

Projekt IO.1: Klasyfikacja ceny pojazdów.

Mateusz Kurowski

Maj 2024

Spis treści

1	Wstęp	2
2	Baza danych	3
3	Preprocessing	4
3.1	Usuwanie zbędnych danych	4
3.2	Usuwanie nieprawidłowych lub odstających danych	4
3.3	Uzupełnianie brakujących danych	5
3.4	Ujednolicenie typów	5
3.5	Własnoręczna klasyfikacja	6
3.5.1	Dziesięcioklasowa	6
3.5.2	Dwuklasowa	6
3.6	Normalizacja	8
3.7	PCA	8
4	Proste klasyfikatory	12
4.1	K-najbliższych sąsiadów (K-Nearest Neighbors - KNN)	12
4.1.1	Wyniki	12
4.2	Wykresy	12
4.3	Naiwny Bayes (Naive Bayes)	17
4.3.1	GaussianNB	17
4.3.2	CategoricalNB	19
4.3.3	MultinomialNB	21
4.3.4	BernoulliNB	23
4.3.5	ComplementNB	25
5	Reguły asocjacyjne	27
6	MLP	27
6.1	Wyniki	29
6.1.1	Kamienie milowe	31
7	Bibliografia	35

1 Wstęp

Samochodami pasjonuję się od kiedy pamiętam, moim pierwszym pomysłem na realizację projektu było rozpoznawanie samochodów ze zdjęć, ale stwierdziłem, że ten pomysł może zrealizuję jako drugi projekt, więc naturalnie jako temat projektu nr.1 przyjąłem klasyfikację cenową pojazdów samochodowych, gdyż wycena auta to naturalny problem występujący przy sprzedaży.

Więc, jak poradziły sobie z tym problemem podstawowe klasyfikatory oraz sieć?

2 Baza danych

Pierwotna baza danych składa się z 426 880 rekordów oraz 26 kolumn:

- id
- url
- region
- region_url
- price
- year
- manufacturer
- model
- condition
- cylinders
- fuel
- odometer
- title_status
- transmission
- VIN
- drive
- size
- type
- paint_color
- image_url
- description
- county
- state
- lat
- long
- posting_date

3 Preprocessing

3.1 Usuwanie zbędnych danych

Postanowiłem usunąć ze zbioru danych kolumny, które nie miały większego znaczenia dla predykcji lub dane nieprzydatne, jak description, które raczej się nie powtarza, jego istnienie nie ma sensu w datasetcie bez tokenizacji, której jeszcze nie umiem. Lista usuniętych kolumn:

- id
- url
- region_url
- VIN
- image_url
- description
- state
- lat
- long
- posting_date
- title_status
- region

3.2 Usuwanie nieprawidłowych lub odstających danych

Dla poprawienia efektywności postanowiłem również usunąć rekordy z nieprawidłowymi danymi gdzie:

- Cena jest równa 0 lub większa niż 5 milionów USD
- Rok produkcji jest większy niż 2024 lub mniejszy niż 1900

W wyniku preprocessingu liczba rekordów zmniejszyła się o około 10%, do 382558 rekordów

3.3 Uzupełnianie brakujących danych

Ze względu na to, iż w zdecydowanej większości rekordów w co najmniej jednej kolumnie znajdowało się "N/A" postanowiłem uzupełnić dane za pomocą imputacji:

- Braki w kolumnach numerycznych uzupełnić medianą z kolumny
- Braki w kolumnach kategoryalnych uzupełnić najczęściej występującą wartością

Kod:

```
def fill_na(df: DataFrame) -> DataFrame:
    new_df = df.copy()
    cat_cols, num_cols = return_categorical_and_numerical_cols(df)

    numerical_imputer = SimpleImputer(strategy="median")
    categorical_imputer = SimpleImputer(strategy="most_frequent")

    # Uzupełnij brakujące wartości w danych numerycznych medianą
    new_df[num_cols] = numerical_imputer.fit_transform(df[num_cols])

    # Uzupełnij brakujące wartości w danych kategorycznych
    # najczęściej występującą wartością
    new_df[cat_cols] = categorical_imputer.fit_transform(df[cat_cols])
    return new_df
```

3.4 Ujednolicenie typów

Następnym moim krokiem było ujednolicenie typów kolumn typu numerycznego i sprowadzenie ich wszystkich typów do typu int.

```
def fix_types(df: DataFrame) -> DataFrame:
    new_df = df.copy()
    new_df["year"] = new_df["year"].astype(int)
    new_df["odometer"] = new_df["odometer"].astype(int)
    new_df["price"] = new_df["price"].astype(int)
    return new_df
```

3.5 Własnoręczna klasyfikacja

3.5.1 Dziesięcioklasowa

Ponadto, by uniknąć korzystania z modeli regresyjnych, które nie były używane na zajęciach, postanowiłem własnoręcznie sklasyfikować wartości kolumny "prices" przez wygenerowanie 10 mniej więcej równych (wg. kwartyli) przedziałów cenowych, do których zostały przydzielone poszczególne rekordy.

```
def get_intervals(df: DataFrame) -> list[tuple[int, int]]:
    X = df["price"] # Column of data
    num_intervals = 10 # Number of intervals
    intervals = qcut(X, q=num_intervals)
    my_intervals = []
    # Retrieving the right side of each interval
    for interval in intervals.unique():
        my_intervals.append((interval.left, interval.right))
    my_intervals.sort() # Sorting the intervals
    return my_intervals

def get_class(price: int, intervals: list[tuple[int, int]]) -> str:
    for left, right in intervals:
        if price < right:
            left_k = round(left / 1000, 1)
            right_k = round(right / 1000, 1)
            return f"({left_k}k, {right_k}k)"
    # If price is greater than all intervals, return the last interval
    last = round(intervals[-1][1] / 1000, 1)
    return f"({last}k+)"

def set_classes(df: DataFrame) -> DataFrame:
    new_df = df.copy()
    intervals = get_intervals(new_df) # Get intervals from new_df
    new_df["price"] = new_df["price"].apply(get_class, intervals=intervals)
    return new_df
```

3.5.2 Dwuklasowa

Dla celów eksperymentalnych, postanowiłem również spróbować klasyfikacji na zbiorze, gdzie przewidywana jest cena w zakresie: "cheap" lub "expensive".

```
def set_easy_classes(df: DataFrame) -> DataFrame:
    median = df["price"].median() # Calculate the median price
    # Use the apply method to generate labels based on the median price
    df["price"] = df["price"].apply(lambda x: "expensive" if x > median else "cheap")
```

```
return df
```

3.6 Normalizacja

Wszystkie kolumny tekstowe oprócz prices, postanowiłem znormalizować za pomocą LabelEncoder'a. Natomiast wszystkie kolumny numeryczne zostały przeze mnie przeskalowane MinMaxScaler'em.

```
def normalize(df: DataFrame) -> DataFrame:
    cat_cols, num_cols = return_categorical_and_numerical_cols(df)
    price_index = np.where(cat_cols == "price")
    cat_cols = cat_cols.delete(price_index)
    # Wykonaj kodowanie kategoriycznych danych za pomoc LabelEncoder
    label_encoder = LabelEncoder()
    for col in cat_cols:
        df[col] = label_encoder.fit_transform(df[col])

    # Wykonaj skalowanie danych numerycznych za pomoc MinMaxScaler
    scaler = MinMaxScaler()
    df[num_cols] = scaler.fit_transform(df[num_cols])

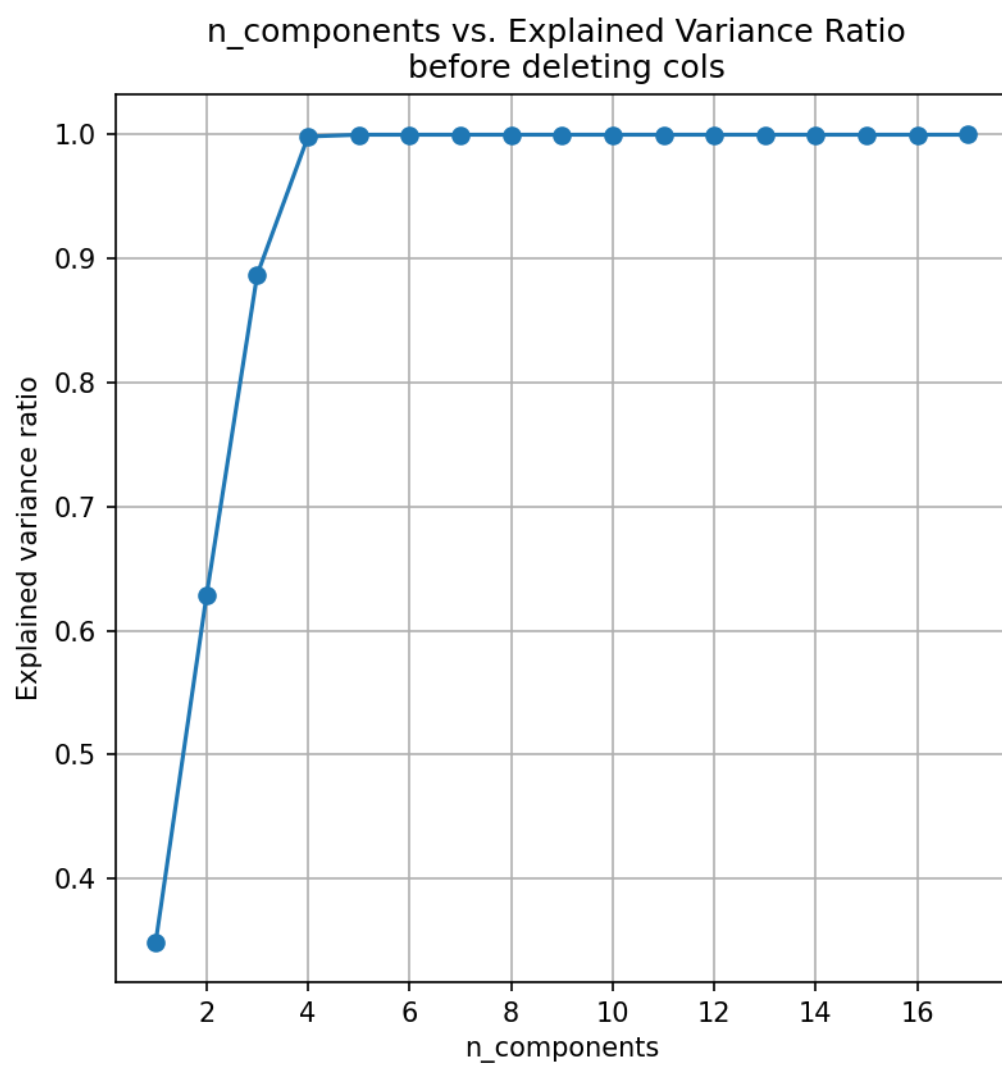
    return df
```

3.7 PCA

Postanowiłem przeprowadzić analizę PCA dla danych przed usunięciem oraz po usunięciu kolumn, przy zachowaniu wszystkich kroków preprossingu oraz normalizacji.

Wyniki analizy przed usunięciem kolumn:

Jak widać na powyższym wykresie przynajmniej 2 komponenty można usunąć, ponieważ nie przynoszą większego sensu dla zróżnicowania danych.



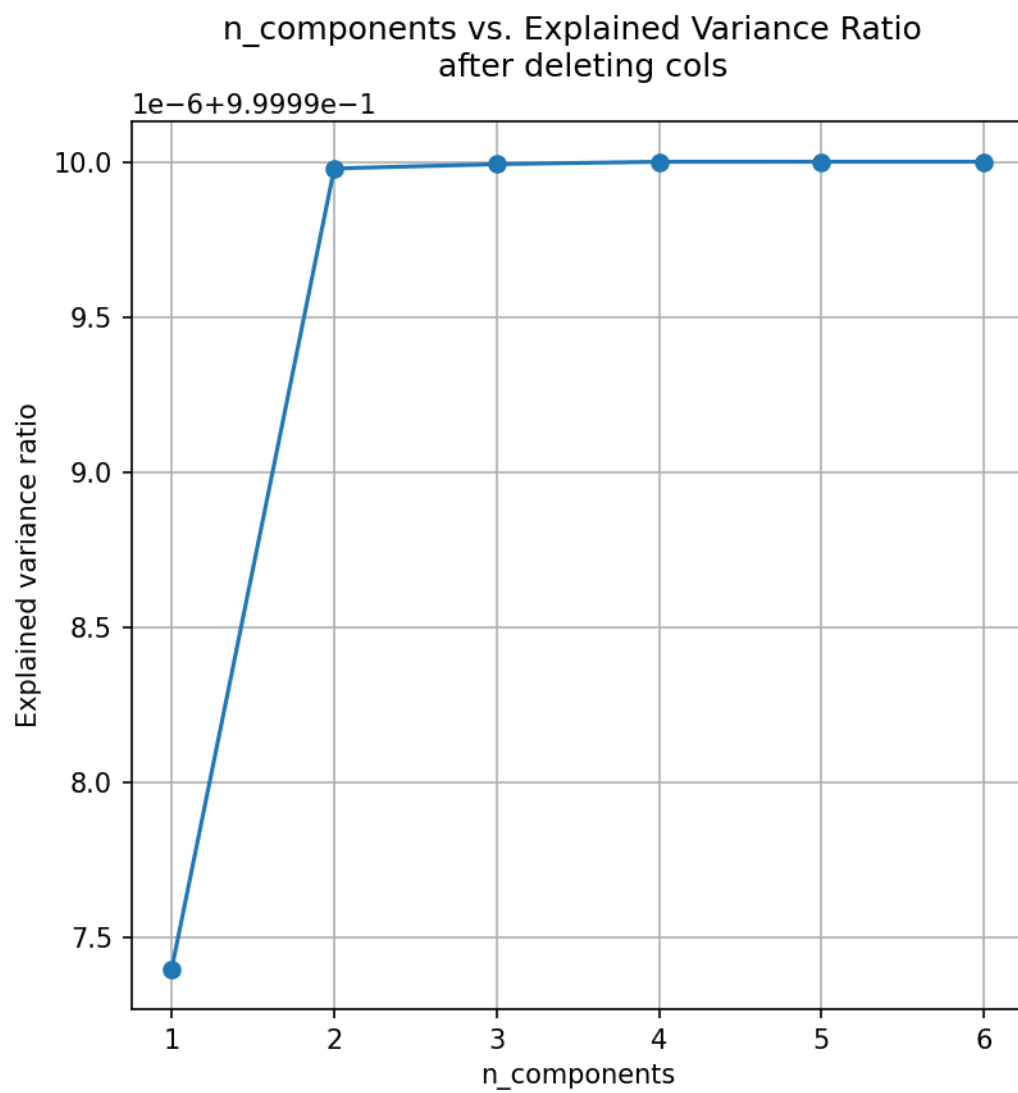
Rysunek 1: PCA przed

Wyniki analizy po usunięciu kolumn: Wyniki wskazują na to, iż usunięcie,

F1	F2	F3	F5	F6
0.9999973975525233	0.9999999781682694	0.9999999920745544	0.9999999998751801	1.0

którejkolwiek z kolumn nie przyniosłoby większych korzyści, a oznaczałoby dużą stratę danych.

Po przeprowadzeniu wszystkich elementów preprocessingu, wyniki analizy PCA dla wszystkich pozostałych (6) kolumn prezentują się następująco:



Rysunek 2: PCA po

4 Proste klasyfikatory

Wszystkie klasyfikatory były wielokrotnie przeze mnie testowane na zbiorze testowym różnej wielkości przy podziale 80% - zbiór treningowy oraz 20% - zbiór testowy

4.1 K-najbliższych sąsiadów (K-Nearest Neighbors - KNN)

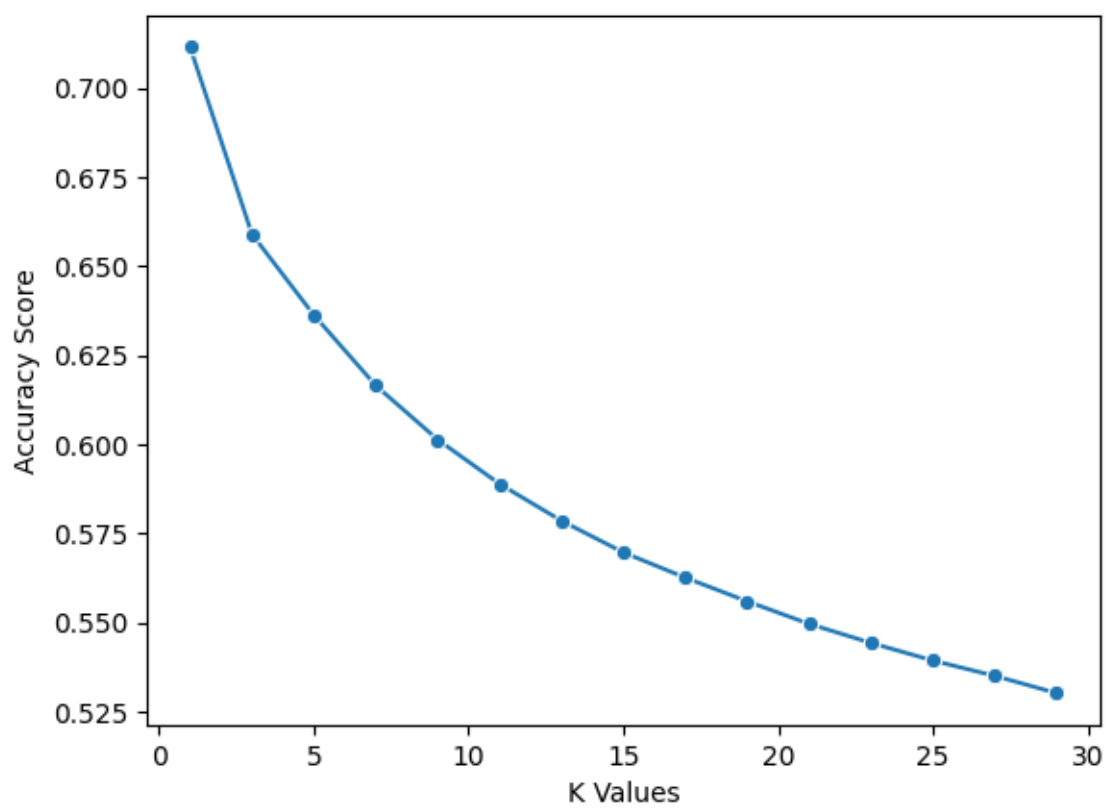
Klasyfikator KNN postanowiłem przetestować dla k równego każdej liczbie nieparzystej od 1 do 29.

4.1.1 Wyniki

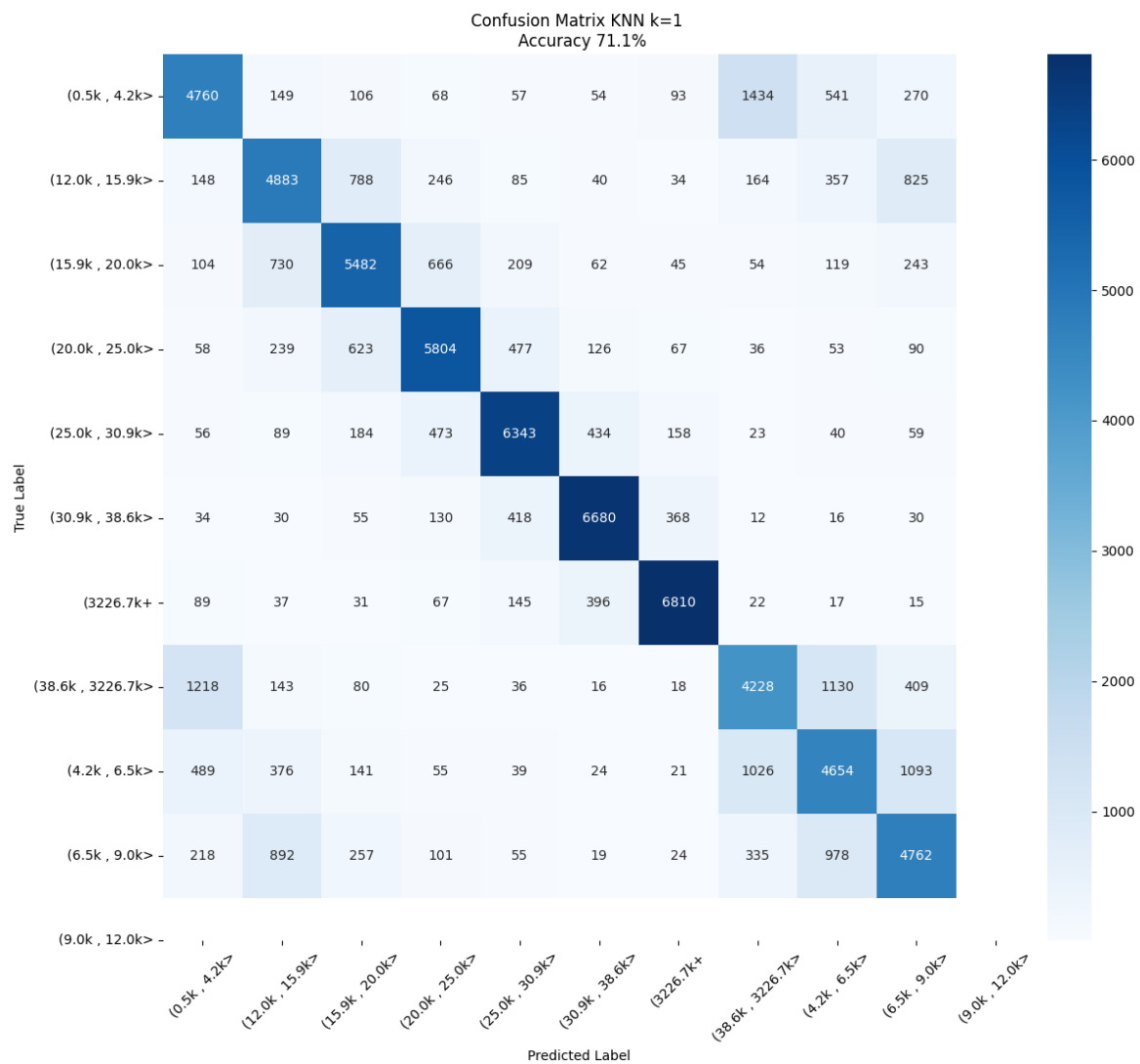
k	Accuracy
1	71.1%
3	65.9%
5	63.7%
7	61.7%
9	60.1%

4.2 Wykresy

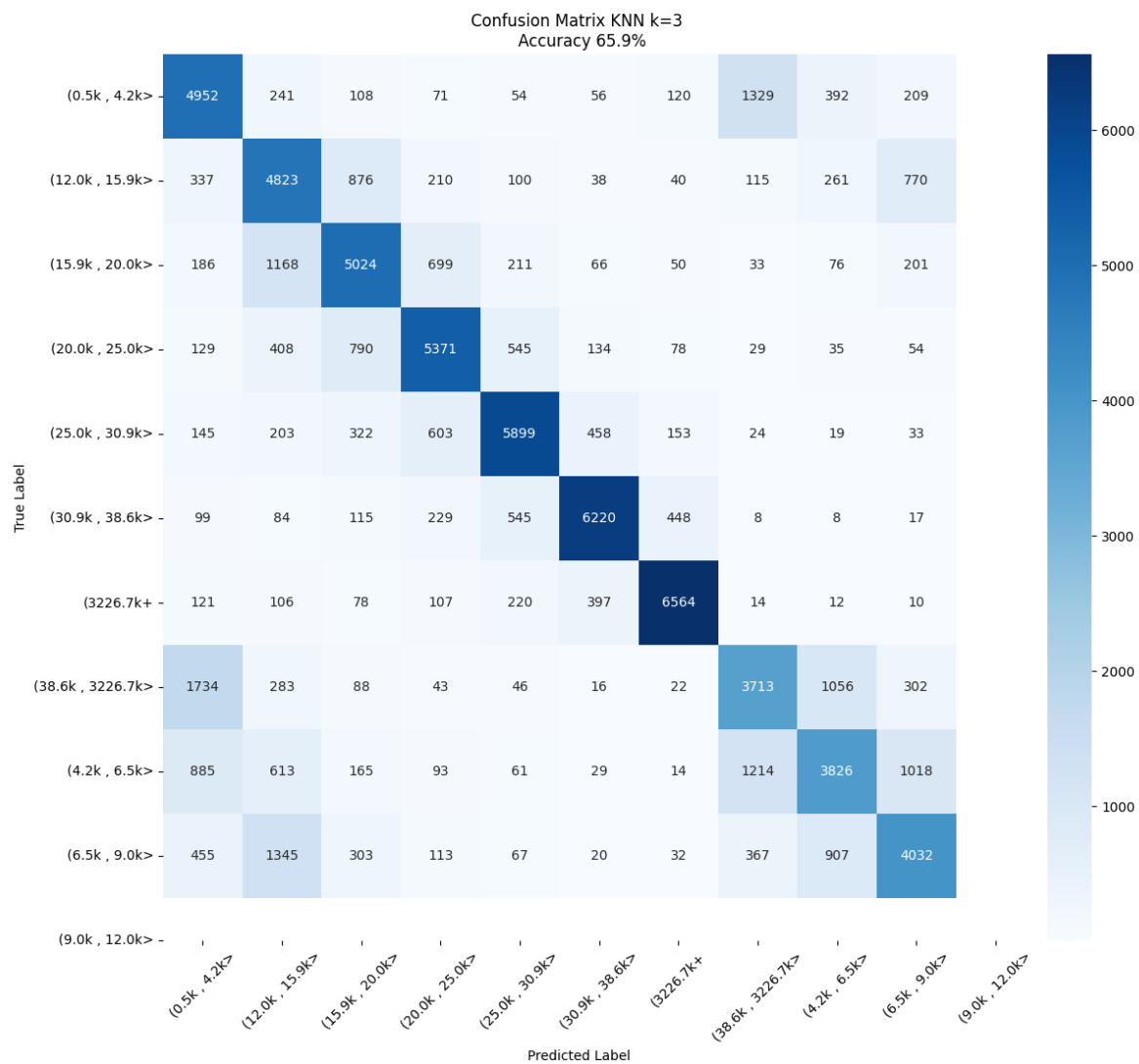
Jak widać na poniższych wykresach oraz na podstawie powyższej tabeli, można wyciągnąć wnioski, iż dane mogą być na tyle skorelowane, że z każdym kolejnym k dokładność spada.



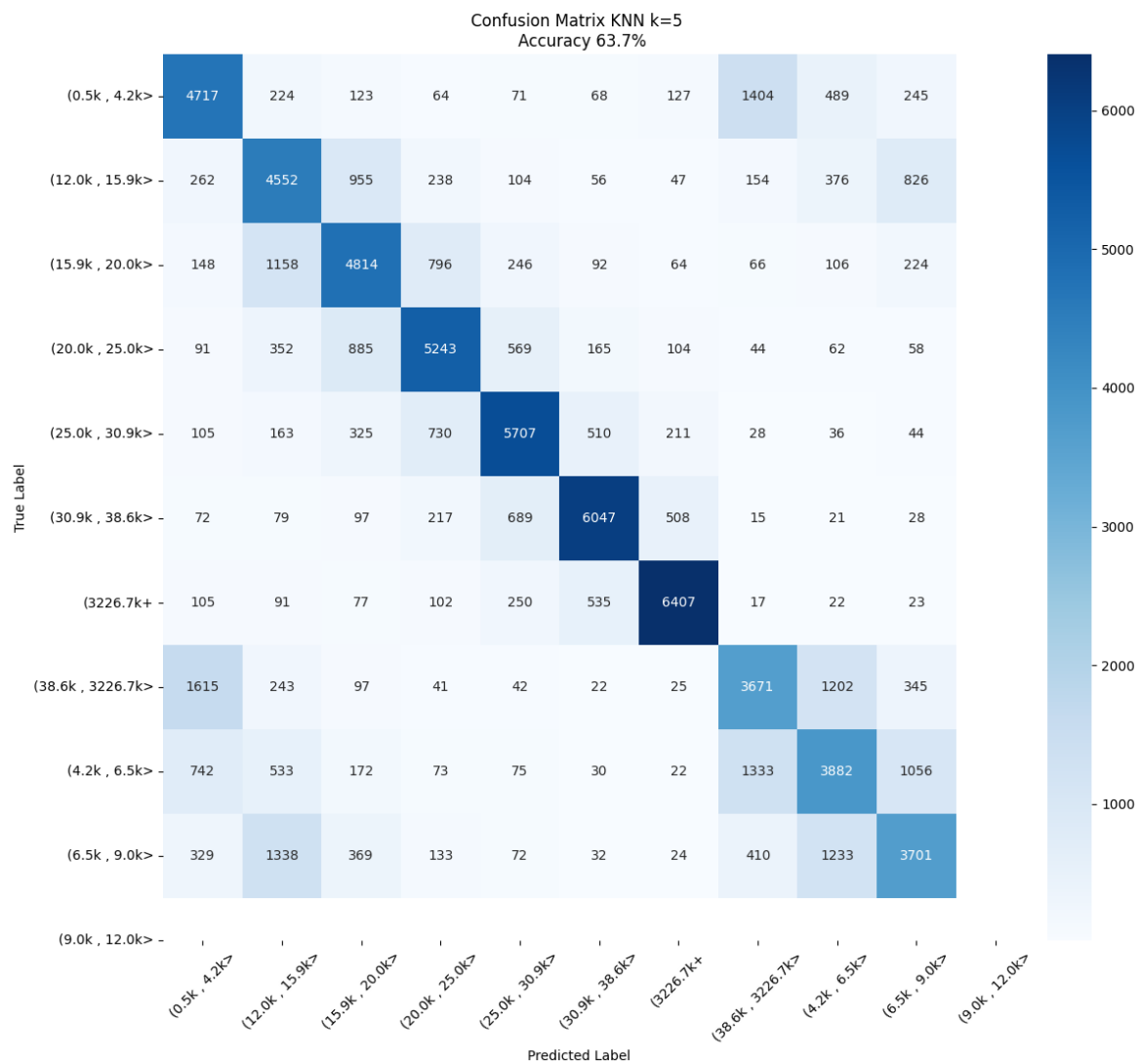
Rysunek 3: Wyniki dla poszczególnych wartości k.



Rysunek 4: Macierz błędów dla $k = 1$



Rysunek 5: Macierz błędów dla $k = 3$



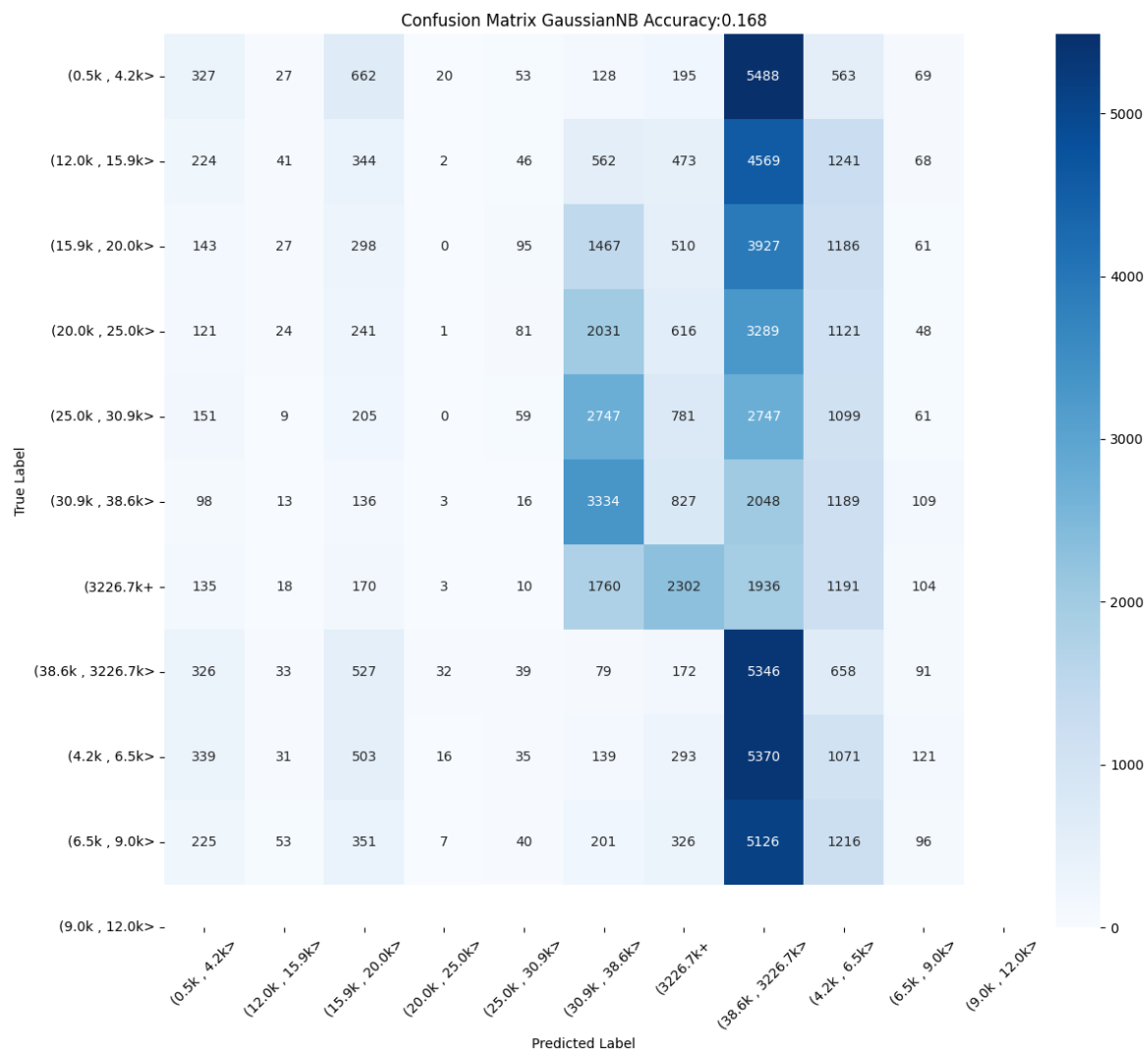
Rysunek 6: Macierz błędów dla $k = 5$

4.3 Naiwny Bayes (Naive Bayes)

Dla celów eksperymentu, na zbiorze danych postanowiłem przetestować następujące klasyfikatory:

4.3.1 GaussianNB

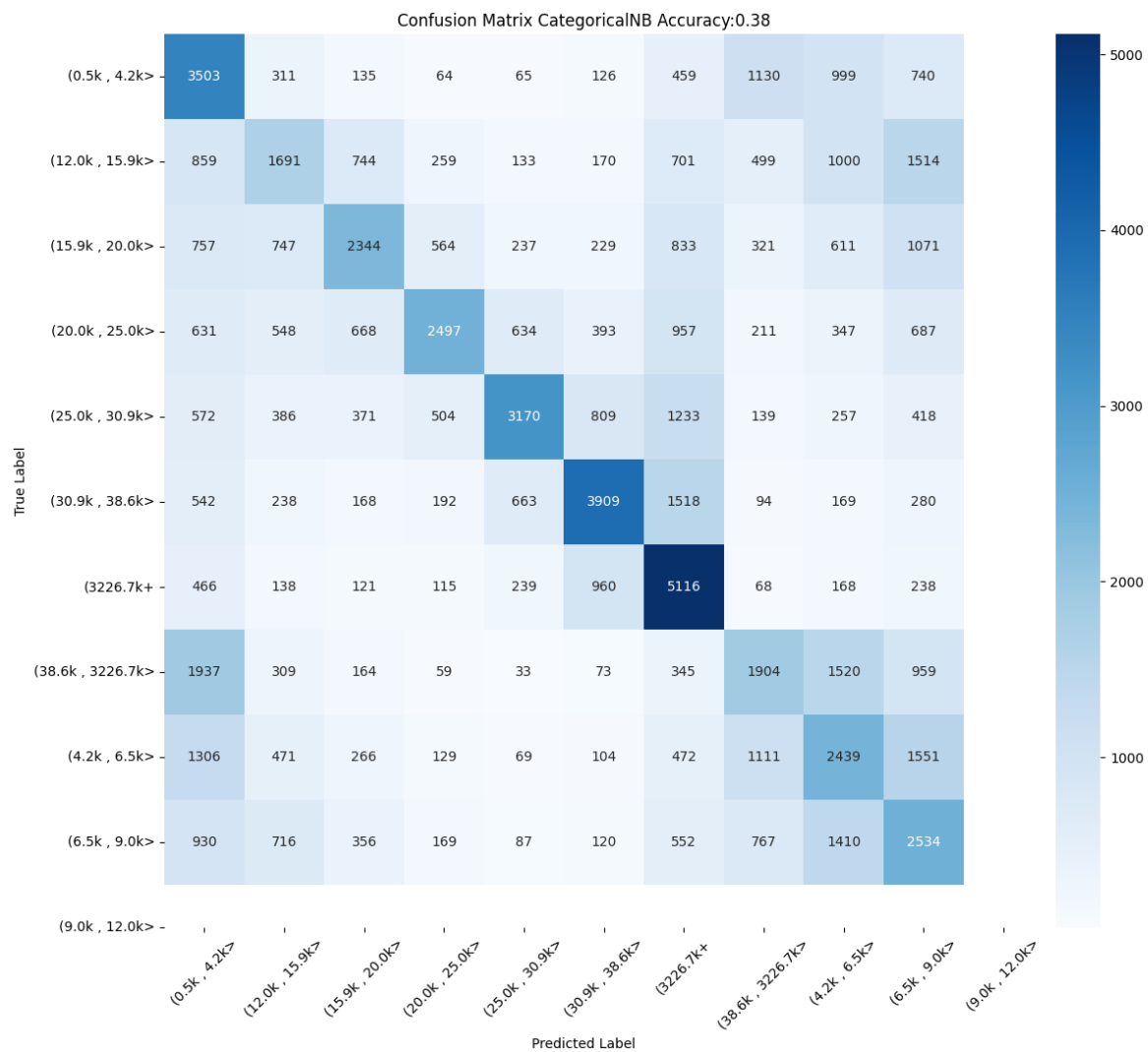
GaussianNB na zbiorze testowym osiągnął dość mierny wynik 17% co może sugerować, iż features nie są zbyt skorelowane, a ich rozkład można opisać jako rozkład normalny.



Rysunek 7: Macierz błędów dla GaussianNB

4.3.2 CategoricalNB

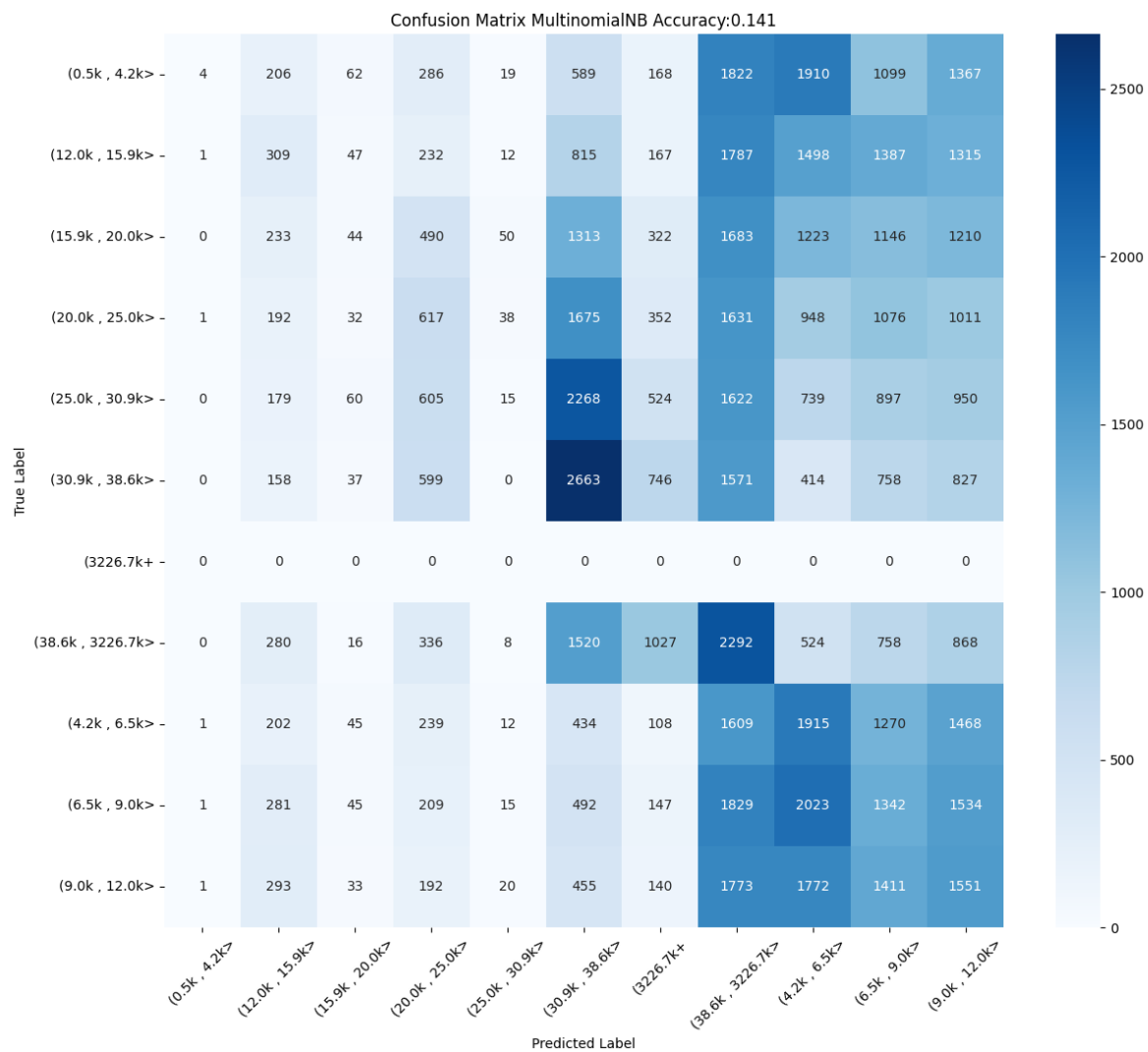
Ten klasyfikator osiągnął wynik 38% co oznacza, iż najlepiej sobie poradził ze zbiorem testowym ze wszystkich przetestowanych przeze mnie klasyfikatorów NB. Jego wyższa dokładność w porównaniu do innych klasyfikatorów typu NB może wynikać z wielomianowego rozkładu danych. Jest dobry dla wartości z zakodowanymi kategoriami.



Rysunek 8: Macierz błędów dla CategoricalNB

4.3.3 MultinomialNB

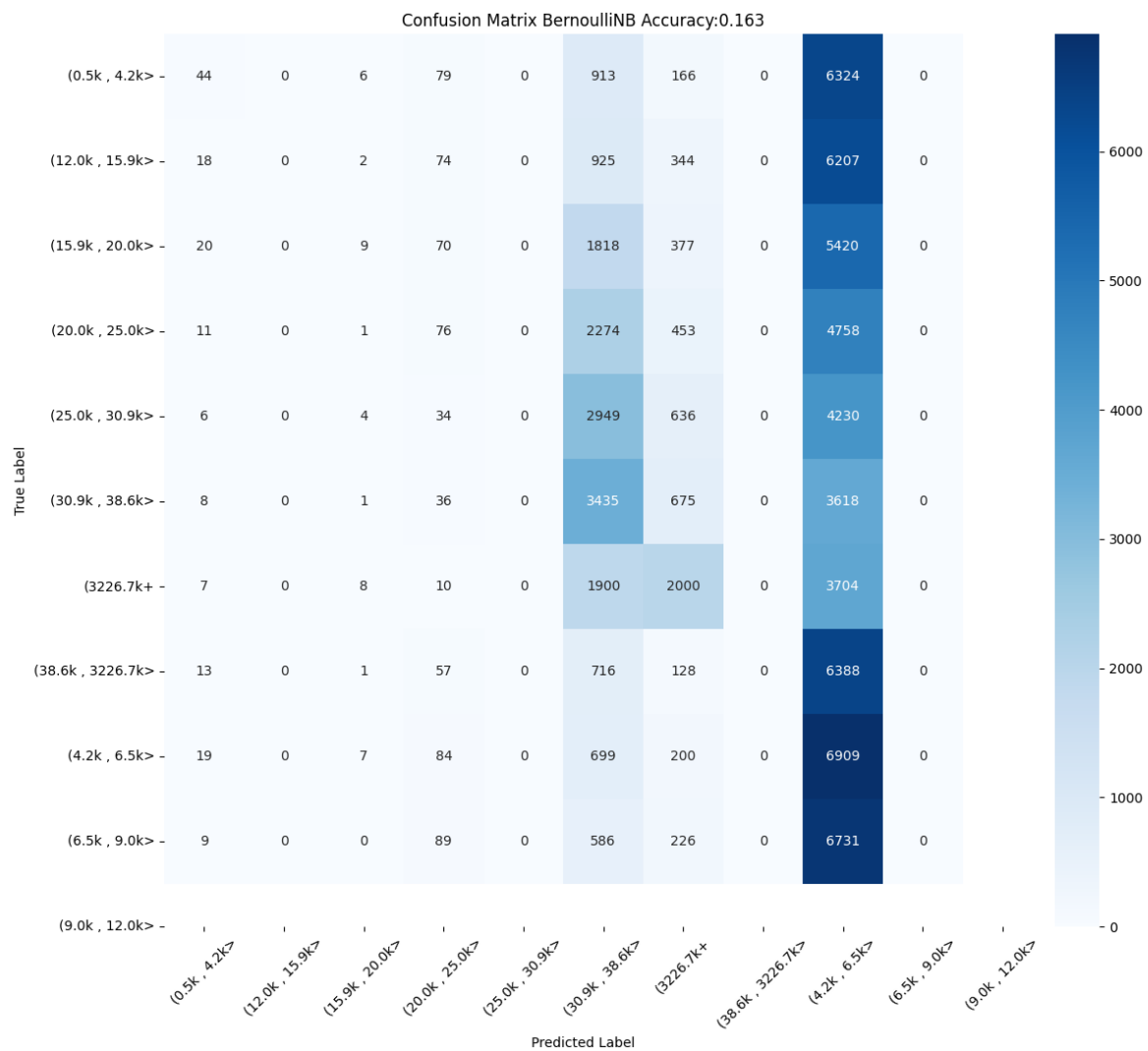
MultinomialNB jest podobny do CategoricalNB, ale jest bardziej ogólny, gdy chodzi o dane z cechami, które mogą być zliczane, takimi jak liczebność słów w dokumentach (np. w analizie tekstu). Ten klasyfikator jest często używany do klasyfikacji dokumentów tekstowych. W moim przypadku znormalizowane dane nie wykazują cech, które predysponowałyby użycie tego klasyfikatora. Stąd też jego wynik w postaci 14%.



Rysunek 9: Macierz błędów dla MultinomialNB

4.3.4 BernoulliNB

Ten klasyfikator jest używany, gdy dane wejściowe są binarne (takie jak wystąpienie lub nie wystąpienie danej cechy). Przykłady to dane uzyskane przez binarne kodowanie, takie jak obrazy binarne lub analiza sentymentu. Biorąc pod uwagę ilość klas w moim zbiorze danych, czyli 10, ten klasyfikator względnie poradził sobie z rozróżnieniem 3 klas, lecz w większości predyktowane były dane z jednej klasy, czego rezultatem był wynik 16% dokładności.

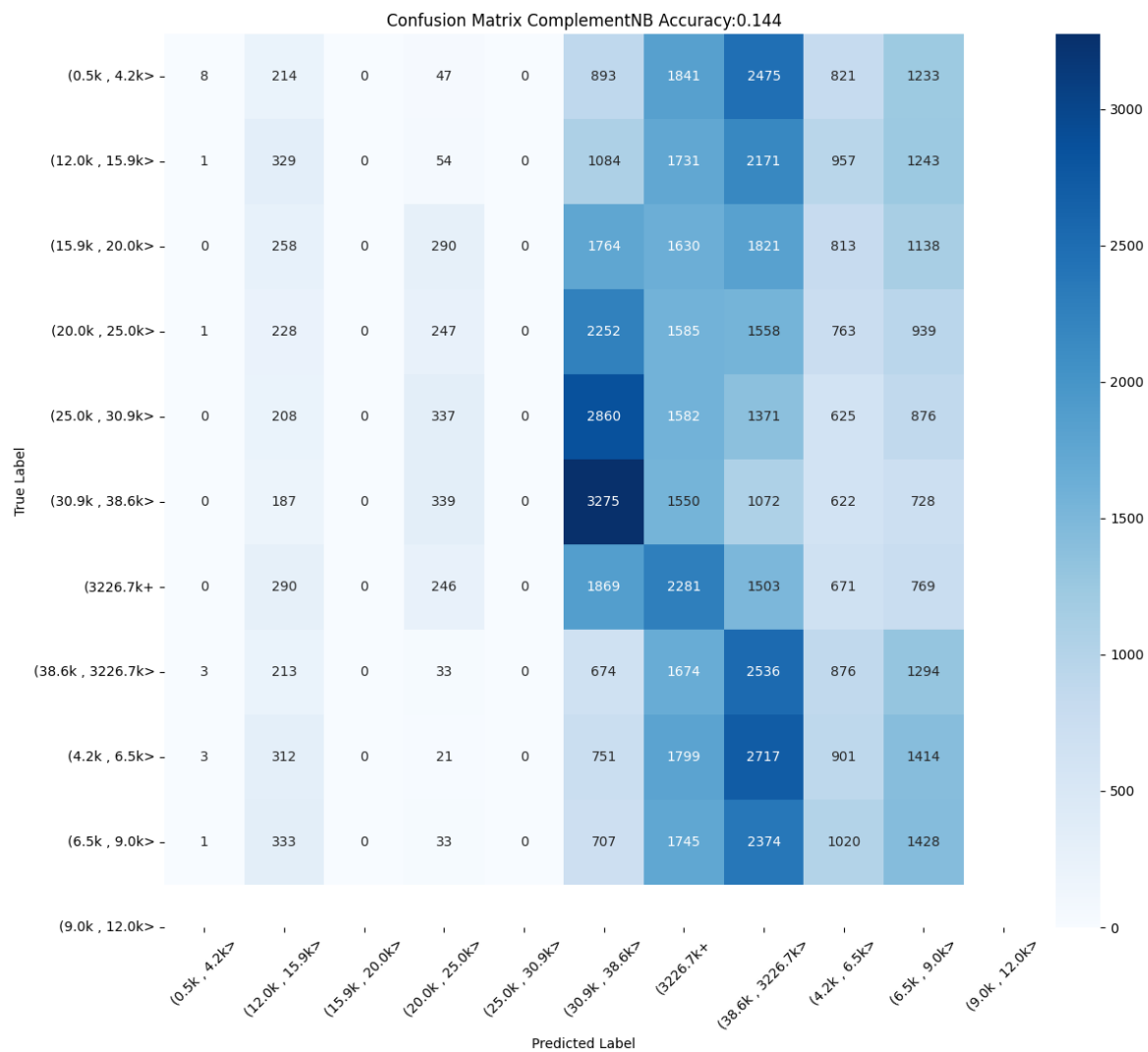


Rysunek 10: Macierz błędów dla BernoulliNB

4.3.5 ComplementNB

Ten klasyfikator jest wariantem MultinomialNB, który jest lepiej dostosowany do danych niezrównoważonych, gdzie jedna klasa jest bardziej liczna niż pozostałe. ComplementNB przeciwdziała problemowi związane z nierównomierną liczbą próbek klas poprzez obliczanie wag dla każdej klasy na podstawie jej częstości w danych treningowych.

W moim zbiorze danych wszystkie klasy są mniej więcej zrównoważone. W zbiorze testowym średnio przypada po 4000 przypadków każdej klasy, co stawia ten klasyfikator na przegranej pozycji z wynikiem zaledwie 14% dokładności.



Rysunek 11: Macierz błędów dla ComplementNB

5 Reguły asocjacyjne

Zdecyowana większość reguł potwierdzonych dużym wsparciem, dotyczy głównie roczników z przedziału 2001-2020, co nie wydaje się zbyt interesujące, bardziej konkretne i sensowne reguły chciałbym przedstawić poniżej.

Antecedents	Consequents	support	confidence	Lift
(price_ (0.5k , 4.2k>)	fuel_gas	0.094556	0.959064	1.136254
fuel_diesel	transmission_automatic	0.059554	0.880196	1.127777
transmission_automatic	fuel_gas	0.684027	0.876430	1.038353
transmission_manual	fuel_gas	0.053683	0.883084	1.046237
odometer_ <100k, 200k)	year_2001-2020	0.342249	0.926820	1.003494
odometer_ <100k, 200k)	transmission_automatic	0.341763	0.925503	1.185828

6 MLP

By skonstruować działający model MLP tym razem już nie za pomocą paczki scikit-learn, a tensorflow, zakodowałem preprocessowane katagoryczne dane za pomocą OneHotEncodera a numeryczne za pomocą MinMaxScalera. Wszystkie modele były trenowane oraz uczone na maksymalnie 10% danych (30 tys. rekordów), ponieważ 20% danych zwiększało zauważalnie czas uczenia, a nie przynosiło żadnych rezultatów, natomiast większa ilość kończyła się zatrzymaniem pracy systemu lub otrzymaniem ostrzeżenia o braku wystarczającej ilości pamięci RAM w il. 72GB, co w połączeniu z 32GB, które posiadam daje około 100GB pamięci potrzebnej do pracy na przynajmniej 30% danych. Do modeli użyłem następujących callbacków z biblioteki tensorflow.callbacks:

- History()
- ModelCheckpoint()
- EarlyStoppage()

Modele testowałem na maksymalnie 1000 epokach, choć większość z nich dochodziła do maksymalnie 100 epok i EarlyStoppage zatrzymywał uczenie. Ze względu na ilość klas w wysokości: 10, ostatnia warstwa zawsze miała funkcję aktywacji Softmax. Kamienie milowe eksperymentu opiszę pod tabelą z wynikami i wykresami.

```

# Funkcja generuj ca model
def create_model(shape):
    model = Sequential(
        [

            layers.Dense(128, activation="relu", input_shape=shape, name="layer2",
                kernel_regularizer=regularizers.l1(0.01)),
            layers.Dropout(0.5),
            layers.BatchNormalization(),
            layers.Dense(64, activation="relu", name="layer3"),
                layers.Dropout(0.5),
                layers.BatchNormalization(),
                layers.Dense(32, activation="relu", name="layer4"),
                layers.Dropout(0.5),
                layers.BatchNormalization(),
                layers.Dense(10, activation="softmax", name="output"),
            ],
        )
    model.compile(
        loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"]
    )
    return model

```

6.1 Wyniki

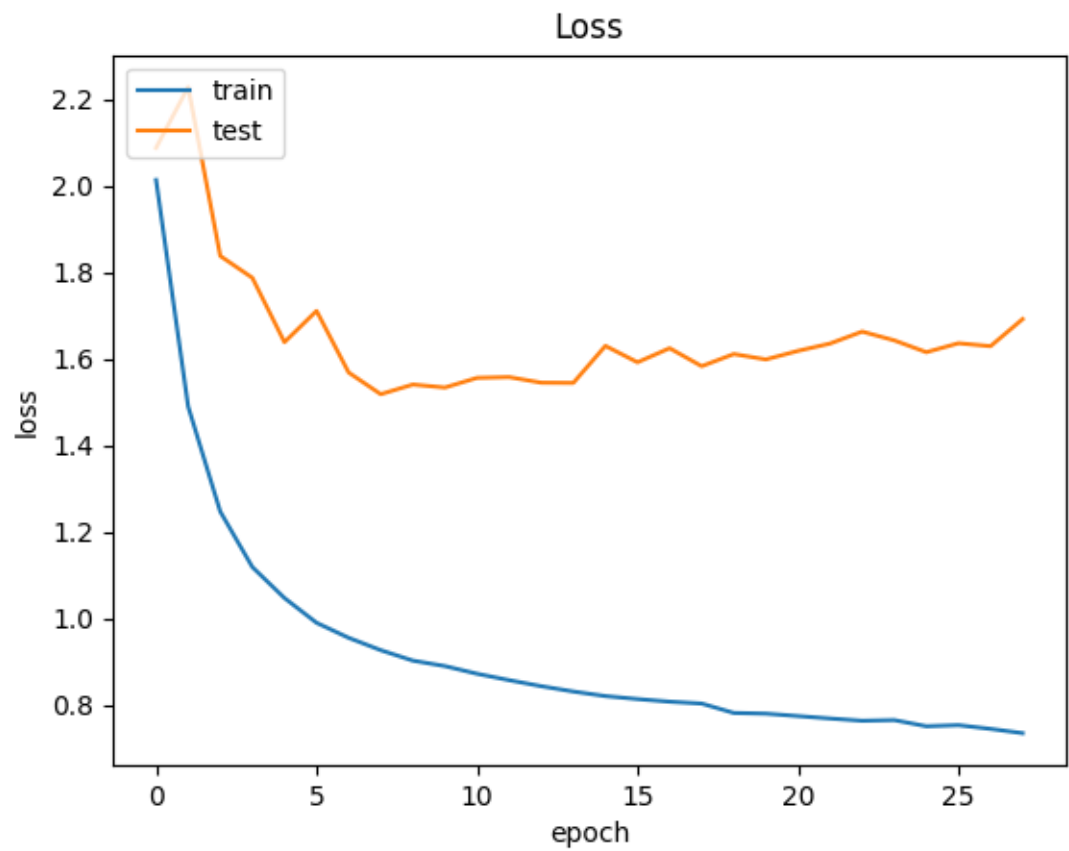
Tabela na następnej stronie.

1	warstwy	test_accuracy	test_loss	epochs(best)	batch_size	optimizer	Uwagi
2	Dense(10,)	35%	1.80	100	64	adam	warstwy relu
3	Dense(10)	24%	2.07	100	64	sgd	warstwy relu
4	Dense(10) Dense(10)	37%	1.68	136	64	adam i sgd	warstwy relu
5	Dense(30) Dropout(0.5) Dense(30) Dropout(0.5) Dense(10)	41%	1.61	12	64	adam	warstwy relu
6	Dense(128,) Dense(64) Dense(32) Dense(10)	46%	1.52	28	64	adam	BatchNormalization po każdej warstwie Dense, oprócz ostatniej warstwy relu
7	Dense(128) Dense(64) Dense(32) Dense(10)	45%	1.49	56	64	adam	Dropout(0.5) BatchNormalization po każdej warstwie Dense, oprócz ostatniej warstwy relu
8	Dense(128) Dense(64) Dense(32) Dense(10)	45%	1.48	91	32	adam	Dropout(0.5) BatchNormalization po każdej warstwie Dense, oprócz ostatniej warstwy relu
9	Dense(128, regularization=0.01) Dense(64) Dense(32) Dense(16) Dense(10)	40	1.6	100	64	adam	Dropout(0.5) BatchNormalization po każdej warstwie Dense, oprócz ostatniej Dropout(0.625) przed ostatnią. warstwy relu
10	Dense(128) Dense(64) Dense(32) Dense(16) Dense(10)	49%	1.4	92	64	adam	jak w sieci nr. 9 tanh jako funkcja aktywacji oprócz ostatniej warstwy

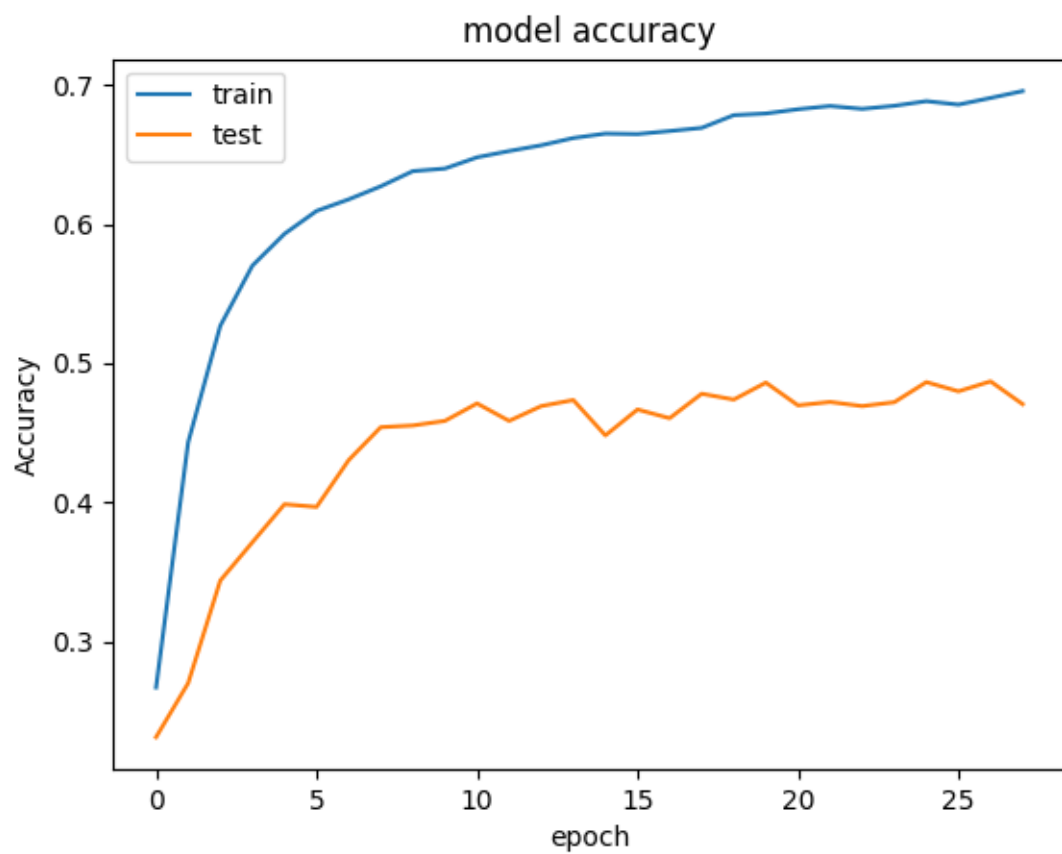
6.1.1 Kamienie milowe

Sieć nr. 6:

Jak widać na poniższych wykresach, model był ewidentnie przeuczony, co starałem się wyeliminować w następnych próbach, lecz mimo tego ta sieć osiągnęła prawie największą dokładność na zbiorze testowym. Użycie warstw Dropout oraz batchNormalization

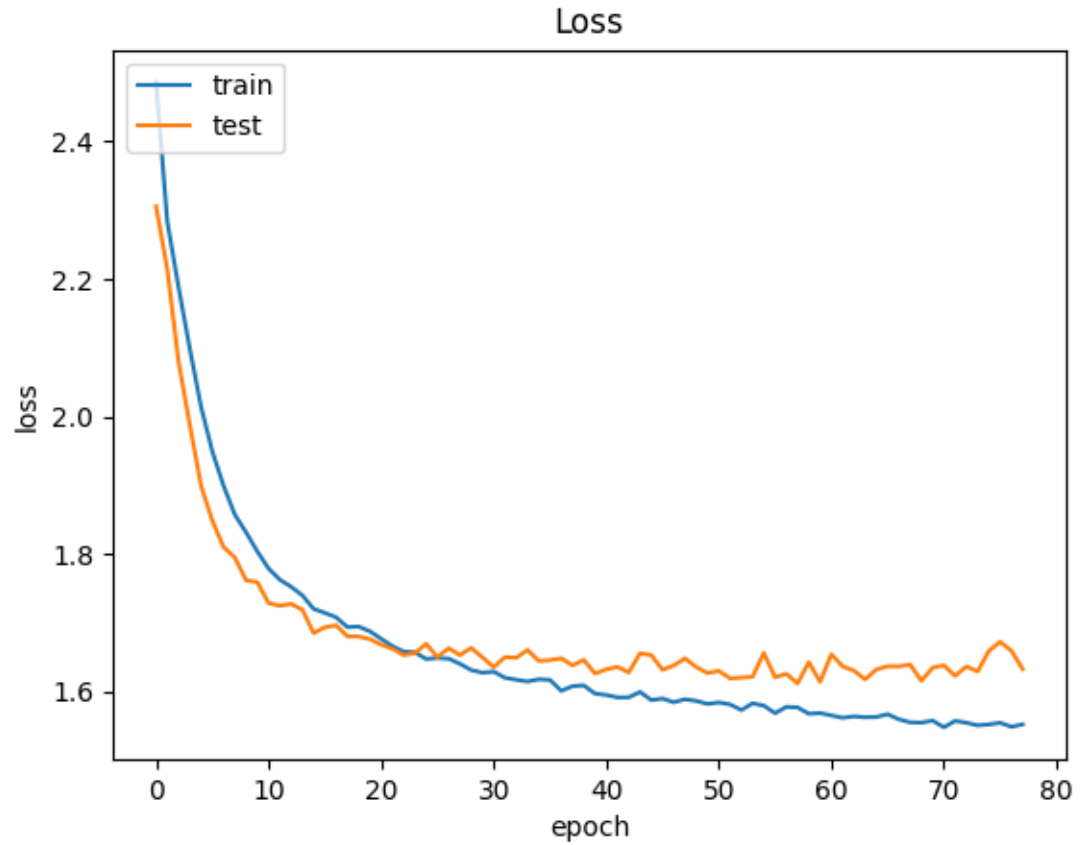


Rysunek 12: Wykres straty sieci nr. 6

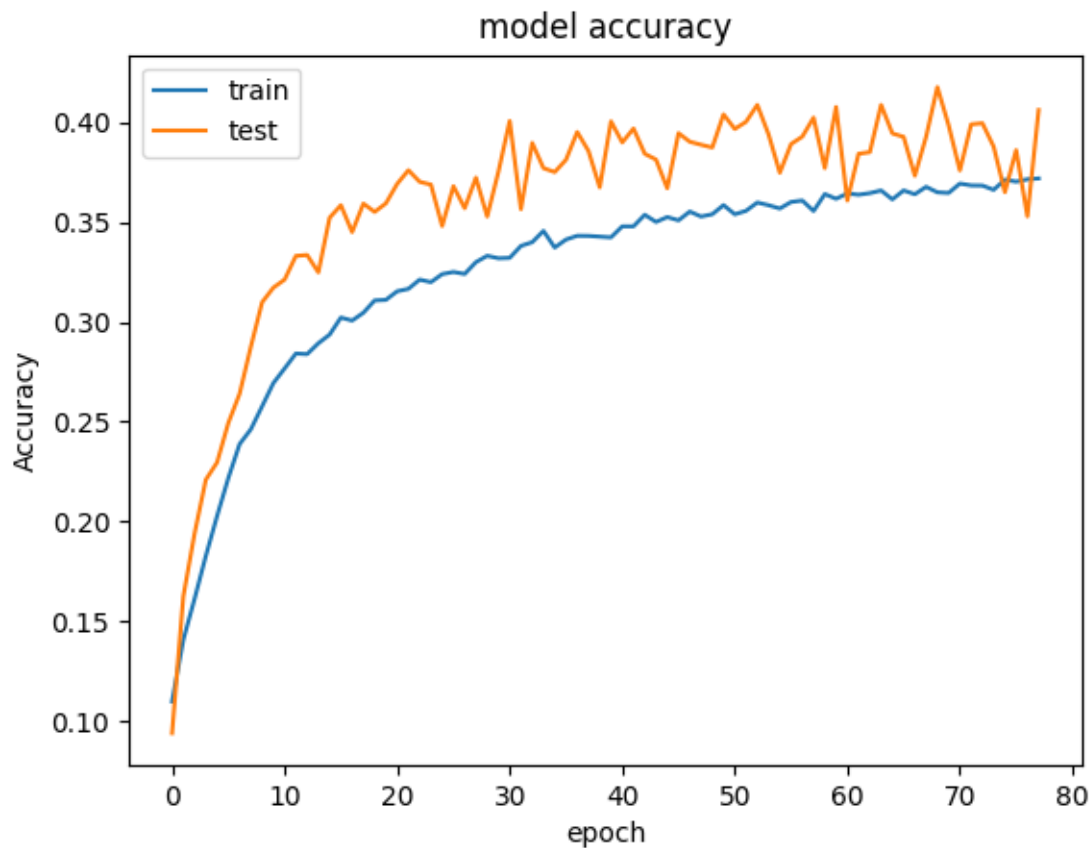


Rysunek 13: Wykres accuracy sieci nr. 6

Sieć nr. 9: W tej warstwie postanowiłem skorzystać z regularizera L2, by zmniejszyć przeuczenie. Wykres jest widocznie bardziej zbliżony do wykresu idealnej krzywej, lecz regularizer oprócz efektów wizualnych przyniósł jedynie negatywne rezultaty w zmniejszonej o 5% dokładności.

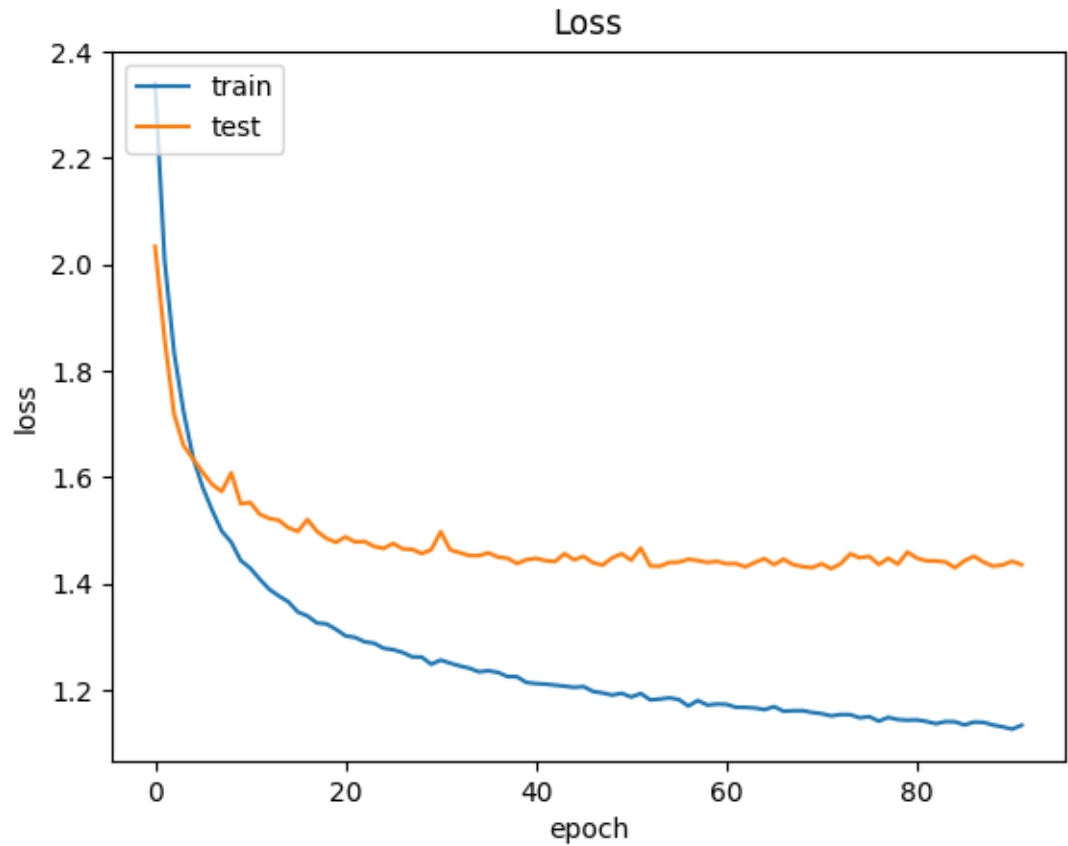


Rysunek 14: Wykres straty sieci nr. 9



Rysunek 15: Wykres accuracy sieci nr. 9

Sieć nr. 10: Sieć nr. 10 była pierwszą siecią, przy której postanowiłem skorzystać z funkcji aktywacji tanh w każdej z warstw typu Dense, co pozwoliło na wynik rzędu 49% dokładności na zbiorze testowym i zauważalnie mniejszej wartości straty rzędu ok. 1.4, aczkolwiek ponownie model był przeuczony, lecz tego już nie udało mi się poprawić, gdyż dodając jeszcze jedną warstwę model osiągał już niższą dokładność przy o wiele dłuższym czasie uczenia się.

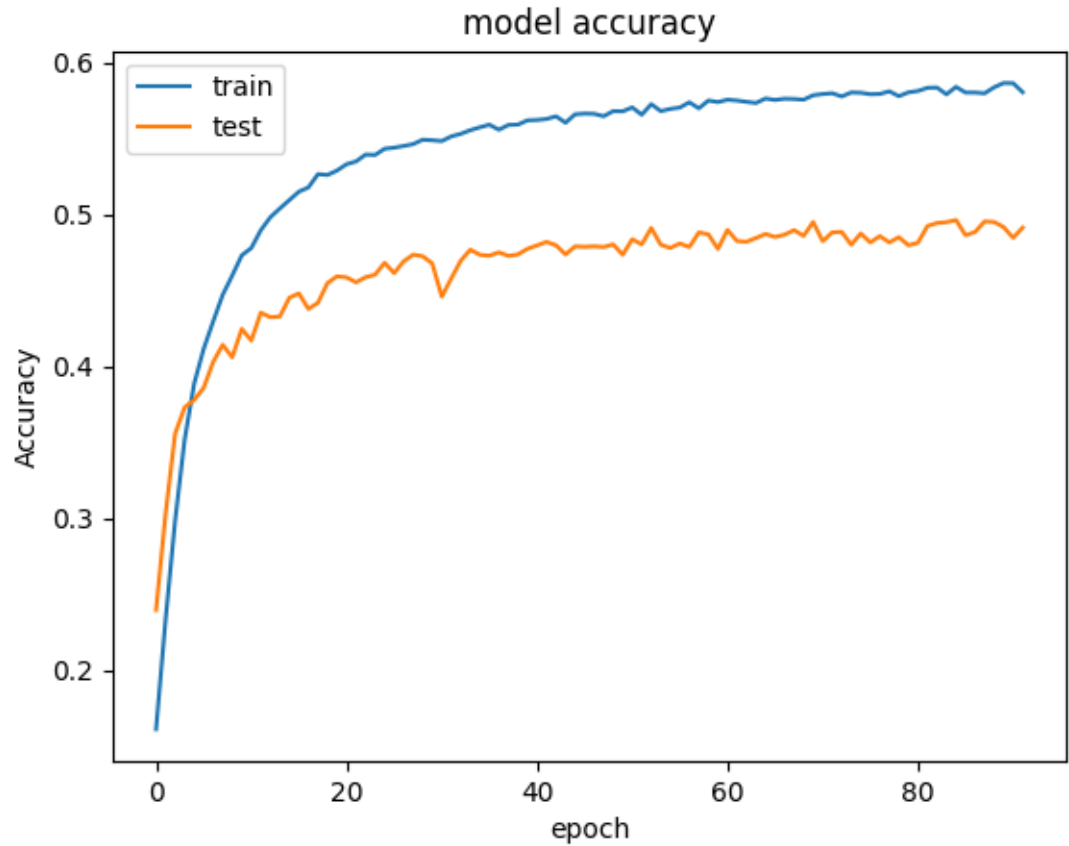


Rysunek 16: Wykres straty sieci nr. 10

Wykres accuracy sieci nr. 10 znajduje się pod bibliografią, nie jestem stanie wyjaśnić dlaczego, ale niezależnie od ustawień, nie chce on powrócić na swoje miejsce.

7 Bibliografia

- Dataset: <https://www.kaggle.com/datasets/austinreese/craigslist-carstrucks-data>
- Tutoriale:
 - <https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn>



Rysunek 17: Wykres accuracy sieci nr. 10

- <https://www.datacamp.com/tutorial/k-nearest-neighbor-classification-scikit-learn>
- <https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/>
- Dokumentacja:
 - <https://scikit-learn.org/stable/>
 - https://www.tensorflow.org/api_docs/python/tf/keras/Regularizer https://www.tensorflow.org/guide/keras/sequential_model $hl = pl$
- https://www.tensorflow.org/tutorials/keras/overfit_and_underfit $hl = pl$
- ChatGPT: <https://chatgpt.com/>

8 Zakończenie

Podsumowując: W moim eksperymencie zdecydowanie najlepsze rezultaty osiągnął klasyfikator KNN oraz DecisionTree.

Marne wyniki niestety osiągnęły zarówno klasyfikatory NB jak i sieć MLP.

Brak efektywności klasyfikatorów NB wynika wprost z niedopasowania danych do rozkładu, do których są przeznaczone klasyfikatory. Efektywność sieci w wysokości około 49% mogłaby się poprawić na danych pełnowymiarowych, jak i przy wykorzystaniu większej ilości warstw, neuronów, których naturalnie w domowym zaciszu zastosować nie mogę ze względu na brak mocy obliczeniowej oraz pamięci. Aczkolwiek jak na moją pierwszą prawdziwą styczność z sieciami neuronowymi, wynik rzędu 49% uznaję za satysfakcjonujący, ze względu na ilość klas wyjściowych w ilości: 10. Pierwszym przykładem przychodzącym mi na myśl może być taki samochód jak Porsche 911, którego podstawowy wariant może być kilka-kilkanaście razy tańszy niż najmocniejsza i najlepiej wyposażona wersja.

Na pierwszy rzut oka mogę stwierdzić, iż tokenizacja opisu mogłaby przynieść dobre rezultaty, gdyż stamtąd można by uzyskać takie informacje jak:

- czy sprzedający zajmuje się handlem zawodowo
- jaka jest wersja wyposażenia pojazdu
- czy samochód był naprawiany lub serwisowany

Podjęrzewam, że obydwie te klasyfikatory osiągnęły, by lepsze wyniki w wersji regresywnej, bez podziału na kategorie cenowe.