Mateusz Przybył 254228
Jakub Ramisz 254230

Analiza danych złożonych z detekcją wyjątków

Zadanie 1

1. Charakterystyka danych

Dane wybrane do zadania dotyczą tematyki medycznej, a dokładniej faz ruchu człowieka. Wykorzystany zbiór danych to „Inertia Sensors for Human Activity Recognition" (link) pochodzący z serwisu Kaggle. Zbiór wykorzystywany w wdrożeniu Systemu HAR (Systemu Rozpoznawania Aktywności Człowieka), którego celem jest identyfikacja czynności, którą użytkownik będzie wykonywał, wyłącznie na podstawie zmian w ruchu ciała użytkownika podczas wykonywania określony aktywności.

Zbiór danych zawiera dwa pliki formatu *csv*, z wartościami gotowymi do przekazania do modeli. Różnica pomiędzy owymi plikami polega na tym, że jeden plik jest już wstępnie przetworzony (usunięcie zduplikowanych wartości, wartości „NaN"). Na potrzeby zadania zdecydowaliśmy posłużyć się zbiorem wstępnie przetworzony liczącym 3778 rekordów, zawierającym informacje na temat zmian w ruchu ciała i aktywności podczas której te zmiany wystąpiły.

2. Wybrane klasyfikatory zespołowe

W zadaniu wybraliśmy trzy klasyfikatory grupowe w celu ich porównania, a są to: K-Nearest Neighbours, Klasyfikator Bayesowski, Maszyna Wektorów Nośnych SVM. Implementacja zadania została wykonana w języku Python, a do wykorzystania algorytmów wybranych przez nas posłużyliśmy się biblioteką *sklearn*.

I. K-Nearest Neighbours

Jeden z podstawowych algorytmów uczenia maszynowego, stosowany do klasyfikacji i regresji. KNN jest przydatny do problemów, gdzie kluczowe jest podobieństwo między punktami, lecz jego skuteczność jest uzależniona od jakości danych - gdy dane mają dużo szumu lub wysoką wymiarowość jego skuteczność spada.

Kroki działania KNN:

I. Przygotowanie danych i wczytanie danych – algorytm wymaga wcześniejszego oznaczenia danych treningowych, które posłużą do przewidywania etykiet nowych danych. Następnie dane są wczytywane i przedstawiane w postaci punktów

II. Obliczenie odległości – w celu sklasyfikowania nowego punktu, algorytm oblicza odległość (w większości przypadków euklidesową) między nowym punktem, a wszystkimi punktami w zbiorze treningowym.

III. Wybór sąsiadów – algorytm dobiera K najbliższych sąsiadów nowego punktu na podstawie odległości z punktu III.

IV. Klasyfikacja – KNN sprawdza etykiety K najbliższych sąsiadów i przypisuje nowemu punktowi etykietę, którą posiada większość z nich (tzw. głosowanie większościowe).

II. Klasyfikator Bayesowski

Klasyfikator bayesowski to algorytm statystyczny, który bazuje na twierdzeniu Bayesa i jest stosowany głównie do klasyfikacji. Jest prosty, szybki i skuteczny, szczególnie dla problemów takich jak klasyfikacja tekstu (np. filtrowanie spamu), analiza opinii, czy segmentacja klientów.

Naive Bayes oblicza prawdopodobieństwo, że dany punkt *x* należy do klasy *C*, wykorzystując twierdzenia Bayes'a:

$$P(C|x) = \frac{P(x|C) \cdot P(C)}{P(x)}$$

gdzie:

- *P(C|x)* – prawdopodobieństwo, że punkt *x* należy do klasy *C* (posteriori),
- *P(x|C)* – prawdopodobieństwo wystąpienia punktu x pod warunkiem, że należy do klasy C (prawdopodobieństwo wiarygodności),
- *P(C)* – A priori prawdopodobieństwo klasy C (częstość występowania klasy C w zbiorze treningowym),
- *P (x)* – całkowite prawdopodobieństwo punktu x.

Kroki działania:

I. Wytrenowanie modelu na podstawie zbioru treningowego - Na podstawie danych treningowych, klasyfikator wylicza *P(C)*, czyli a priori prawdopodobieństwa dla każdej klasy i również *P(x|C),*

czyli prawdopodobieństwa pojawienia się konkretnych cech (np. słów w klasyfikacji tekstu) dla każdej klasy.,

II. Założenie niezależności cech – Klasyfikator Naive Bayes zakłada, że wszystkie cechy są od siebie statystycznie niezależne w obrębie każdej klasy, co znacznie upraszcza obliczenia i dzięki temu, zamiast obliczać prawdopodobieństwo całkowite, obliczany jest iloczyn prawdopodobieństw dla poszczególnych cech,
III. Obliczenie prawdopodobieństwa dla każdej klasy – W celu przewidzenia klasy dla nowego punktu, klasyfikator oblicza prawdopodobieństwa $P(C|x)$ dla wszystkich klas C, używając wzoru Bayesa i wybierając klasę, dla której $P(C|x)$ jest największe.

III. SVM

Maszyna Wektorów Nośnych (z ang. Support Vector Machine) to algorytm nadzorowanego uczenia maszynowego używany głównie do klasyfikacji i regresji. W klasyfikacji jego celem jest znalezienie optymalnej granicy (hiperpłaszczyzny), która najlepiej oddziela próbki z różnych klas.

Kroki działania:

I. Przygotowanie danych – Model przyjmuje dane treningowe, w których każda próbka ma przypisaną klasę. Każdy przykład reprezentowanych jest poprzez wektor cech,
II. Wyznaczenie hiperpłaszczyzny i marginesu – Na podstawie wybranego typu jądra (kernel) i parametru C model jest trenowany, znajdując taką hiperpłaszczyznę, która maksymalizuje margines między najbliższymi punktami obu klas,
III. Klasyfikacja nowych danych – Nowe punkty klasyfikowane są w zależności od tego, po której stronie hiperpłaszczyzny się znajdują.

3. Wybrana sieć neuronowa

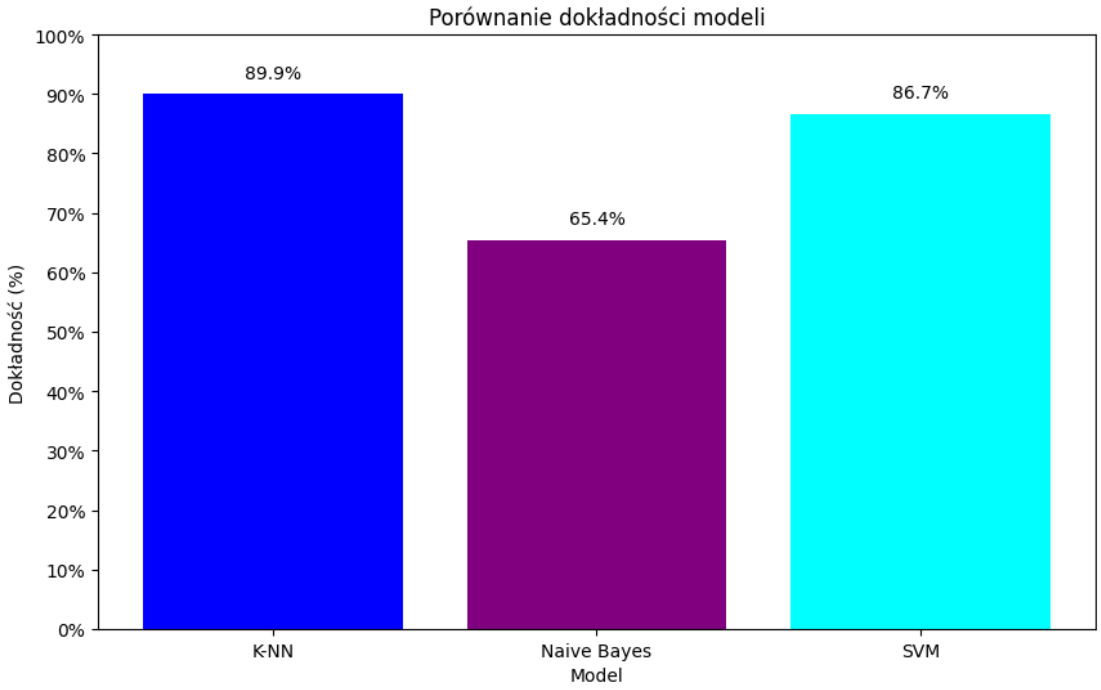
Do implementacji wybranej sieci neuronowej posłużyliśmy się bibliotekami *keras* i *tensorflow* dostępnymi w Python. Z pomocą wspomnianych bibliotek zaimplementowaliśmy prostą sieć neuronową (typu MLP) dokonującą klasyfikacji danych.

Sieć jest zdefiniowana jako model Sequential, co oznacza, że warstwy są ułożone sekwencyjnie, warstwa po warstwie. Każda warstwa bierze dane z poprzedniej i przekazuje je dalej. Model posiada dwie warstwy ukryte – pierwsza ma 64 neurony i wykorzystuje funkcję aktywacji ReLU, a druga zawiera 32 neurony i również korzysta z funkcji aktywacji ReLU. Jeśli chodzi o kompilacje modelu to korzysta on z optymalizatora Adam oraz funkcji straty (ang. categorical crossentropy) – powszechnie stosowana funkcja w klasyfikacji wieloklasowej. Model trenowanych jest na danych przez 30 epok z wielkością partii 32 oraz przy podziale danych na walidację z wykorzystaniem 20% próbek.

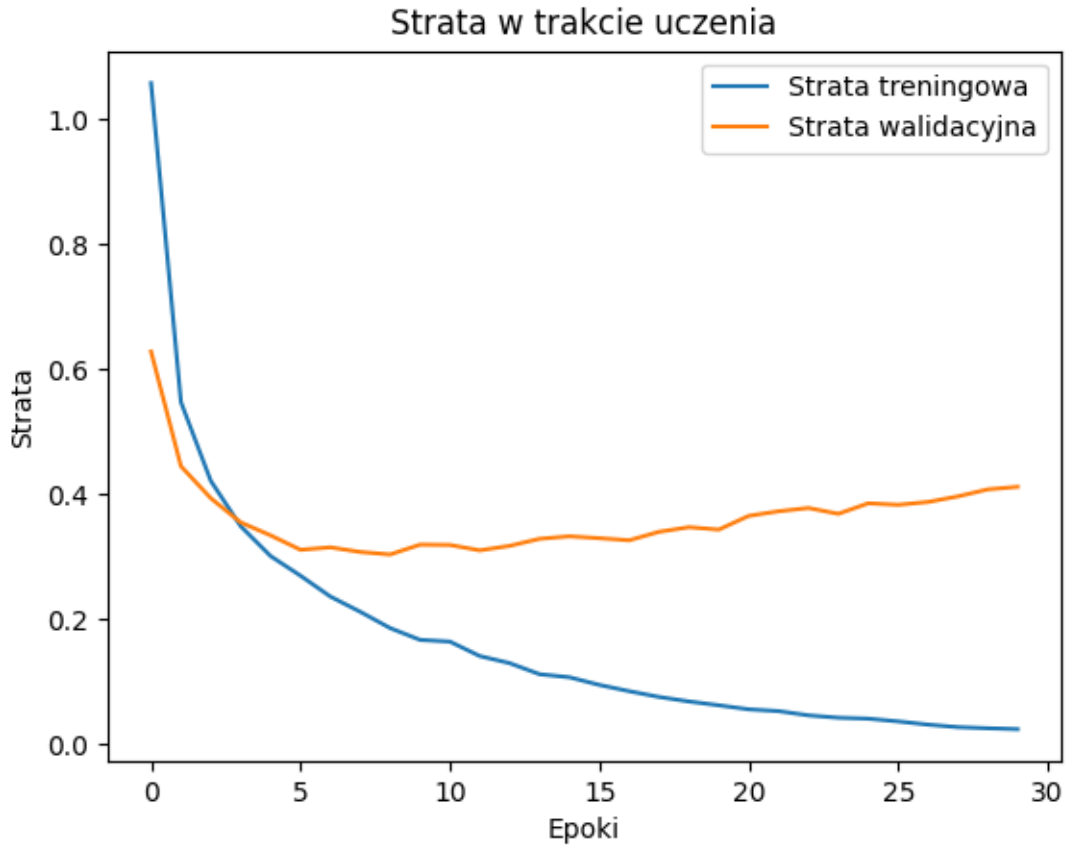
4. **Wyniki**

```
K-NN Accuracy: 0.8994708994708994
K-NN Classification Report:
              precision    recall  f1-score   support

     Cycling       0.92      0.94      0.93       353
    Football       0.90      0.94      0.92       359
     Jogging       0.80      0.29      0.42        14
    JumpRope       0.83      0.63      0.72        90
     Pushups       0.80      0.53      0.64        30
    Swimming       0.90      0.96      0.93       288

    accuracy                           0.90      1134
   macro avg       0.86      0.71      0.76      1134
weighted avg       0.90      0.90      0.89      1134

Naive Bayes Accuracy: 0.654320987654321
Naive Bayes Classification Report:
              precision    recall  f1-score   support

     Cycling       0.70      0.82      0.75       353
    Football       0.91      0.48      0.63       359
     Jogging       0.21      0.93      0.34        14
    JumpRope       0.65      0.57      0.61        90
     Pushups       0.13      0.63      0.21        30
    Swimming       0.81      0.69      0.74       288

    accuracy                           0.65      1134
   macro avg       0.57      0.69      0.55      1134
weighted avg       0.77      0.65      0.68      1134

SVM Accuracy: 0.8668430335097002
SVM Classification Report:
              precision    recall  f1-score   support

     Cycling       0.89      0.89      0.89       353
    Football       0.89      0.92      0.91       359
     Jogging       0.53      0.57      0.55        14
    JumpRope       0.71      0.61      0.66        90
     Pushups       0.57      0.40      0.47        30
    Swimming       0.88      0.92      0.90       288

    accuracy                           0.87      1134
   macro avg       0.75      0.72      0.73      1134
weighted avg       0.86      0.87      0.86      1134
```

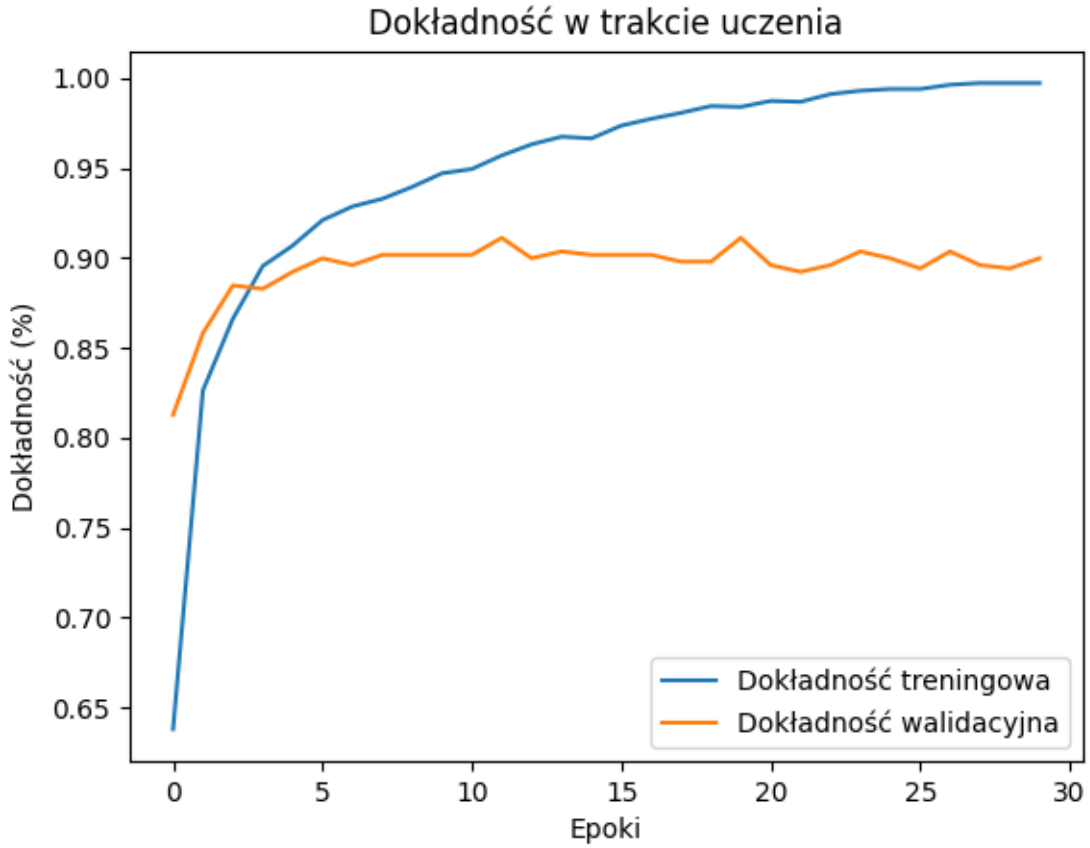
Rys. 1 – raport klasyfikacji dla algorytmów KNN (k=5), Naive Bayes, SVM (kernel=linear).



Rys. 2 – porównanie dokładności dokonanej klasyfikacji poprzez modele (z Rys. 1).



Rys. 3 – wykres strat w trakcie uczenia sieci neuronowej.



Rys. 4 – wykres dokładności klasyfikacji w tracie uczenia sieci neuronowej.

5. Wnioski

Algorytmy klasyfikacji zespołowej

Ze wszystkich algorytmów najlepiej poradził sobie KNN osiągając dokładność 90%, tuż za nim uplasował się SVM z dokładnością na poziomie 87%, a najniższą dokładnością charakteryzuje się Naive Bayes (65%) co wskazuje, że algorytm ten ma trudności z poprawnym klasyfikowaniem danych w tym zestawie, być może ze względu na założenie o niezależności cech.

K-NN i SVM wypadły najlepiej w klasach o dużych liczbach próbek: "Cycling", "Football", oraz "Swimming". Dla tych klas wskaźniki precyzji, czułości i f1-score były najwyższe, co sugeruje, że oba modele dobrze klasyfikują dominujące klasy. Aczkolwiek Naive Bayes wykazuje zmienną skuteczność w zależności od klasy. Uzyskał wysoką precyzję dla klasy "Football" (0.91) i dość dobre wyniki dla klasy "Cycling", ale jednocześnie ma niskie wyniki dla "Pushups" i "Jogging", co obniża ogólną dokładność i f1-score. Warto również zauważyć, że dla rzadziej reprezentowanych klas ("Jogging" i "Pushups") wszystkie algorytmy wykazują niższą skuteczność, co może być wynikiem małej liczby przykładów tych klas w zbiorze treningowym.

Możemy również zauważyć, iż K-NN osiągnął wyraźnie lepszy wynik w "Jogging" i "Pushups" niż SVM, co sugeruje, że metoda oparta na bliskości sąsiadów lepiej dopasowuje się do rzadziej reprezentowanych klas niż metoda SVM, która optymalizuje granicę między klasami.

Dodatkowo K-NN osiąga najwyższe wyniki w miarach makro (macro avg) i ważonych (weighted avg), co wskazuje, że lepiej radzi sobie z nierównowagą klas i ma dobrą średnią skuteczność dla wszystkich klas. Z kolei Naive Bayes wykazuje niższe wartości makro dla precyzji i f1-score (około 0.57 i 0.55), co wskazuje na większe problemy w skuteczności dla mniej reprezentowanych klas, a SVM osiąga wartości makro na poziomie średnim między K-NN a Naive Bayes, co oznacza, że model ten radzi sobie ogólnie lepiej niż Naive Bayes, ale nie jest tak efektywny jak K-NN dla całego zestawu danych.

Podsumowując można stwierdzić, iż ze wszystkich klasyfikatorów zespołowych najlepiej poradził sobie KNN i jest on najlepszym algorytmem klasyfikującym w zadaniu, zwłaszcza w kontekście równowagi między precyzją a czułością dla dominujących klas oraz poprawnej klasyfikacji rzadziej występujących klas.

Sieć neuronowa

Na wykresie dokładności widzimy, że dokładność na zbiorze treningowym systematycznie wzrasta wraz z liczbą epok, osiągając poziom bliski 100%, a dokładność walidacyjna początkowo rośnie, ale stabilizuje się na poziomie około 90%, z pewnymi drobnymi zmianami. Od około 10-tej epoki przestaje rosnąć znacząco, co może wskazywać na ustabilizowanie się osiągów modelu na zbiorze walidacyjnym.

W przypadku wykresu starty widać, że zarówno strata treningowa, jak i walidacyjna gwałtownie spadają w pierwszych kilku epokach, co oznacza, że model na początku szybko się uczy. Strata treningowa nadal maleje aż do końca procesu uczenia, co sugeruje, że model dobrze dopasowuje się do danych treningowych. Strata walidacyjna osiąga punkt minimalny wcześniej, około 10-15 epoki, a następnie zaczyna wzrastać, co jest typowym sygnałem przeuczenia (tzw. overfittingu). Wzrost straty walidacyjnej przy jednoczesnym spadku straty treningowej wskazuje, że model zaczyna lepiej dopasowywać się do specyfiki danych treningowych, ale traci ogólną zdolność do generalizacji na nowych danych.

W celu rozwiązania problemu przeuczenia można wykorzystać wartwy „*Dropout*", czyli warstwy, która mogłaby pomóc w uogólnieniu modelu poprzez losowe wyłączanie części neuronów w czasie treningu, co by zapobiegło nadmiernemu dopasowaniu. Można również zastanowić się nad wykorzystaniem tzw. *Early Stoppingu,* czyli mechanizmu wcześniejszego zatrzymania, który kończy proces treningowy, gdy strata walidacyjna przestaje się poprawiać.

Podsumowując opracowana sieć neuronowa wykazuje dobre dopasowanie do danych treningowych, z zadowalającą dokładnością na poziomie porównywalnym, a nawet lepszym niż KNN ale rosnąca strata walidacyjna oraz różnica między dokładnością treningową i walidacyjną sugerują przeuczenie.