yada-cnn-nedir-nasil-calisir-97a0f5d34cad

[13]https://stackoverflow.com/questions/60195735/use-tf-idf-with-in-keras-model

[14]https://devhunteryz.wordpress.com/2018/04/08/evrisimsel-sinir-aglariconvolutional-neural-network/

[15]https://medium.com/@ishakdolek/lstm-d2c281b92aac

[16]https://dergipark.org.tr/tr/download/article-file/394923

[17]http://bilgisayarkavramlari.sadievrenseker.com/2012/10/22/tf-idf/