**DOĞAL DİL İŞLEME**

**[KONU: GOOGLE ÇEVİRİ API]**

**[Betül SARITEKE]**

**2022**

İÇİNDEKİLER TABLOSU

Giriş 2

Gelişme 2

**1. Google Translate Apı Kullanarak Veri Çoğaltma** 2

1.1 Kullanılan Kütüphaneler: 2

1.2 İşlemler ve Kodlar:: 2

**2. Supervısed Learnıng (Denetimli Öğrenme) Makine Öğrenmesi ile Sınıflandırma**5

2.1 Kullanılan Kütüphaneler: 5

2.2 İşlemler ve Kodlar:: 6

Sonuç10

**3. Değerlendirme ve Yorumlar** 10

Ekler 14

# **Giriş**

Metin sınıflandırma, genellikle kısa metinlerin (haber başlıkları, Twitter gönderileri vb. gibi) içerikleri doğrultusunda belirli sınıflara ayrılmasıdır. Bu amaçla oluşturulan veri kümeleri bir metin parçası ve sınıfı olmak üzere iki kısımdan oluşur. Metin sınıflandırma işlemi doğası gereği denetimli (supervised) makine öğrenmesi algoritmaları ya da derin öğrenme kullanılarak yapılır. Bu çalışma makine öğrenmesi ile yapılmıştır. Kullandığım veri setinden bir parça şu şekilde:

|  |  |  |
| --- | --- | --- |
| id | product | narrative |
| 0 | credit\_card | Kredi kartımın bilgileri halka açık istenmeden kullanılıyor kredi kartımı iptal edilmesini talep ediyorum. |
| 1 | credit\_reporting | Kredi izleme veya kimlik koruması, Dolandırıcılık uyarılarıyla ilgili sorun yaşıyorum. |
| 2 | mortgages\_and\_loans | Konut kredisi veya kredi limiti, Kredi kararı ile ilgili bilgi istiyorum. |
| 3 | debt\_collection | Devam eden alacaklı olmayan borçları tahsil etmek isteniyor ancak borç benim değil borcu ödemek istemiyorum. |

Google Çeviri anlamsal içeriğe dayalı çeviriler üretmektedir. Bu nedenle çevirinin çevirisi şeklinde kurulan bir zincirde aynı anlamlı birden çok cümle oluşmakta ve her adımda çeviriler birbirinden farklılaşır. Bu yapı ile öğrenme verilerini arttırdım.

Bu yapıyı ve denetimli öğrenme ile sınıflandırma ve model oluşturma öğrenme sağlamayı sırasıyla ve adımları ile birlikte açıkladım.

# **Gelişme**

1. **GOOGLE TRANSLATE API KULLANARAK VERİ ÇOĞALTMA**

Elimizde bulunan verileri, çeşitli tekniklerle değiştirerek yeni ve sentetik veri oluşturma işlemine veri artırımı (data augmentation) denir. Çeviri zincirin olurşturulur, ardından da verilen veri kümesi içine aynı sınıf etiketiyle ekler . Bu çalışmada Google’ın sunduğu kütüphane ile python dilinde bu işlemi yapıyorum.

1.1 Kullanılan kütüphaneler:

#*python library*

*from* googletrans *import* Translator

*import* numpy *as* np, pandas *as* pd

*from* dask *import* bag, diagnostics

1.2 İşlemler ve Kodlar:

Translate işlemi cümleyi birden fazla dile çevirerek cümleyi anlamı değişmeden farklı yapılara dönüştürür sonuç olarak elimizde aynı anlama sahip birden fazla cümle olur bu şekilde verileri arttırmış oluruz.

# *cümleyi dillere göre çevirir ardından ilk tespit edilen dile göre listeler*

    def *Translate\_Google*(sequence, PROB = 1):

        languages = ['en', 'fr', 'tr', 'ru', 'bg', 'de', 'es', 'el']

        translator = Translator()

        org\_lang = translator.detect(sequence).lang

        random\_lang = np.random.choice([lang *for* lang *in* languages *if* lang is not org\_lang])

        translated = translator.translate(sequence, dest = random\_lang).text

        translated\_back = translator.translate(translated, dest = "tr" ).text

*if* np.random.uniform(0, 1) <= PROB:

            output\_sequence = translated\_back

*else*:

            output\_sequence = sequence

*return* output\_sequence

Kullanacağım verile csv uzantılı dosyada olduğu için öncelikle dosyadaki satırları tek tek aldım ve ayırarak narrative ve product sütunlarına göre translate fonksiyonunu kullandım. Product sütunu çeviriye eklemedim çünkü denetimli öğrenme işleminde işaretlenmiş verilere ihtiyacım var. Sınıf ismi değişmemeli.

def *CSV\_Translate\_Google*(dataset):

        narrative\_list =  bag.from\_sequence(dataset['narrative'].tolist()).map(Translate\_Google)

        product\_list =  bag.from\_sequence(dataset['product'].tolist())

*with* diagnostics.ProgressBar():

            translate\_narrative = narrative\_list.compute()

            product = product\_list.compute()

        print("İşleme Başlatıldı Lütfen Bekleyiniz!....")

        row\_narrative = translate\_narrative

        row\_product = product

        new\_data=[]

        new\_product=[]

*for* x *in* range(5) :

*for* i *in* range(len(row\_narrative)):

                    output = Translate\_Google(row\_narrative[i])

                    new\_data.append(output)

                    new\_product.append(row\_product[i])

        veriler = {"product": new\_product, "narrative": new\_data}

        dataset = pd.DataFrame(veriler)

*return* dataset

Tüm satırları okur ve çevirir çoğaltılan verileri yeni csv dosyasına ekler. Bunun için python dictionary yapısına product ve narrative listesini ekledim ve DataFrame fonksiyonu ile csv dosyası için uygun formata dönüştürdüm.

Kullandığım dosyayı tanımladım ve yeni oluşacak dosyaya isim verdim. Son olarak ekranaişlem ile ilgili bilgi veriyor.

  data\_csv = pd.read\_csv('complaints.csv')

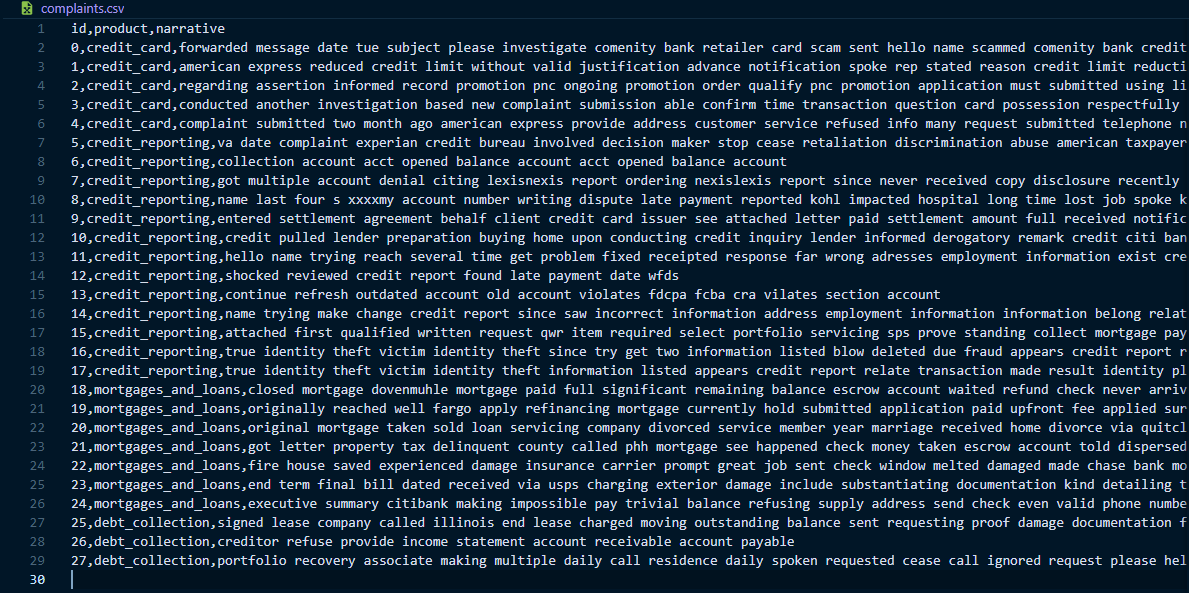
    yeni\_data = data\_csv.pipe(CSV\_Translate\_Google)

    yeni\_data.to\_csv('Translate\_complaints.csv')

    print("İşleme Tamamlandı!....")

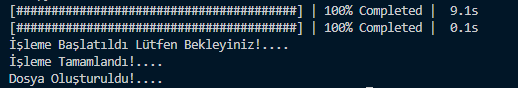
    print("Dosya Oluşturuldu!....")

Complaint.csv:



(29 satır)

**Kodun terminal çıktısı:**



**Oluşan yeni csv dosyası ( Translate\_complaints.csv ):**



(155 satır)

**2) SUPERVISED LEARNING (DENETİMLİ ÖĞRENME) MAKİNE ÖĞRENMESİ İLE SINIFLANDIRMA**

Veri seti Complaints.csv içerisinde üç faklı veri bulunuyor. Eşsiz id verisi. Kullanıcı yorum ve şikayetlerinin bulunduğu narrative sütunu ve bunların ait olduğu etiketler (sınıflar). Her narrative için bir adet sınıf bulunuyor toplam dört farklı sınıf var bunlar:

* credit\_card (kredi kartı ile ilgili olanlar)
* credit\_reporting (kredi raporlarıyla ilgili olanlar)
* mortgages\_and\_loans (ipotekler ve kredi sorunları ile ilgili olanlar)
* debt\_collection (borç işlemleri ilgili olanlar)

Elde ettiğim yeni çoğaltılmış veriler ve eski verileri istatistiksel bir modeli eğitmek için kullanıyorum. Birden fazla yöntemle model oluşturdum ve daha sonra analiz için etiketlenmemiş test verisi girerek öğrenmenin ne kadar doğru olduğunu test ediyorum.

2.1 Kullanılan Kütüphaneler:

#*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Kullanılan Kütüphaneler \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#*

*import* numpy *as* np

*import* pandas *as* pd

*import* seaborn *as* sns

*import* nltk

*import* matplotlib.pyplot *as* plt

*from* sklearn.feature\_extraction.text *import* TfidfVectorizer

*from* sklearn.model\_selection *import* train\_test\_split

*from* sklearn.feature\_extraction.text *import* CountVectorizer

*from* sklearn *import* preprocessing

*from* sklearn.feature\_selection *import* chi2

*from* sklearn.model\_selection *import* train\_test\_split

*from* sklearn.feature\_extraction.text *import* TfidfTransformer

*from* sklearn.naive\_bayes *import* MultinomialNB

*from* sklearn.linear\_model *import* LogisticRegression

*from* sklearn.ensemble *import* RandomForestClassifier

*from* sklearn.svm *import* LinearSVC

*from* sklearn.model\_selection *import* cross\_val\_score

*from* sklearn.model\_selection *import* train\_test\_split

*from* sklearn.metrics *import* confusion\_matrix

*from* sklearn *import* metrics

*import* string

*from* nltk.corpus *import* stopwords

nltk.download('stopwords')

pd.options.mode.chained\_assignment = None

2.2 İşlemler ve Kodlar:

Öncelikle verilerimin olduğu dosyayı tanımladım ve önişleme adımlarını uyguladım. Önişleme adımlarını sadece narrative satırlarına uyguladım. Yaptığım işlemler sırasıyla:

* Tüm harfleri küçük harfe dönüştürme
* Noktalama işaretlerini silme
* Stop Word kaldırma
* Gereksiz boşlukları silme

#*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* data \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*#*

Data = pd.read\_csv('complaints.csv',low\_memory=False )

#*################## ön işleme ###############################3*

dt = Data[["narrative"]]

dt["narrative"] = dt["narrative"].astype(str)

dt["text\_lower"] = dt["narrative"].str.lower()

dt.drop(["text\_lower"], axis=1 , inplace=True)

PUNCT\_TO\_REMOVE = string.punctuation

def *noktalamaSil*(text):

*return* text.translate(str.maketrans('', '', PUNCT\_TO\_REMOVE))

dt["text\_data"] = dt["narrative"].apply(lambda text: noktalamaSil(text))

STOPWORDS = set(stopwords.words('english'))

def *stopwordsSil*(text):

*return* " ".join([word *for* word *in* str(text).split() *if* word not in STOPWORDS])

dt["text\_wo\_stop"] = dt["text\_data"].apply(lambda text: stopwordsSil(text))

#*######################################################################*

Veriler şimdi daha temiz ve kullanıma daha uygun bir formatta oldu. Ardından tablodaki narrative ve product için işe yaramayan sütunları eliyoruz böylece işaretli olmayan sütunlar kullanıma dahil edilmeyecek. Bir yeni sınıf etiketi ekiliyoruz bu elimdeki sınıfları ayırmak için kullanılan yeni bir tanımlayıcı görevi görecek. Category\_id product sayısına göre atandı yani dört tanedir.

Son olarak tüm data içindeki product dağılımı ve tüm verilerin sayısını yazdırıyoruz. Narrative product dağılımını görsel tablo olarak çıktısını alıyoruz.

#*######################################################################*

*#print(Data)*

print(Data.dtypes)

print(pd.notnull(Data['narrative']).value\_counts()) #*# işe yaramayan kullanılmayacak olan ögeler elenir*

col = ['product', 'narrative']

Data = Data[col]

print(Data.columns) #*#data için*

Data.columns = ['product', 'narrative']

Data['category\_id'] = Data['product'].factorize()[0]

category\_id\_df = Data[['product', 'category\_id']].drop\_duplicates().sort\_values('category\_id')

category\_to\_id = dict(category\_id\_df.values)

id\_to\_category = dict(category\_id\_df[['category\_id', 'product']].values)

print ("---------- Data Gruplandırması ------------------------")

print(Data.groupby('product').narrative.count())

print ("-------------------- Data  -----------------------------")

print(Data.head(10))

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

Data.groupby('product').narrative.count().plot.bar(ylim=0)

plt.show()

Model yalnızca sayısal verileri işleyebildiğinden, TfidfVectorizer ve CountVectorizer fonksiyonlarını metin verilerini vektörlere dönüştürmek kullandım. CountVectorizer'da yalnızca bir kelimenin belgede kaç kez göründüğünün çıktını veriyor. TfidfVectorizer'da bir kelimenin genel belge ağırlığını veriyor. En sık kullanılan kelimeleri tespit etmek için kullanıyorum.

#*##########################  Count vect - Tf-Idf vect ##############*

print ("----------- Count vect - Tf-Idf vect ---------------")

tfidf\_vect = TfidfVectorizer(analyzer='word', token\_pattern='\w{1,}', max\_features=5000)

tfidf\_vect.fit(Data['narrative'].astype('U').values)

Features = tfidf\_vect.transform(Data['narrative'].astype('U').values)

encoder = preprocessing.LabelEncoder()

Labels1 = encoder.fit\_transform(Data['product'])

print(Features.shape)

print(Features[0], Labels1)

print("\n")

Ardından filtreleme işlemi yapıyorum böylece özellik ile hedef değişken arasındaki ilişkiye göre seçim yapar Ki-Kare Testi için (Chi2) fonksiyonuyla testi girdi ve çıktı iki kategorik değişken arasında istatistiksel anlamlılık düzeyinde ilişki olup olmadığını ölçümler.

Bu da çıktı olarak ilişki düzeyine göre uniagram ve biagramları verir ancak elimdeki veri sayısı az olduğu için biagramları tespit etme konusunda yetersiz kaldı.

#*######################  filtreleme  ##############################*

N = 2

*for* product, category\_id *in* sorted(category\_to\_id.items()):

  features\_chi2 = chi2(Features, Labels1 == category\_id)

  indices = np.argsort(features\_chi2[0])

  feature\_names = np.array(tfidf\_vect.get\_feature\_names\_out())[indices]

  unigrams = [v *for* v *in* feature\_names *if* len(v.split(' ')) == 1]

  bigrams = [v *for* v *in* feature\_names *if* len(v.split(' ')) == 2]

  print("# '{}':".format(product))

  print("  . En ilişkili unigramlar:\n       . {}".format('\n       . '.join(unigrams[-N:])))

  print("  . En ilişkili biagramlar:\n       . {}".format('\n       . '.join(bigrams[-N:])))

Giriş verilerini sayısal olarak kodladım fit\_transform kelime hazinesini ve one-hot öğrenir ve transform aynı kelime hazinesini ve one-hot kodlamalarını kullanır. Çıktı olarak eğitim setinin boyutlarını yazdırır. Son olarak değerlendirme için test verisi olarak elle örnek girdim.

#*## Öğrenme Traning Count vect - Tf-Idf vect – MultinomialNB ##############*

print ("-------------------------------------------------")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(Data['narrative'], Data['product'], random\_state = 0)

count\_vect = CountVectorizer()

X\_train\_counts = count\_vect.fit\_transform((X\_train).astype('U').values)

tfidf\_transformer = TfidfTransformer()

X\_train\_tfidf = tfidf\_transformer.fit\_transform((X\_train\_counts).astype('U'))

clf = MultinomialNB().fit(X\_train\_tfidf, y\_train)

#*############### Test örnekleri #############################*

print(clf.predict(count\_vect.transform(["test için örnek1"])))

print(clf.predict(count\_vect.transform(["test için örnek2"])))

Data[Data['narrative'] == "test için örnek1 "]

Data[Data['narrative'] == "test için örnek2 "]

#Not: örnekler uzun olduğu için buraya eklemedim. Kod içerisinde mevcut.

Verideki etiketlerle narrative verisini birer değişkene atayarak eğitim ve test için veriyi ayrıdım. Model oluşturma aşamasında kullandığım yöntemleri sırasıyla test ederek değerlerini ve doğruluk tablosunu yazdırdım. Model olarak kullandığım yöntemler:

* Random Forest Sınıflandırıcı
* Doğrusal Destek Vektör Sınıflandırıcısı
* Naive Bayes
* Lojistik Regresyon

#*######### Model kontrol ve değerlendirme #################*

print ("############## Model #################")

models = [

    RandomForestClassifier(n\_estimators=200, max\_depth=3, random\_state=0),

    LinearSVC(),

    MultinomialNB(),

    LogisticRegression(random\_state=0),

]

CV = 5

cv\_df = pd.DataFrame(index=range(CV \* len(models)))

entries = []

*for* model *in* models:

  data\_google\_model = model.\_\_class\_\_.\_\_name\_\_

  accuracies = cross\_val\_score(model, Features, Labels1, scoring='accuracy', cv=CV)

*for* fold\_idx, accuracy *in* enumerate(accuracies):

    entries.append((data\_google\_model, fold\_idx, accuracy))

cv\_df = pd.DataFrame(entries, columns=['data\_google\_model', 'fold\_idx', 'accuracy'])

#*########################### tablo #####################*

sns.boxplot(x='data\_google\_model', y='accuracy', data=cv\_df)

sns.stripplot(x='data\_google\_model', y='accuracy', data=cv\_df,

              size=8, jitter=True, edgecolor="gray", linewidth=2)

plt.show()

print(cv\_df.groupby('data\_google\_model').accuracy.mean())

print ("####################################################################")

Doğru sınıflandırılmış ve yanlış sınıflandırılmış verileri heatmap yapısı ve karışıklık matrisi ( confusion matrix) kullanarak tablo haline getirdim. Böylece Kullandığım her model için değerleri karşılaştırarak en uygun modeli seçebilirim.

#*################ conf-matris ##############################*

model = LinearSVC()

X\_train, X\_test, y\_train, y\_test, indices\_train, indices\_test = train\_test\_split(Features, Labels1, Data.index, test\_size=0.33, random\_state=0)

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

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

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

fig, ax = plt.subplots(figsize=(8,6))

sns.heatmap(conf\_mat, annot=True, fmt='d', xticklabels=Data['product'].unique(),yticklabels=Data['product'].unique())

plt.ylabel('Actual')

plt.xlabel('Predicted')

plt.show()

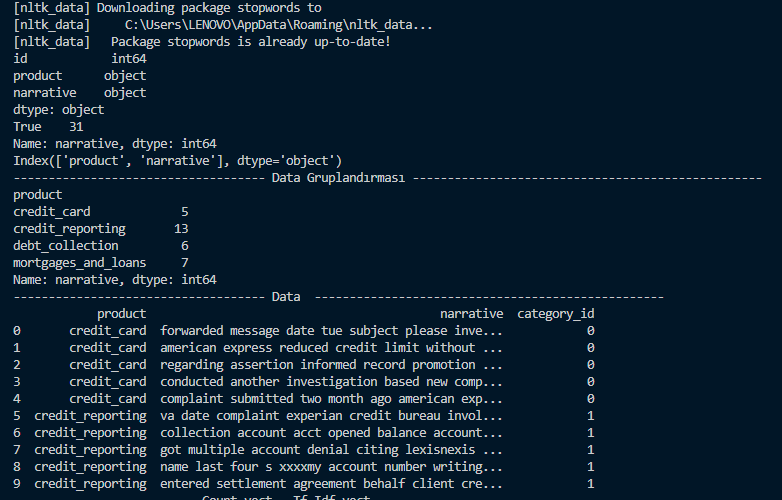
print(model.fit(Features, Labels1))

print(metrics.classification\_report(y\_test, y\_pred, target\_names=Data['product'].unique()))

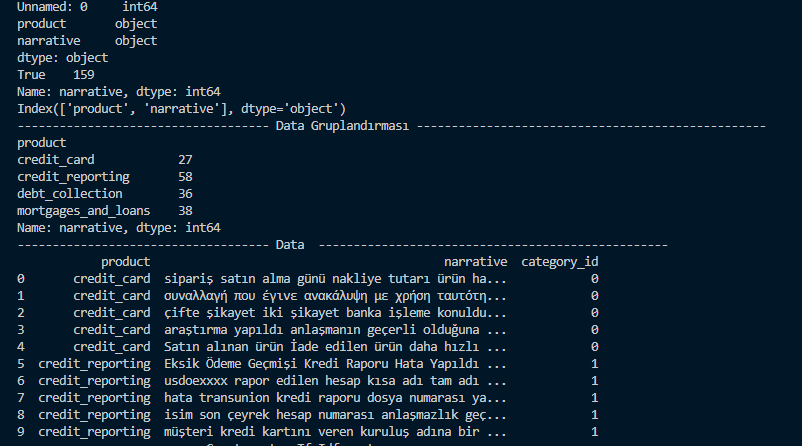
# **Sonuç**

1. **DEĞERLENDİRME VE YORUMLAR**

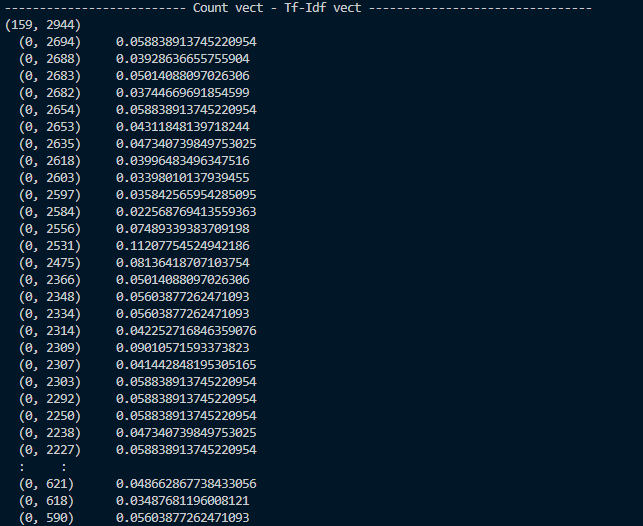
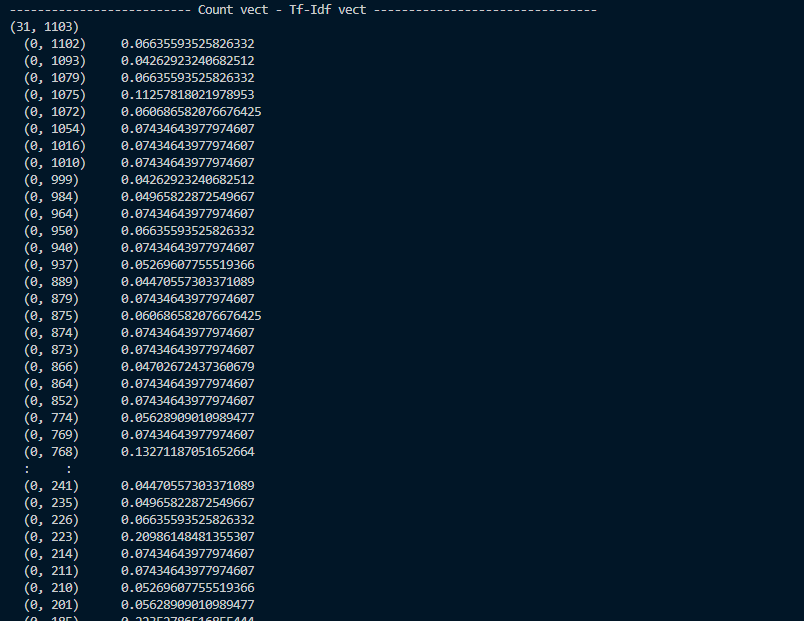
Veri arttırma işlemi öncesi data analizi sınıfların gruplandırması önişleme çıktısı ve yeni etiket atanan dataları terminale yazdırıyoruz.



Translate ile verileri arttırma data analizi sınıfların gruplandırması önişleme çıktısı ve yeni etiket atanan dataları terminale yazdırıyoruz.

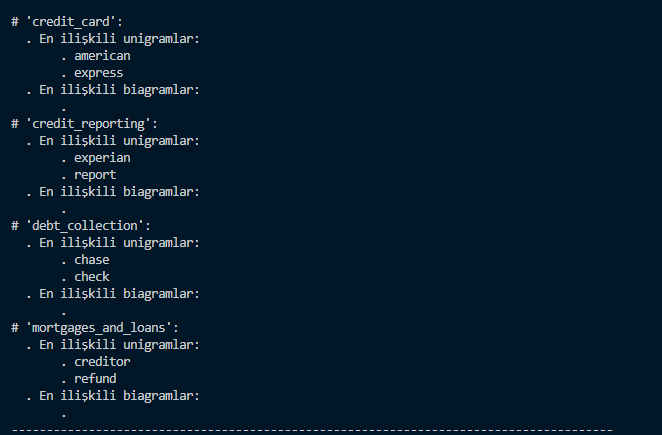
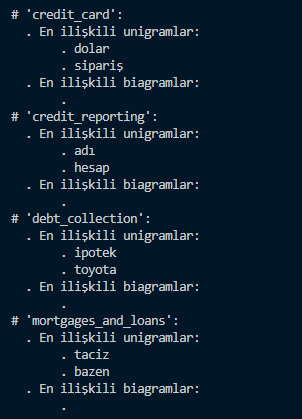


TfidfVectorizer ve CountVectorizer fonksiyonlarının çıktısı veri arttırma öncesi ve Veri arttırdıktan sonra Burada her iki vektörde de veri sayısı arttıkça değerlerin değiştiği ve frekansların daha doğru belirlendiği görülüyor.



Bu da çıktı olarak ilişki düzeyine göre uniagram ve biagramları verir ancak elimdeki veri sayısı az olduğu için biagramları tespit etme konusunda yetersiz kaldı.

Önce: Sonra:

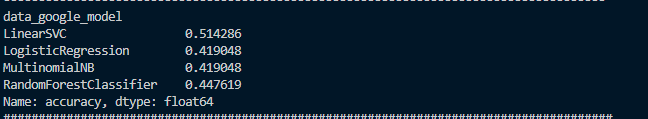
Test örnekleri çıktı veri arrtırmadan önce biri hatalı diğeri doğru olarak sınıflandırıldı. Sonrasında ise iki örnek de doğru olarak sınıflandırıldı:

Önce: Sonra:

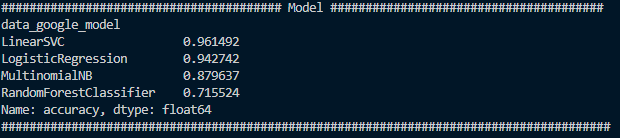
 

**Model Değerlendirmeleri:**

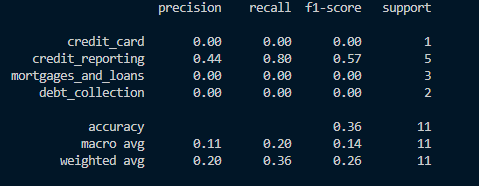
Translate öncesi tüm model çeşitlerinin değerleri birbirine yakındır modeller arasında uyumluluk için seçim yapmak zor.



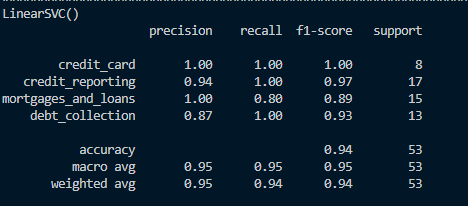
Veri artırımı sonrasında iyi uyum sağlayan iki model vardır. Doğrusal Destek Vektör Sınıflandırıcısı ve Lojistik regresyon değerleri veriye uyumluluğu gösterir.



Veri artırımı sonrası ve öncesi doğruluk değerlendirme: Genel doğruluk 0.36 çıkmıştır bu iyi bir değer değildir elimdeki veriler ile yeterince öğrenme sağlanamamıştır. 1,3 ve 4 sınıflar için sınıflandırma doğru yapılmamıştır. Bu kategorilerin yanlış sınıflandırılması hatalıdır sorunları çözemeyecekler ve uygun sınıfa eklenmeyecektir.

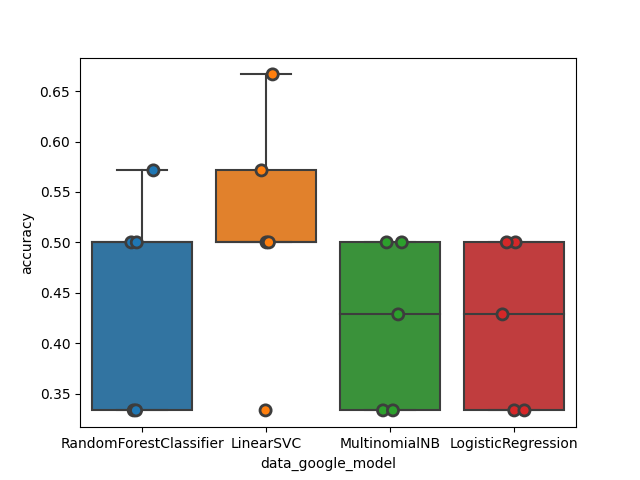
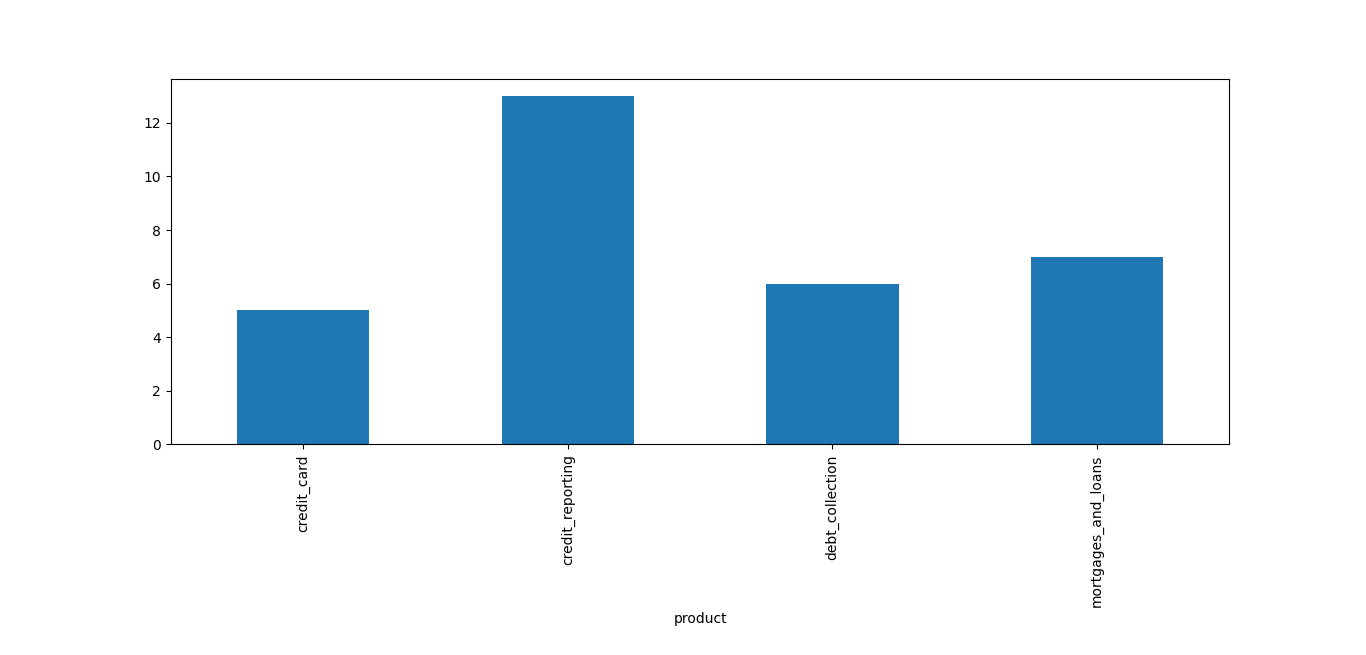


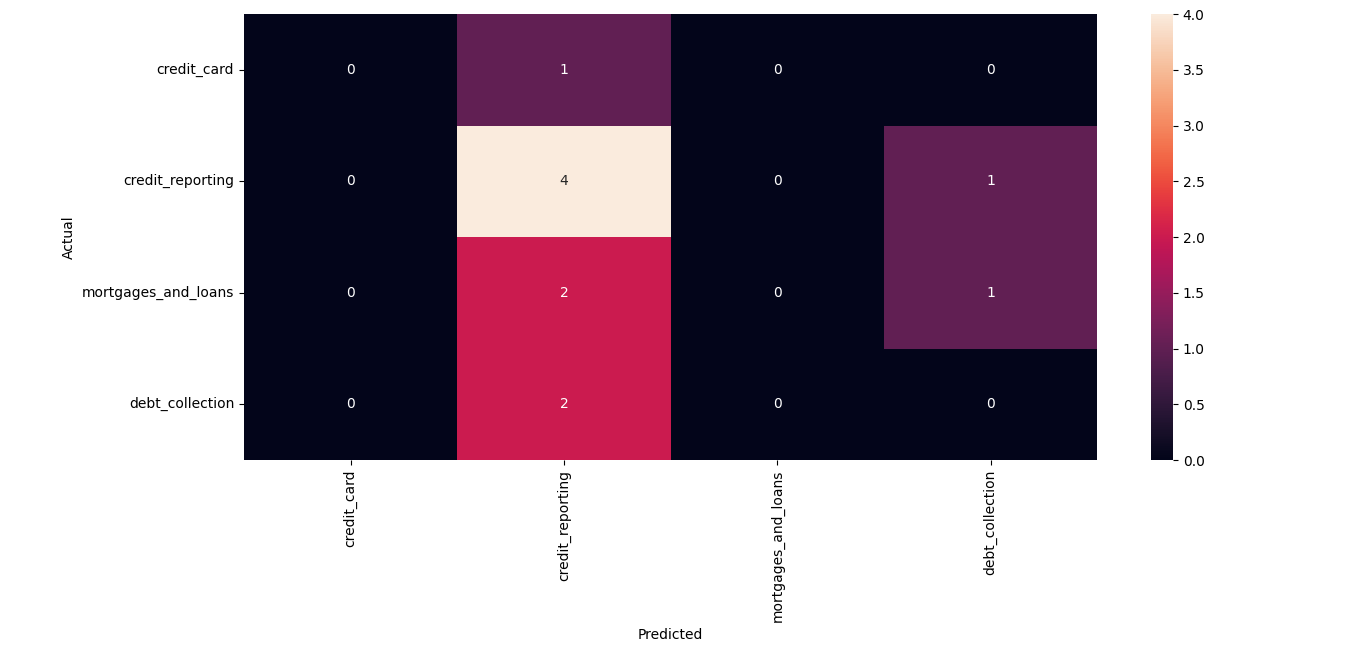
Sonrasında sonuçlar: Doğruluk değeri 0.94 çıkmıştır bu iyi görünen fakat sorun oluşturabilecek bir değerdir. Aşırı uyum sağlamış olabilir. Bu durum da istenmeyen bir durumdur. Neredeyse tüm sınıflar için sınıflandırma değerleri çok yüksektir sınıflandırma sağlanmıştır. Veri artırımı sonrası model ve değerlendirme sonuçları çok daha iyi duruma gelmiştir daha yüksek miktardaki veri sayısı ile aşırı uyumluluk sorunu da giderilebilir.



# **Ekler:**

**TRANSLATE İLE VERİ ARTIRIMI ÖNCESİ ÇIKTI TABLOLARI:**





**TRANSLATE İLE VERİ ARTIRIMI SONRASI ÇIKTI TABLOLARI:**

