Paweł Lis
Maciej Pęśko
Magdalena Segieda

Dokumentacja projektu z przedmiotu PSZT

Temat: Stwórz sieć neuronową, która klasyfikuje transakcje bankową jako podejrzaną

1. Zastawienie kluczowych decyzji projektowych

- Perceptron wielowarstwowy,
- Opracowanie sieci neuronowej w bibliotece keras w celu zbadania czy opracowanie podejście ma szansę powodzenia.
- Opracowanie zbalansowanych danych (oryginalnie silnie niezrównoważone dane) po wcześniejszej ich eksploracji
- Implementacja sieci neuronowej wraz z algorytmem wstecznej propagacji gradientu (metoda gradientu prostego) w taki sposób aby można było dowolnie modyfikować parametry liczba warstw, liczba neuronów w warstwie, liczba epok, learning rate, parametr regularyzacji, batch size czy liczbę podziałów w walidacji k-fold.

2. Instrukcja dla użytkownika

- Projekt został umieszczony w Jupyterze w dwóch wersjach, które odpowiadają dwóm podejściom do rozwiązania problemu silnie niezrównoważonego zbioru danych.
- Aby program został uruchomiony poprawnie należy w folderze ‘data’ umieścić pliki ściągnięte z:
  https://github.com/maciej3031/bank_transactions_classifier/tree/master/data
  Są to zbiory treningowo-walidacyjny oraz testowy użyte to wytrenowania sieci.
- Wymagane biblioteki: numpy, pandas, matplotlib i tqdm. Zalecane jest uruchomienie wirtualnego środowiska Pythona i instalacja w/w pakietów lub skorzystanie z gotowego pakietu naukowego jakim jest Anaconda.
- Notebooki zostały tak sformułowane, że do uzyskanie wyników wystarczy je uruchomić. Podczas ich wykonywania sieć zostanie nauczona na zadanych parametrach a następnie w dalszej części widoczne będą wyniki.
- Całość można też pobrać z:
  https://github.com/maciej3031/bank_transactions_classifier

3. Opis struktury programu

3.1.Struktura sieci

Do stworzenia sieci neuronowej wybrano perceptron wielowarstwowy. Zaimplementowano dwie możliwe funkcje aktywacji: relu oraz sigmoid, możliwość wyboru parametru learning rate i liczby epok. Ponadto zaimplementowano możliwość trenowania różnej wielkości porcjami danych (parametr batch size) oraz regularyzację jako dodatkowy czynnik w funkcji kosztu równy iloczynowi współczynnika regularyzacji i wag. Na stworzonej sieci eksperymentalnie ustalono liczbę neuronów w warstwie ukrytej oraz wszystkie inne parametry, aby finalnie dostać jak najlepsze wyniki. Na wyjściu znajduje się jeden neuron z funkcją aktywacji sigmoid, który klasyfikuje transakcję jako złą bądź dobrą z określonym prawdopodobieństwem. Parametr batch_size ustalono jako rozmiar próbek w zbiorze uczącym.

3.2.Algorytm uczenia sieci

Częścią programu jest algorytm wstecznej propagacji gradientu. Jego celem jest minimalizacja funkcji kosztu, w tym przypadku została zastosowana funkcja MSE (błąd średnio-kwadratowy)

$$J(\theta) = \frac{1}{2}\left\|\bar{f}(x;\theta) - y\right\|^2$$

gdzie: $\bar{f}(x;\theta)$ oznacza wektor wyjść, x wektor wejść, θ wagi połączeń, a y to oczekiwany wynik.
Algorytm składa się z dwóch głównych kroków: global_forward_step i global_backwaard_step. W pierwszym następuje początkowe ustawienie wag oraz obliczenie wyjścia sieci. Natomiast w drugim kroku obliczany jest skumulowany gradient (metoda gradientu prostego), który dzielony jest przez liczbę transakcji w zbiorze treningowym, a potem przy pomocy jego uśrednionej wartości modyfikowane są wagi:

$$\theta_{k+1} = \theta_k - \alpha \times \nabla J(\theta)$$

Gdzie: $\theta_k$wagi początkowe, θ_{k+1}wagi uaktualnione, $\nabla J(\theta)$ to uśredniony gradient funkcji kosztu a α to parametr learning rate.

3.3.Podział danych

Na początku programu następuje podział danych na część testową oraz treningowo-walidacyjną. Pierwsza z nich zapisana została do pliku ‘train_dataset.data’ i nie jest używana podczas trenowania sieci. W niej znajdują się losowo wybrane 20% transakcji z silnie niezrównoważonego zbioru danych. Zbiór ten będzie użyty do końcowej ewaluacji dwóch wybranych podejść. Żeby pozbyć się negatywnych skutków niezrównoważonego zbioru danych treningowych i walidacyjnych, zastosowano dwa podejścia, które zostaną porównane we wnioskach:

- Podejście 1: Do utworzenia zbioru walidacyjno-treningowego wykorzystano wszystkie, znajdujące się w nim, złe transakcje oraz losowo wybrane dobre, których liczba jest równa liczbie tych pierwszych. Następnie zbiór ten podzielono na k = 5 części i wykonano walidację k-fold. Konsekwencją tego podejścia jest to, że sieć uczy się tylko na małej części dobrych transakcji.

- Podejście 2: Ze zbioru walidacyjno-treningowego oddzielamy od siebie dobre i złe transakcje, żeby potem podzielić każdy z tych zbiorów na k = 5 części i w każdym etapie k-fold wybierać po jednej do walidacji a pozostałe do treningu. Przed uczeniem się sieci oraz co każdą epokę do treningowego zbioru złych transakcji losowana jest taka liczba dobrych transakcji, ile jest złych, a następnie losowo mieszane są one ze sobą. Analogicznie w części walidacyjnej. W tym podejściu zwiększona jest liczba dobrych transakcji, na których uczy się sieć (zmieniamy je co epokę).

3.4.Prezentacja wyników

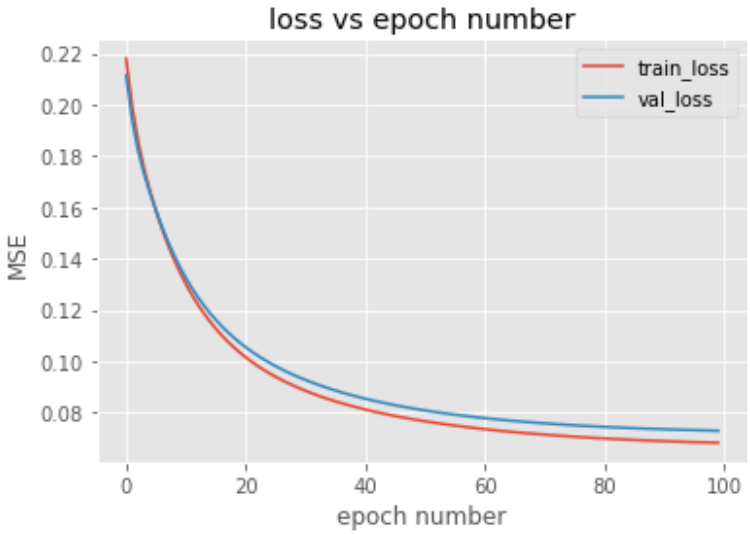
Do przedstawienia wyników uczenia się sieci neuronowej, które są we wnioskach, użyto krzywą uczenia się, ROC, confusion_matrix oraz parametry tj, wartość f. kosztu, recall, precision, f1-score i accuracy.

4. Wnioski

Ostateczna ewaluacja: Liczba epok = 100 dla podejścia 1 i 120 dla podejścia 2, learning_rate = 0.03, współczynnik regularyzacji = 0.1, liczba neuronów w warstwie ukrytej = 512

4.1. Wyniki pierwszego podejścia:

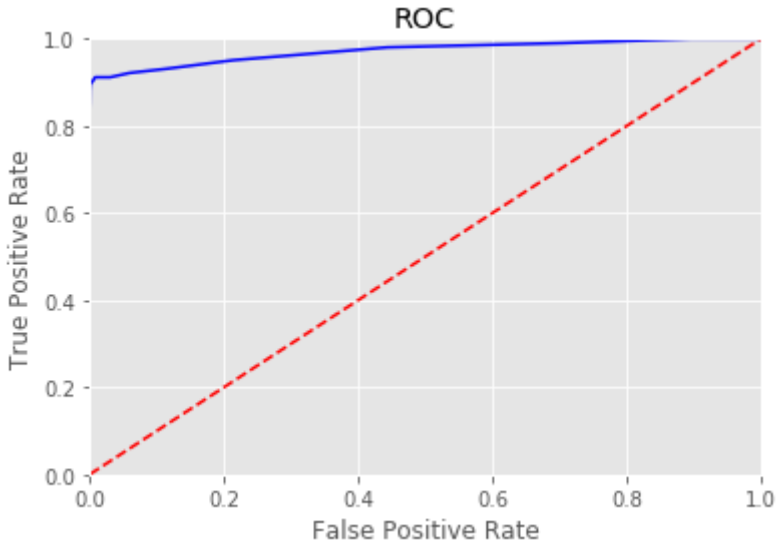
- Uśredniona krzywa uczenia się dla pięciu etapów k-fold



- Ewaluacja danych testowych, wartości f. kosztu, parametrów accuracy, precision, recall f-score oraz confusion matrix.

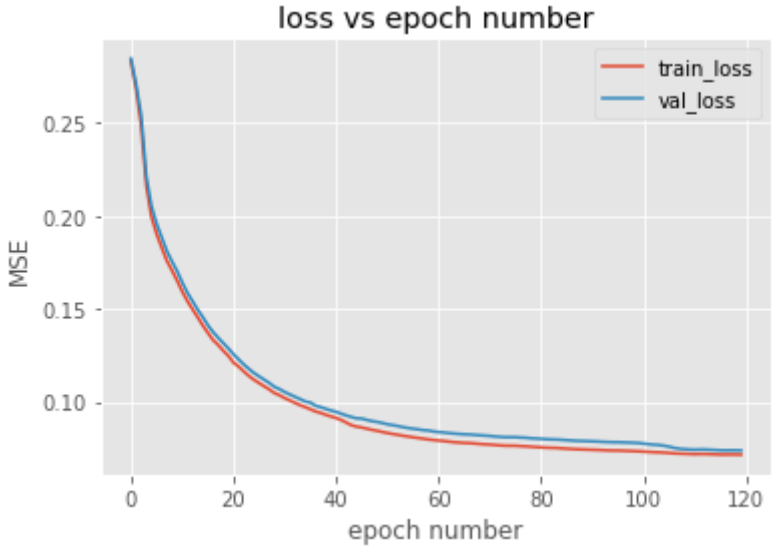
```
Loss:  0.0701891322517
Accuracy:  0.985025104455602

Precision: 0.09707724425887265
Recall: 0.9117647058823529
F-score: 0.17547169811320756


             actual 1  actual 0
predicted 1        93       865
predicted 0         9     55995
```

- Krzywa ROC



4.2.Wyniki drugiego podejścia:

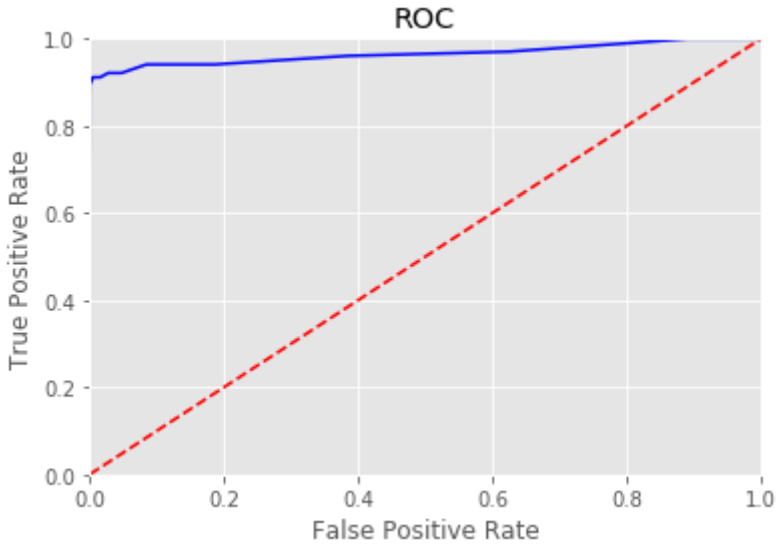
- Uśredniona krzywa uczenia się dla pięciu etapów k-fold



- Ewaluacja danych testowych, wartości f. kosztu, parametrów accuracy, precision, recall f-score oraz confusion matrix.

```
Loss:  0.0698601400928
Accuracy:  0.9791439907306625

Precision: 0.056220095693779906
Recall: 0.9215686274509803
F-score: 0.10597519729425027


                actual 1  actual 0
predicted 1           94      1578
predicted 0            8     55282
```

- Krzywa ROC



4.3.Porównanie

Na uśrednionych krzywych uczenia dobrze widać iż sieć jest zaimplementowana i nauczona poprawnie. Wartość funkcji kosztu stopniowo maleje, a uczenie przerwano w momencie kiedy krzywa wypłaszacza się i zaczyna się przeuczenie sieci co owocuje widoczną minimalnie mniejszą wartością funkcji kosztu w końcowych epokach dla zbioru treningowego. Ewaluacja danych testowych w obu przypadkach odbyła się na identycznych danych. Widać, że w podejściu pierwszym dane wynikowe są minimalnie lepsze niż w przypadku podejścia drugiego jeżeli chodzi o m.in. dokładność (accuracy) czy precyzję (precision). Wartość czułości (recall) jest podobna w obu podejściach. Jest to jeden z ważniejszych parametrów bo w przypadku problemu wyszukiwania zdefraudowanych transakcji najważniejsze jest wykrycie jak największej ich ilości. Czułość (precision) w tym wypadku odgrywa mniejszą rolę. Minimalne lepsze wyniki dla sieci z pierwszego podejścia prawdopodobnie wynikają z nauki na zdecydowanie mniejszej liczbie dobrych transakcji niż z drugiego podejścia. Prawdopodobnie wybrane tam dane, choć losowo, były dobrą reprezentacją całej klasy. Natomiast gorsze wyniki w drugim podejściu mogą być skutkiem trudności dopasowania się sieci do ciągle zmienianych dobrych transakcji.