Die Labels aus den Annotations habe ich mit dem LabelEncoder von sklearn encoded.

Das nicht-numerische Feature „dns_query_name“ enthält als Werte Listen, die nur 1 String beinhalten. Daher müssen diese Strings erstmal extrahiert werden.

Für nicht-numerische Features, die als Werte Strings enthalten, habe ich einen eigenen Feature-Encoder „My_Feature_Encoder“ programmiert. Im Grunde genommen ist es ein LabelEncoder, der aber nach dem Fitting neuhinzugekommene Werte hinzufitten kann und auch mit fehlenden Werten bzw. nans umgehen kann. Dieser Encoder encodiert Strings auf numerische Werte.

Ich habe mich dafür entschieden, fehlende Werte bzw. nans mit der Zahl -1 zu ersetzen. Denn sowohl in Trainings- als auch die Test-Daten ist die kleinste vorkommende Zahl eine 0 gewesen. Diese Idee wird im ganzen Programm umgesetzt.

Danach werden sowohl die Trainings- als auch die Test-Daten preprocesst. Hier gilt es fehlende Werte mit -1 zu ergänzen und nicht-numerische Features in numerische Features zu verwandeln.

In den Trainings- und Test-Dataframes werden diese Features entfernt:

- 'da', 'sa': Können nur 1 Wert annehmen. Daher sind sie irrelevant
- "rev_intervals_ccnt", "pld_ccnt", "rev_hdr_ccnt", "rev_pld_ccnt", "rev_ack_psh_rst_syn_fin_cnt", "ack_psh_rst_syn_fin_cnt", "intervals_ccnt", "hdr_ccnt": Diese Histogramme haben nach dem RandomForest-Fitting bei Ausgabe der Importances so gut wie gar keine Wichtigkeit aufgewiesen. Außerdem ist es sehr aufwendig diese Histogramme zu encodieren. Denn jedes dieser Histogramme müsste in 5 bis 16 Features aufgeteilt werden. Das verbraucht bei 8 Histogrammen sehr viel Speicher und Leistung, obwohl sie so oder so unwichtig scheinen.
- 'dns_query_name_len', 'dns_answer_ttl', 'dns_query_class', 'dns_answer_cnt', 'dns_query_type', "dns_query_cnt", 'http_uri': Aus logischer Sicht für mich unwichtig für die Klassifizierung.

Alle TLS-Arrays sind nicht-numerische Features. Die Arrays bestehen aus numerischen Werten und werden zu Mittelwerten transformiert. Dabei werden alle numerischen Werte addiert und durch die zugehörigen Counting-Features dividiert. Denn diese TLS-Arrays haben keine gleiche Länge. Um für jedes Feature ein Array auf die maximale Länge zu füllen, müsste man manche Feature in 80 bis 192 Features aufteilen. Das würde sehr viel Speicher und Leistung verbrauchen.

Die nicht-numerischen Features 'dns_query_name', 'http_content_type', 'http_host' haben als Werte Strings. Diese werden mit den jeweiligen My_Feature_Encodern encodiert.

Anschließend haben wir noch das nicht-numerische Feature „dns_answer_ip“. Es enthält Listen ohne fixe Länge. Diese Listen enthalten IP-Adressen. Wie bei den TLS-Arrays würde es sich nicht lohnen das Feature auf die maximale Länge x einer Liste in x Features aufzuteilen. Stattdessen berechnet das Programm die gemeinsame Netzwerkadresse als Dezimalzahl und speichert sie in „dns_answer_common_network_ip“.

Sowohl das Trainings- als auch das Test-Dataframe müssen für das Fitting und das Predicten die gleiche Feature-Reihenfolge haben. Daher erhält das Test-Dataframe die Feature-Reihenfolge des Trainings-Dataframes.

Ich habe mich für einen RandomForest als Classifier entschieden. Als Anzahl der Estimators habe ich 100 gewählt, weil es sich bei meinen Tests als besten Wert herausgestellt hat.