RAPOR

KONU: machine learning malware detection

Açıklama: makine öğrenimi ile oluşturulmuş bir dedektörün malware ve zararı olmayanlardan string ve iat tablolarından elde etmiş olduğu bilgiler ile analiz yapıp test edilen dosyanın zararlı olup olmadığına karar veriyor.

Öncellikle projem boyunca malwaredatascience adlı kitaptan yararlanıp onların sağlamış olduğu işletim sistemindeki kaynaklardan yararlandım. Bu kitapta olan machine learning malware detector kodundaki stringe göre ayıklama yapan sisteme öğretmiş oldukları iat tablolarından da bilgi çıkarıp analize işlemine katıp ilerlettim.

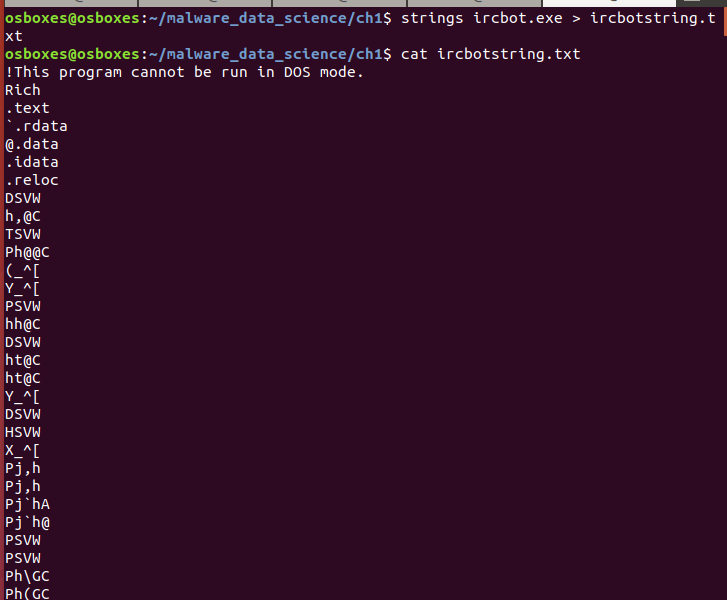
Şimdi konu konu size anlatıp en son kodların üzerinden geçelim

Statik analiz:Bir programın dosyasının disassembled kodunun , grafik görüntülerinin , yazdırılabilir dizelerin strings ve diğer disk üzerindeki kaynakların analiz edilmesiyle gerçekleşir. Programı çalıştırmadan reverse engineering tersine mühendisliği ifade eder. Statik analizin eksikleri olsa da bize bir zararlıyı anlamamızda yardımcı bilgiler verir.Statik analiz teknikleri çoğu windows programı tarafından kullanılan portable executable(PE) dosya formatından bilgi çıkarma, içe aktarma analizi describe techniques import analysis, grafik görüntü analizi , graphical image analysis ve dizi analizi strings analysis tekniklerinden bahsedebiliriz.

Benim projemde analiz yapmış olduğum içe aktarma analizi describe techniques import analysis ve stringsleri kullandım. malware detection sistemler için kullanılan scifit learn açık kaynak makine öğrenimi paketinden yararlandım.

-String Features:Dize Özellikleri:

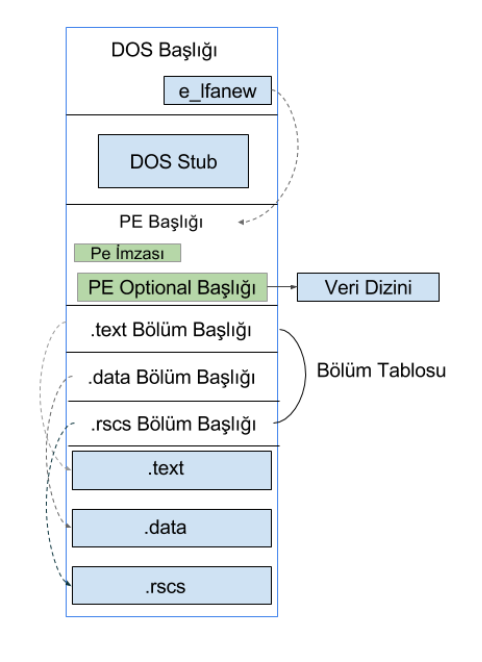
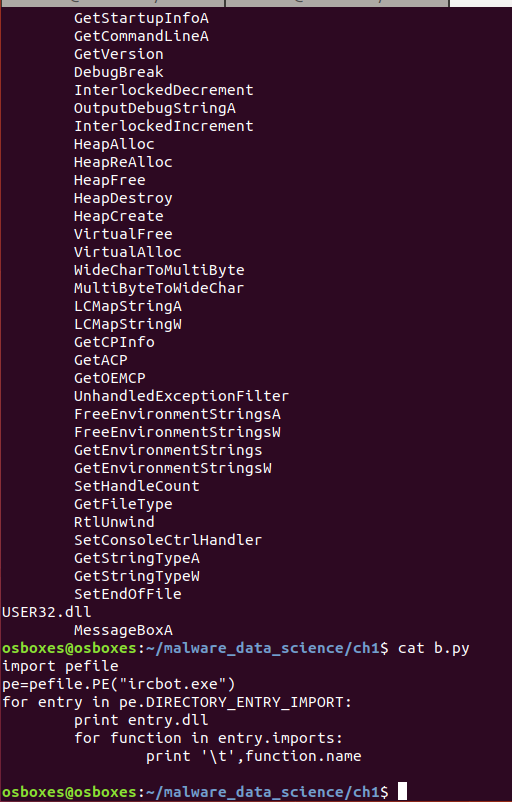
Dizeleri çıkarıp en az karakteri belirleyip eleyip fazla dizeleri alıyoruz. String özelliklerinin sklearnın anlayabileceği bir biçime dönüştürmek için onları bir Python sözlüğüne koymamız lazım. Bunu, gerçek dizeleri sözlük anahtarları olarak kullanarak ve ardından söz konusu ikilinin bu dizeyi içerdiğini belirtmek için değerlerini 1 olarak ayarlayarak yaparız. Örneğin, önceki örnek ikili, {"PE yürütülebilir": 1, "Kötü amaçlı yük": 1} özellik vektörünü alacaktır. Elbette, çoğu yazılım ikili dosyasında yalnızca iki değil, yüzlerce yazdırılabilir dize bulunur ve bu dizeler, bir programın ne yaptığı hakkında zengin bilgiler içerebilir.



- Import Address Table (IAT) Features:

IAT, bir yazılım ikili dosyasının harici DLL dosyalarından içe aktardığı işlevlerin ve kitaplıkların bir listesini içerir. Bu nedenle, IAT, PE başlık özelliklerini tamamlamak için kullanabileceğiniz program davranışı hakkında önemli bilgiler içerir.

IAT'yi makine öğrenimi özelliklerinin kaynağı olarak kullanmak için, her dosyayı bir özellikler sözlüğü olarak temsil etmeniz gerekir; burada içe aktarılan kitaplığın ve işlevin adı anahtar, anahtar ise dosyanın söz konusu belirli içe aktarmayı içerir (örneğin, "KERNEL32.DLL: LoadLibraryA" anahtarı, burada KERNEL32.DLL DLL ve LoadLibraryA işlev çağrısıdır). Bir örnek için IAT özelliklerinin bu şekilde hesaplanmasından elde edilen özellik sözlüğü, bir ikili dosyada gözlemlenen herhangi bir tuşa 1 atayacağımız {KERNEL32.DLL: LoadLibraryA: 1, ...} gibi görünür.

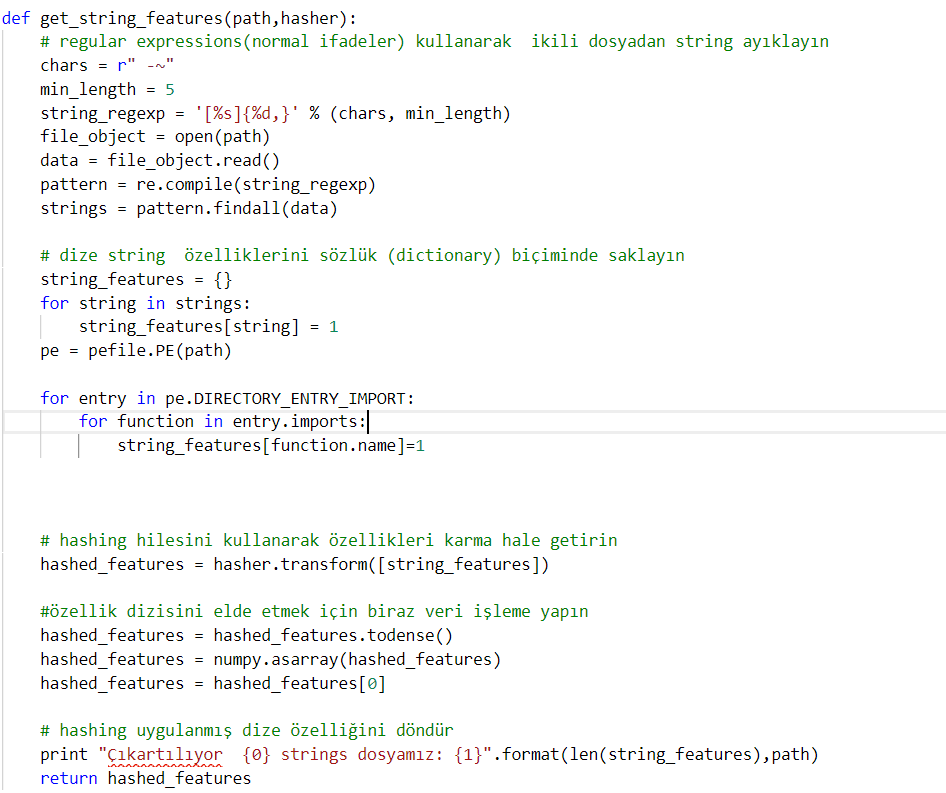


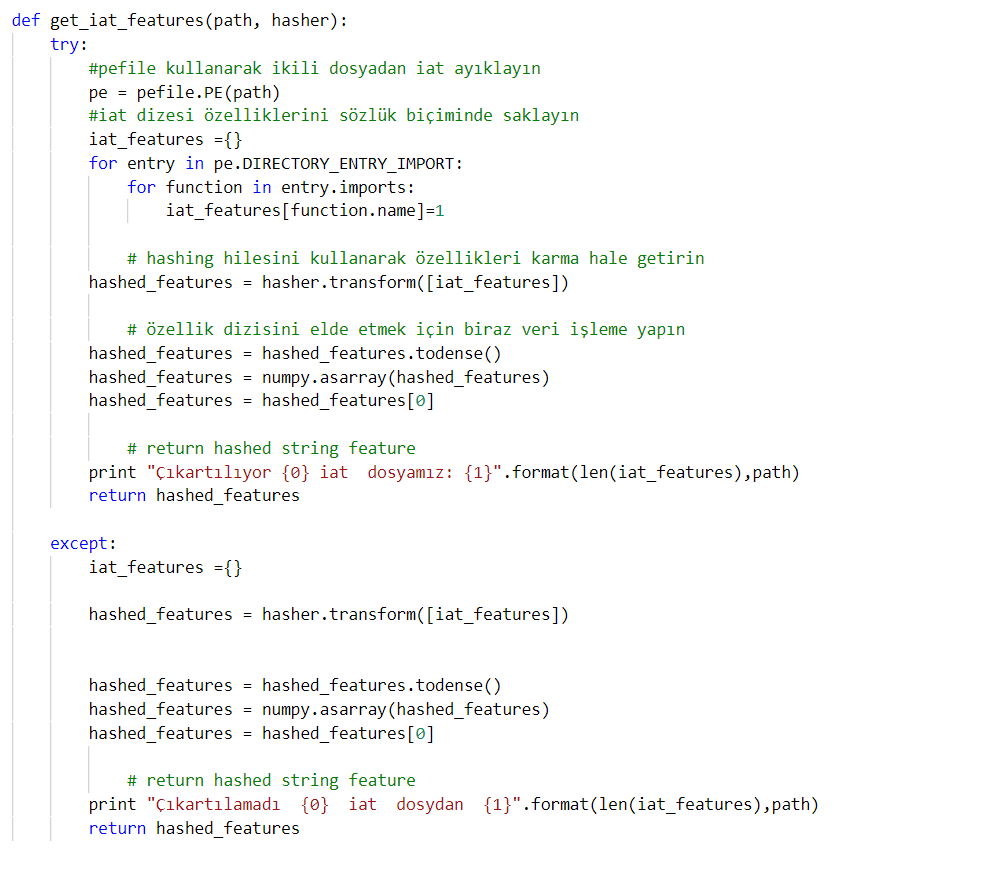
Gerçek dünya dedektörlerimizi oluştururken makinemizi eğitmek için birden fazla örnek kullanırız. Ve dolayısıyla milyonlarca belki özellik çıkar.Bunların analizi ve saklanmasında problemler çıkar. Bunu önlemek için kitapta bahsedilen özellik sıkıştırmak için hashing işlemi hilesini kullandım. Zaten sklearn’ün bunun için bir FeatureHasher adlı sınıfı var. Bunu

“from sklearn.feature\_extraction import FeatureHasher” diye kodumuza dahil ederiz.” hasher = FeatureHasher(n\_features=2000)” şeklinde kullanabiliriz.Bu bize 2000 boyutlu bir dizi verir. “hashed\_features = hasher.transform(features)” diyerek kendi özelik arrayimizi bu arraye dönüştürmüş oluruz.

Şimdi kendi dedektörümüzün kodlarına geçelim. gerçek dünya dedektörümüzün üç şey yapması gerekecektir: eğitim ve tespitte kullanmak için yazılım ikili dosyalarından özellikleri çıkarmak, eğitim verilerini kullanarak kötü amaçlı yazılımları tespit etmek için kendini eğitmek ve yeni yazılım ikili dosyalarında gerçekten algılama gerçekleştirmek. Kodları açıklayabilmek için visual stdioda açıklamalarını Türkçeleştirdim.Ama bana sağlanan işletim sisteminde Türkçe karakterlerden dolayı hata alacağımdan dolayı orada açıklamalar İngilizcedir.

\*Özellikleri Çıkarmak:

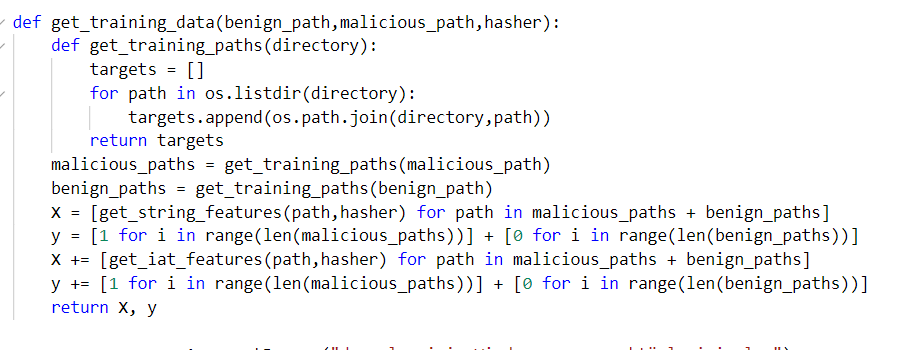




Dedektörümüzü oluşturmak için uyguladığımız ilk şey, eğitim ikili dosyalarından özellikleri çıkarmak için koddur. Özelliklerin ayıklanması, eğitim ikili dosyalarından ilgili verilerin çıkarılmasını, bu özelliklerin bir Python sözlüğünde saklanmasını ve daha sonra, benzersiz özelliklerimizin sayısının engelleyici bir şekilde artacağını düşünürsek, sklearn'ın karma hile uygulamasını kullanarak bunları dönüştürmeyi içerir.Burada fonksiyonun argümanları hedef ikilinin yolunu ve sklearn hash sınıfını alır. Ardından get\_string\_features ve get\_iat\_features hedef dosyanın özelliklerini ayrıştırır. Saha sonra sözlüğe eklenip değerini 1 olarak ayarlarlar.Ardıdan hasher ile özellikleri hash hale getiriyoruz. Sonra verilerimizi vektör haline todense() ve asarray() ile dönüştürüp hashed\_heatures özellik vektörünü geri döndürürüz.

\*Training the Detector:Dedektörü Eğitme:

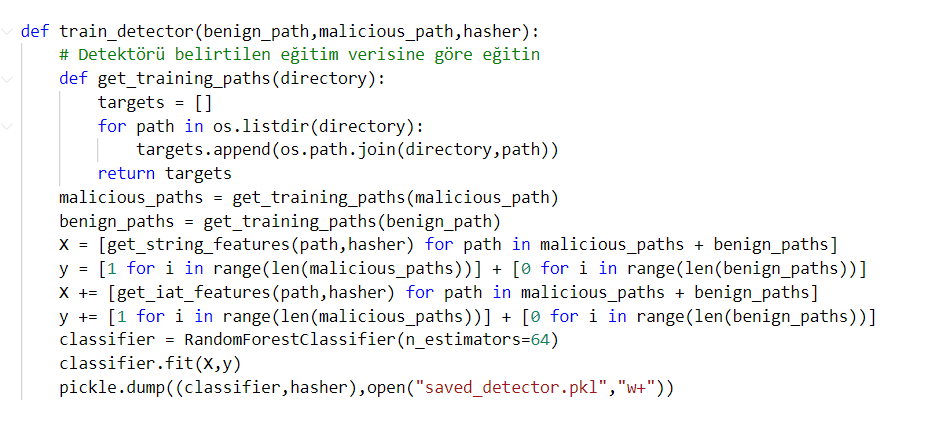
Sklearn makine eğitim kısmında bize çok yardımcı oluyor. Özellikleri çıkardıktan sonra kullanmak istediğimiz özellik hasherini ve sklearn makine öğrenimi algılayıcısını somutlaştırmamız gerekir (bu durumda rastgele bir orman sınıflandırıcı kullandım). Daha sonra, örneklerin ikili dosyaları üzerinde eğitmek için detektörde sklearn'ın fit yöntemini çağırmamız gerekir. Son olarak, detektörü ve özellik hasher'ını diske kaydederiz, böylece ileride dosyaları taramak istediğimizde onları kullanabiliriz.



Sağladığımız eğitim örneklerinden özellikleri çıkaran get\_training\_data () fonksiyonunu bildirerek başlayalım. İşlevin üç bağımsız değişkeni vardır: iyi huylu ikili programların örneklerini içeren bir dizine giden bir yol (benign\_path), kötü niyetli ikili programların örneklerini içeren bir dizine giden yol (kötü niyetli\_yol) ve özellik karması yapmak için kullanılan sklearn FeatureHasher sınıfının bir örneği ( hasher). Daha sonra, bize belirli bir dizinde meydana gelen dosyalar için mutlak dosya yollarının listesini veren yerel bir yardımcı işlev olan get\_training\_paths () oluştururuz. Sonraki iki satırda, bize kötü niyetli ve iyi huylu eğitim örnek dizinlerinde oluşan yolların listelerini almak için get\_training\_paths kullanıyoruz.

Son olarak, özelliklerimizi çıkarıyoruz ve etiket vektörümüzü oluşturuyoruz.. Etiket vektörünün her kötü niyetli yol için bir 1'e ve her iyi huylu yol için bir 0'a sahip olduğuna dikkat edin, öyle ki etiket vektöründeki indekslerdeki sayılar, X dizisindeki aynı indekslerdeki özellik vektörlerinin etiketine karşılık gelir. Bu, sklearn'ın özellik ve etiket verilerini beklediği biçimdir ve kitaplığa her özellik vektörünün etiketini söylememizi sağlar.

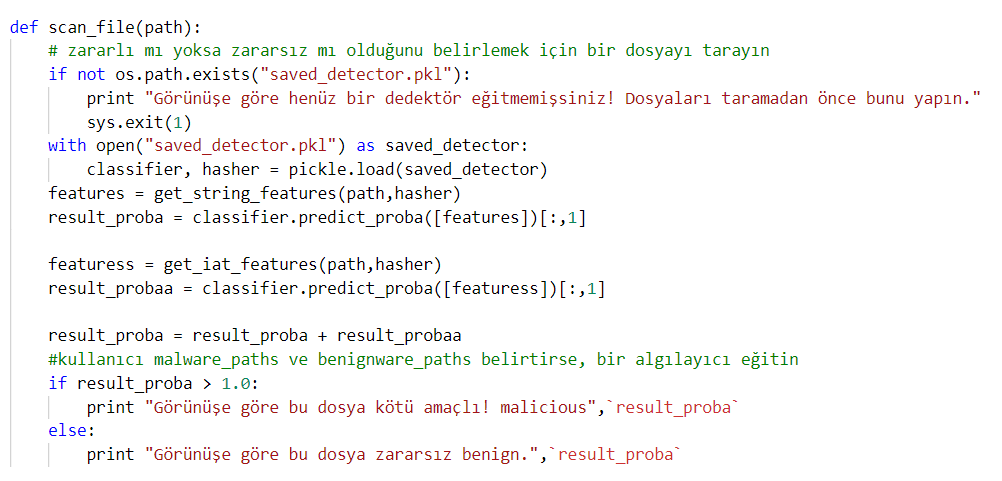
Artık özellikleri çıkarmayı bitirdiğimize ve özellik vektörümüzü X ve etiket vektörümüz y'yi oluşturduğumuza göre, sklearn'a özellik vektörlerini ve etiket vektörünü kullanarak dedektörümüzü eğitmesini söylemeye hazırız.



Bunu, üç bağımsız değişken alan train\_detector () fonksiyonunu kullanarak yapıyoruz: eğitim örneği özellik vektörleri (X), etiket vektörü (y) ve sklearn'ın özellik karma (hasher) örneği. Fonksiyon gövdesinde sklearn detektörü olan tree.RandomForestClassifier'ı başlatıyoruz. Ardından, onu eğitmek için dedektörün yerleştirme yöntemine X ve y'yi geçiririz ve ardından dedektörü ve hasher'ı ileride kullanmak üzere kaydetmek için Python pickle modülünü kullanırız.

\*Yeni ikilileri Çalıştırma:

Şimdi, yeni program ikili dosyalarında kötü amaçlı yazılımları tespit etmek için yeni eğittiğimiz kaydedilmiş algılayıcıyı nasıl kullanacağımızı gözden geçirelim.



Burada, bir dosyanın kötü niyetli mi yoksa zararsız mı olduğunu belirlemek için scan\_file () işlevini açıklıyoruz. Tek argümanı, tarayacağımız ikili dosyaya giden yoldur. Fonksiyonun ilk işi, kaydedilen detektörü ve hasher'ı kaydedildikleri pickle dosyasından yüklemektir. Ardından, tanımladığımız get\_string\_features ve get\_iat\_features işlevlerini kullanarak hedef dosyadan özellikleri çıkarıyoruz.

Son olarak, çıkarılan özellikler göz önüne alındığında, söz konusu dosyanın kötü amaçlı olup olmadığına karar vermek için algılayıcının tahmin yöntemini çağırıyoruz. Bunu, sınıflandırıcı örneğinin predikt\_proba yöntemini kullanarak ve döndürdüğü dizinin ikinci öğesini seçerek yaparız, bu da dosyanın kötü amaçlı olma olasılığına karşılık gelir. Bu olasılık 1.0 üzerindeyse dosyanın kötü amaçlı olduğunu söyleriz; aksi takdirde, kullanıcıya zararsız olduğunu söyleriz. Yanlış pozitifleri en aza indirmek için bu karar eşiğini çok daha yüksek bir değere değiştirebiliriz.

SON TOPLAM KODLARIMIZ ve UYGULAMA RESMİ:

