Zadania na 3.0 – sprawozdanie

Wszystkie zadania będą wykonywane przeze mnie przy użyciu Pythona i jego bibliotek. Komentarze w kodzie będę wykonywał w języku angielskim.

1. Stworzenie drzewa decyzyjnego na datasecie titanic (dostępny pod linkiem: <https://www.kaggle.com/competitions/titanic>)

Do wykonania tego zadania użyję biblioteki Scikit-Learn, do wczytania i manipulacji danymi będę używał Pandas.

Otwieram zestaw danych, do treningu i do testu, dane są w postaci plików CSV

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Interesuje mnie zestaw do treningu, z którego muszę wydzielić dane oraz target, którym będzie kolumna Survived.

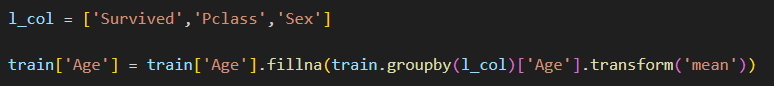
Zanim jednak to zrobie, warto byłoby uzupełnić brakujące dane wiekowe niektórych pasażerów. Wiek w tym przypadku pełni dosyć ważną rolę jeśli rozważamy przetrwanie pasażerów. Pustych danych jest całkiem sporo, co mogę łatwo sprawdzić sumując je:





Puste dane postanowiłem uzupełnić, grupując pasażerów kolumnami ['Survived','Pclass','Sex'] i wyciągając średnią wiekową z tych grup.

Uzupełniam dane przy użyciu .fillna():



Przed wrzuceniem danych do klasyfikatora, trzeba je jeszcze nieco wyczyścić oraz zamienić wartości słowne na liczbowe. Dodaje nową kolumnę FamilySize oraz IsAlone, rezygnując z niepotrzebnych.

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

Przy użyciu biblioteki Seaborn, można w fajny sposób zwizualizować korelacje między danymi:

Obraz zawierający zrzut ekranu, tekst, kwadrat, Wielobarwność

Opis wygenerowany automatycznie

Chciałbym teraz sprawdzić, jak głębokie drzewo będzie najbardziej optymalne w naszym przypadku. Do tego, użyję sobie techniki Cross Validation, która będzie dzielić nasz train set na części i obliczać celność dla każdej z nich.

Możliwe głębokości ustawiłem na od 1 do ilości kategorii w datasecie, ilość podziałów na 10:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, czarne, Czcionka

Opis wygenerowany automatycznie

Widzę, że najlepiej poszło na 5, także wykorzystam to jako max głębokość swojego drzewa.

Robię sobie podział na dane i target:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Tworzę drzewo, wrzucam do niego dane, robię test na testowych danych.

Wytrenowany model zapisuje sobie, dodatkowo zapisuje sobie png z moim drzewem w celu wizualizacji:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie

Opis wygenerowany automatycznie

Do stworzenia obrazu użyłem biblioteki Graphviz

Obraz drzewa zapisany jest w pliku tree\_image (za szeroki by tu wkleić).

Na koniec liczę sobie celność modelu, która wyniosła 86.08.

1. Prosta sieć dwuwarstwowa ucząca się XORA

XOR – czyli alternatywa rozłączna, poniższa tabela prezentuje działanie tej bramki logicznej:

Obraz zawierający tekst, zrzut ekranu, numer, Czcionka

Opis wygenerowany automatycznie

Model, który stworze będzie modelem dwuwarstwowym, wejście + 1 ukryta + wyjście.  
Wejście będzie posiadać 2 neurony – na 1 i 0, warstwa ukryta może być w tym przypadku prosta, na 3 lub 4 neurony. Wyjście na 1 neuron jest wystarczające, będzie to float od 0 do 1, który zostanie zaokrąglony do najbliższej liczby.

Do implementacji potrzebna mi będzie biblioteka PyTorch.

Z racji, że XOR jest łatwym problemem, w tym przypadku nie będzie konieczne robienie dużego datasetu:

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, Oprogramowanie multimedialne

Opis wygenerowany automatycznie

Tworzę sobie prosty model sieci dwuwarstwowej:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Teraz gdy mamy nasz model, zostało zinicjalizowanie go. Następnie musimy też wybrać funkcję straty i optymalizatora.

Jako f-cje straty użyje MSE Loss ( Mean Squared error Loss), która jest stosowania do zadań regresji, jako iż XOR możemy potraktować jako regresje binarną.

Co do optymalizatora użyje SGD (Stochastic Gradient Descent), który jest prostym i powszechnie używanym optymalizatorem.

Przy bardziej skomplikowanych problemach, warto poeksperymentować z doborem obu funkcji, jednak tutaj z racji prostoty problemu, nie będzie to potrzebne.

Przy problemach i przy tworzeniu sieci, często tworzy się blok w kodzie, dedykowany stworzeniu hiperparametrów, czyli parametrów modelu, które będą decydować o jego nauce. Są to parametry takie jak: ilość epok, learning rate(jak szybko model ma się uczyć), wielkość warstw modelu, ilość neuronów w warstwie, wielkość batchy czy foldów w przypadku Cross Validation itd…

Odpowiedni dobór tych parametrów stanowi wyzwanie przy tworzeniu bardziej zaawansowanych modeli, stosuje się róże techniki aby jak najlepiej dopasować je w celu osiągnięcia najwyższej celności modelu.

W kodzie będzie to wyglądać następująco:

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

Mamy wszystko czego potrzebujemy, teraz wytrenujemy nasz model przy użyciu prostej pętli.

1. Sieć konwolucyjna ucząca się MNIST