Doğal Dil İşleme Dersi Proje Ödevi

Spam Mail Detection

19011062 – Sude Şevval Aydın

18011068 – Çağdaş Çetin

Spam Mail Nedir?

Yeni teknoloji çağında, e-postalar profesyonel, ticari ve kişisel iletişim için önemli miktarda kullanıcı topladı. 2019'da ortalama olarak her birey her gün 130 e-posta alıyordu ve o yıl toplamda 296 Milyar e-posta gönderildi.

Yüksek talep ve büyük kullanıcı tabanı nedeniyle, aynı zamanda spam olarak da bilinen istenmeyen e-postalarda bir artış var. Spam e-postaların toplam e-postalara oranının %50'ye kadar çıktığı tespit edilmiştir.

Spam e-postaların kullanıldığı bazı farklı amaçlar şunlardır:

* Ticari reklamlar ve pazarlama faaliyetleri
* Kampanya ve duyurular
* Yanıltıcı ve gerçek olmayan vaatler (Hızlı zengin olma yöntemleri vb.)
* Yasal olmayan veya yarı yasal faaliyet/servis duyuruları
* UBE (Unsolicited Bulk E-Mail: Talep Edilmemiş Kitlesel E-Posta)
* Politik veya ideolojik bir görüşün propagandası
* Belli bir konuda kamuoyu oluşturma
* Kimlik avı saldırısı

Biz de bu istenmeyen maillerin tespit edilmesi üzerine çalıştık ve projemizi Python dilini kullanarak bu konu üzerine geliştirdik.

Veri Setleri

İki adet veri seti kullanmayı tercih ettik, ve kullanırken bu setleri birleştirdik.

* <https://www.kaggle.com/itsbhups/98-accuracy-spam-ham-messages/data>
* <https://www.kaggle.com/venky73/spam-mails-dataset>

Bu veri setlerini;

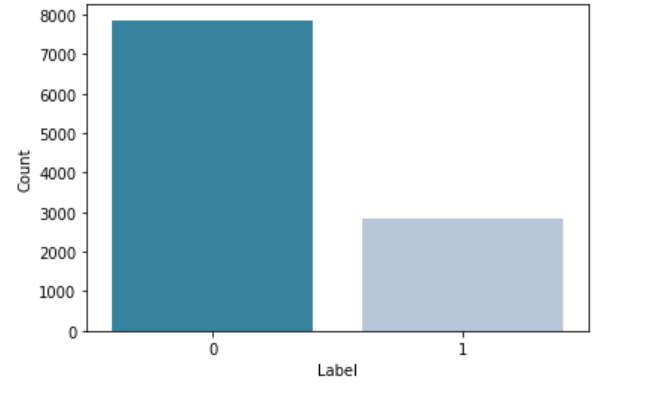
data=pd.read\_csv("Email spam.csv")  
data.rename(columns={"spam":"target"},inplace=True)  
data1=pd.read\_csv("spam\_ham\_dataset.csv")  
data1.drop(["label","Unnamed: 0"],axis=1,inplace=True)  
data1.rename(columns={"label\_num":"target"},inplace=True)  
datadf=pd.concat([data,data1],axis=0)  
datadf.reset\_index(inplace=True,drop=True)  
datadf.index=range(datadf.shape[0])  
datadf=datadf.drop\_duplicates()  
datadf.head(10)

yaparak birleştirdik.

Preprocessing aşamaları

Önce verilerimizin spam olma oranlarına baktık. Spam oranı yaklaşık olarak %20 çıktı

sns.countplot(x ='target', data = datadf,palette="PuBuGn\_r")  
plt.xlabel("Label")  
plt.ylabel("Count")  
plt.show()



Daha sonra verimizi temizlemeye başladık. Text sütunumuzun ilk kelimesi olan Subject’i sildik. Tüm harfleri küçük harfe çevirdik. Stopword’leri temizledik. Noktalama işaretleri ve tek harfleri sildik, bu sayede üzerinde işlem uygulayabileceğimiz temiz bir veri setini elde etmiş olduk.  
for i in range(len(datadf)):  
 datadf.iloc[i, 0] = datadf.iloc[i, 0].split(' ', 1)[1]  
  
datadf['text']= datadf['text'].apply(lambda x: x.lower())  
  
stop\_words = set(stopwords.words('english') + ['u', 'ü', 'ur', '4', '2', 'im', 'dont', 'doin', 'ure'])  
datadf['text']=datadf['text'].apply(lambda x: " ".join(term for term in x.split() if term not in stop\_words))  
  
def remove\_punctuation(text):  
 punctuationfree="".join([i for i in text if i not in string.punctuation])  
 return punctuationfree  
datadf['text']= datadf['text'].apply(lambda x:remove\_punctuation(x))  
  
def remove\_singlechar(text):  
 singlecharfree="".join([i for i in text if len(i)<=1])  
 return singlecharfree  
datadf['text']= datadf['text'].apply(lambda x:remove\_singlechar(x))

Stemming, bir kelimenin son birkaç karakterini kaldırır veya köklendirir, bu da genellikle yanlış anlamlara ve yazımlara yol açar. Lemmatizasyon, bağlamı dikkate alır ve kelimeyi Lemma adı verilen anlamlı temel formuna dönüştürür. Lemmatizasyon için WordNet ten POS etiketleri kullanılır. En doğru sonuca bu şekilde ulaşır. Biz de bu sebeplerden ötürü lemmatization kullandık.

print(datadf.head())  
datadf["text"]=datadf['text'].apply(word\_tokenize)  
def get\_pos( word ):  
 w\_synsets = wordnet.synsets(word)  
 pos\_counts = Counter()  
 pos\_counts["n"] = len( [ item for item in w\_synsets if item.pos()=="n"] )  
 pos\_counts["v"] = len( [ item for item in w\_synsets if item.pos()=="v"] )  
 pos\_counts["a"] = len( [ item for item in w\_synsets if item.pos()=="a"] )  
 pos\_counts["r"] = len( [ item for item in w\_synsets if item.pos()=="r"] )  
 most\_common\_pos\_list = pos\_counts.most\_common(3)  
 return most\_common\_pos\_list[0][0]  
wordnet\_lemmatizer = WordNetLemmatizer()  
def lemmatizer(text):  
 lemm\_text = [wordnet\_lemmatizer.lemmatize(word,get\_pos(word)) for word in text]  
 return lemm\_text  
datadf['text']=datadf['text'].apply(lambda x:lemmatizer(x))  
def concanatee(text):  
 listToStr = ' '.join([str(elem) for elem in text])  
 return listToStr  
datadf['text']=datadf['text'].apply(lambda x:concanatee(x))

Bag of Words, belgede geçen kelime sayısını içeren bir dizi vektör oluştururken, TF-IDF (Term Frequency and Inverse Document Frequency) modeli daha önemli ve daha az önemli kelimeler hakkında bilgi içerir. Bag of Words vektörlerinin yorumlanması kolaydır. Ancak, TF-IDF genellikle makine öğrenimi modellerinde daha iyi performans gösterir. Biz de bu yüzden TF-IDF kullanmayı tercih ettik.

vectorizer = TfidfVectorizer()  
vectors = vectorizer.fit\_transform(datadf['text'])  
features = vectors  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, datadf['target'], test\_size=0.2)  
sonuc=[]  
algo=["KNN","Random Forest","SVM"]

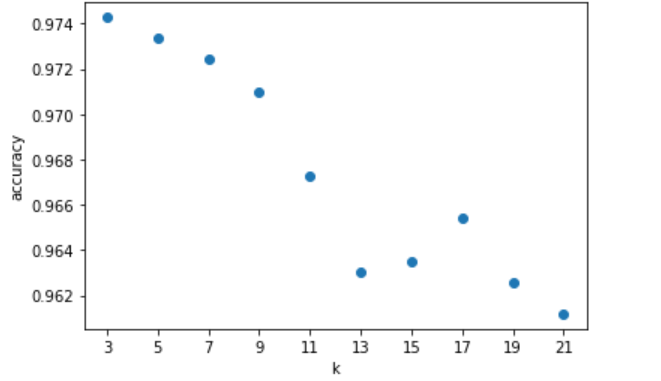
Kullandığımız Algoritmalar ve Doğrulukları

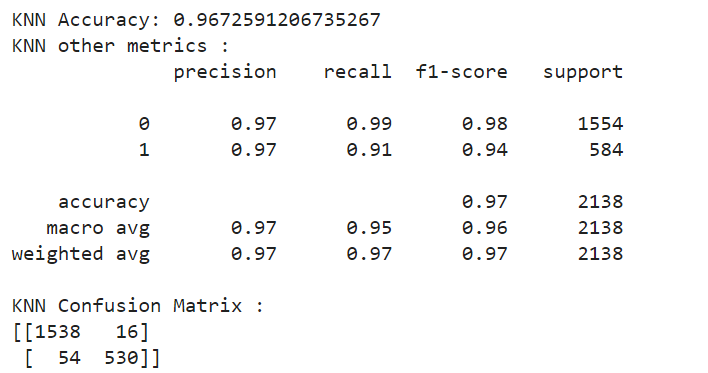
Projemiz için 3 farklı algoritmayı test etmeye karar verdik. Bu algoritmalar, KNN (K-Nearest Neighbors), Random Forest ve SVM (Support Vector Machine) algoritmalarıydı.

1. KNN

Bu algoritma, sınıfları belli olan bir örnek kümesindeki verilerden yararlanılarak kullanılmaktadır. Örnek veri setine katılacak olan yeni verinin, mevcut verilere göre uzaklığı hesaplanıp, k sayıda yakın komşuluğuna bakılır.

k\_range = range(3,22,2)  
scores = []  
for k in k\_range:  
 knn = KNeighborsClassifier(n\_neighbors = k)  
 knn.fit(X\_train, y\_train)  
 scores.append(knn.score(X\_test, y\_test))  
  
plt.figure()  
plt.xlabel('k')  
plt.ylabel('accuracy')  
plt.scatter(k\_range, scores)  
plt.xticks(range(3,22,2))  
plt.show()



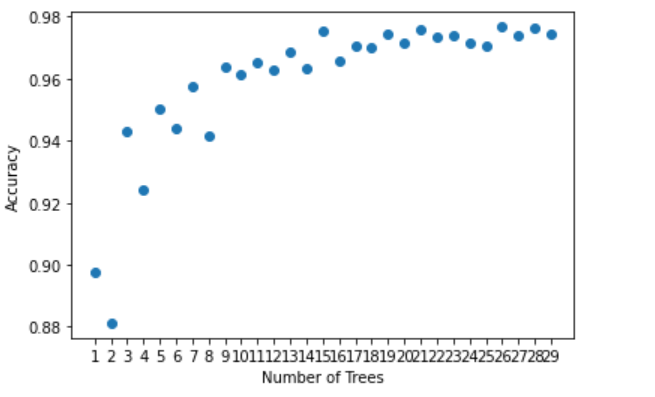
knn = KNeighborsClassifier(n\_neighbors=11)  
knn.fit(X\_train, y\_train)  
y\_pred2 = knn.predict(X\_test)  
print("KNN Accuracy:",metrics.accuracy\_score(y\_test, y\_pred2))  
sonuc.append(metrics.accuracy\_score(y\_test, y\_pred2))  
report2 = metrics.classification\_report(y\_test, y\_pred2)  
print("KNN other metrics :")  
print(report2)  
matrix2 = metrics.confusion\_matrix(y\_test, y\_pred2)  
print("KNN Confusion Matrix :")  
print(matrix2)

2. Random Forest

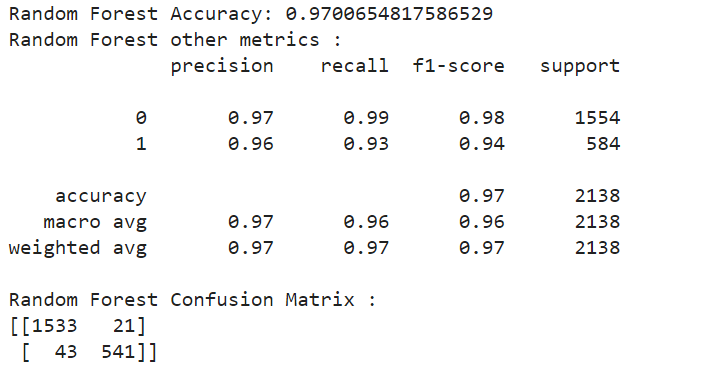
Random Forest, birden fazla karar ağacını oluşturur ve daha doğru ve istikrarlı bir tahmin elde etmek için onları birleştirir.

Random Forest’ın büyük bir avantajı, mevcut makine öğrenmesi sistemlerinin çoğunu oluşturan hem sınıflandırma hem de regresyon problemleri için kullanılabilmesidir.

n\_range = range(1, 30)  
scores = []  
for n in n\_range:  
 clf=RandomForestClassifier(n\_estimators=n)  
 clf.fit(X\_train,y\_train)  
 scores.append(clf.score(X\_test, y\_test))  
plt.figure()  
plt.xlabel('Number of Trees')  
plt.ylabel('Accuracy')  
plt.scatter(n\_range, scores)  
plt.xticks(range(1, 30))

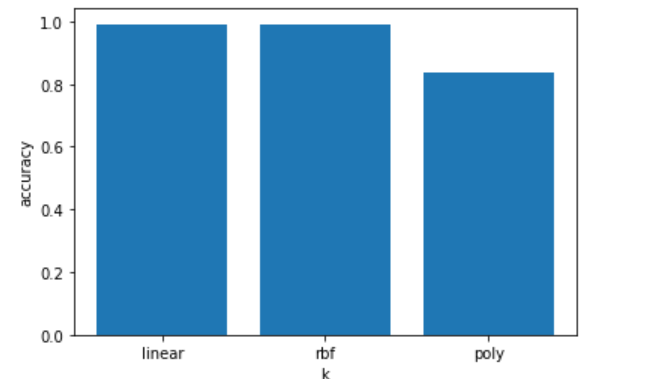


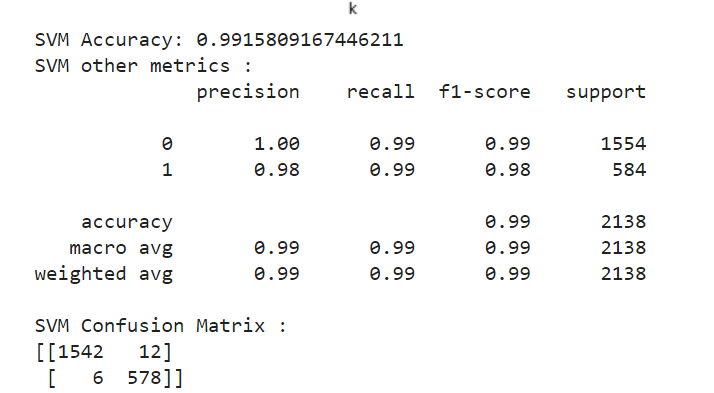
clf=RandomForestClassifier(n\_estimators=28) # 3-30 arası değerleri denedii en yüksek doğruluğu 18 verdi  
clf.fit(X\_train,y\_train)  
y\_pred=clf.predict(X\_test)  
print("Random Forest Accuracy:",metrics.accuracy\_score(y\_test, y\_pred))  
report = metrics.classification\_report(y\_test, y\_pred)  
sonuc.append(metrics.accuracy\_score(y\_test, y\_pred))  
print("Random Forest other metrics :")  
print(report)  
matrix = metrics.confusion\_matrix(y\_test, y\_pred)  
print("Random Forest Confusion Matrix :")  
print(matrix)



3. SVM

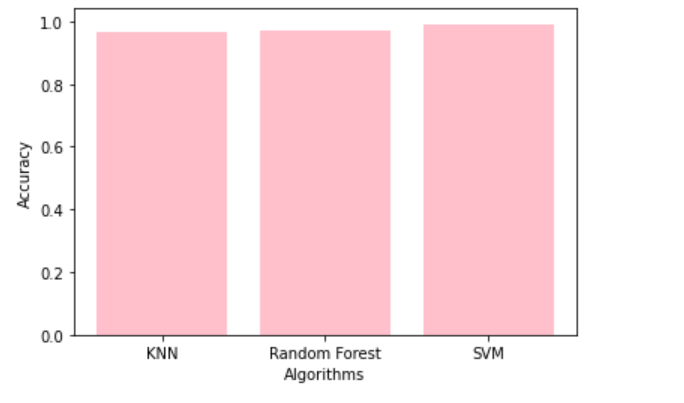
Bu algoritmada, her bir veri maddesini belirli bir koordinatın değeri olan her özelliğin değeri ile birlikte n-boyutlu boşluğa (burada n sahip olduğunuz özelliklerin sayısı) bir nokta olarak çizilir. Ardından, iki sınıftan oldukça iyi ayrım yapan hiper-düzlemi bularak sınıflandırma gerçekleştirilir. Projemizde rbf, linear ve poly kernel tipleri denenip linear seçilmiştir.

kernel\_type=["linear","rbf","poly"]  
scores = []  
for k in kernel\_type:  
 svmclf = svm.SVC(kernel=k,probability=True)  
 svmclf.fit(X\_train, y\_train)  
 y\_pred3 = svmclf.predict(X\_test)  
 scores.append(metrics.accuracy\_score(y\_test, y\_pred3))  
plt.bar(kernel\_type, scores)  
plt.xlabel('k')  
plt.ylabel('accuracy')  
plt.show()

print("SVM Accuracy:",metrics.accuracy\_score(y\_test, y\_pred3))  
report3 = metrics.classification\_report(y\_test, y\_pred3)  
sonuc.append(metrics.accuracy\_score(y\_test, y\_pred3))  
print("SVM other metrics :")  
print(report3)  
matrix3 = metrics.confusion\_matrix(y\_test, y\_pred3)  
print("SVM Confusion Matrix :")  
print(matrix3)

Doğruluk Oranlarının Karşılaştırmalı Grafiği

Kullandığımız 3 adet algoritmayı karşılaştırdığımızda spam mailleri belirleme oranı en yüksek olan algoritmanın SVM algoritması olduğunu görmekteyiz

plt.bar(algo,sonuc,color="salmon")  
plt.xlabel('Algorithms')  
plt.ylabel('Accuracy')  
plt.show()