Miniprojekt II

Metody probabilistyczne w uczeniu maszynowym

Łukasz Trzos

0 Temat projektu

Plik **rp.data** zawiera wyniki badań diagnostycznych pozwalających stwierdzać, czy wykryty rak piersi jest łagodny czy złośliwy. Każdy wiersz pliku zawiera 10 liczb: pierwsze dziewięć z nich odpowiada wynikom dziewięciu różnych parametrów (każdy o wartości od 1 do 10), natomiast ostatnia dana jest prawdziwą informacją o nowotworze: 2 oznacza, że jest on łagodny, natomiast 4 świadczy o złośliwości.

Porównaj działanie regresji logistycznej oraz naiwnego klasyfikatora bayesowskiego **implementując** oba algorytmy i korzystając z danych zawartych w pliku **rp.data**. Załóż, że wszystkie dziewięć cech jest od siebie niezależnych.

Implementując klasyfikator bayesowski zastosuj wygładzenie Laplace'a oraz załóż, że dla cech $x_j \in \{1, ..., 10\}$, gdzie $j = 1, ..., 9$, zachodzi

$$p(x_j = d|y = c) = \phi^j_{c,d}, \quad \text{gdzie} \quad \sum_{d=1}^{10} \phi^j_{c,d} = 1,$$

czyli że zmienne $x_j|y = c$ mają rozkład wielomianowy. Implementując regresję logistyczną, możesz dodać składnik regularyzujący.

W obu algorytmach wykorzystaj zbiór 2/3 obserwacji jako zbiór treningowy, natomiast pozostałą 1/3 jako zbiór testowy. Zwróć uwagę na to, żeby wybrać 2/3 obserwacji dla każdej z klas.

Dla obu algorytmów **stwórz wykres** zawierający krzywe uczenia, czyli przedstawiające błąd klasyfikacji jako funkcję rozmiaru zbioru treningowego. Zaznacz punkty odpowiadające błędom obliczonym na całym zbiorze testowym po zastosowaniu algorytmu na następujących frakcjach zbioru treningowego: 0.01, 0.02, 0.03, 0.125, 0.625, 1. Aby uwiarygodnić wyniki, uśrednij co najmniej 5 przebiegów algorytmu na losowych wyborach obserwacji do zbioru treningowego i testowego.

Opisz wnioski jakie możesz wyciągnąć na podstawie osiągniętych wyników. Co możesz powiedzieć o zbieżności krzywej uczenia w obu przypadkach?

Zapoznaj się z artykułem *On Discriminative vs. Generative Classifiers: A comparasion of logistic regression and naive Bayes* Andrew Nga i Michaela Jordana. Czy otrzymane przez Ciebie wyniki zgadzają się z teoretycznymi wynikami opisanymi w tej pracy?

1 Naiwny klasyfikator bayesowski

1.1 Opis teoretyczny modelu

Zakładając, że wszystkie 9 cech jest od siebie niezależnych, oraz korzystając z tw. Bayesa:

$$p(y|x_1, x_2, ..., x_9) \propto p(y) \prod_{i=1}^{9} p(x_i|y)$$

Mając dany zbiór treningowy D o rozmiarze m, obliczamy następujące prawdopodobieństwa:

$$p(y) = \frac{\text{Liczba próbek w D, gdzie klasa jej przypisana to y}}{|D|}$$

$$p(x_i|y) = \frac{\text{Liczba próbek w D, gdzie i-ta zmienna wynosi } x_i \text{ oraz klasa jej przypisana to y} + \alpha}{\text{Liczba próbek w D, gdzie klasa jej przypisana to y} + 10\alpha}$$

gdzie w drugim równaniu $\alpha > 0$ to współczynnik wygładzenia Laplace'a, który w mianowniku mnożymy przez liczbę wartości osiąganych przez cechę x_i. Dzięki jego zastosowaniu w obliczeniach nie mamy zerowych prawdopodobieństw, zatem przypisanie nowego zestawu danych do dowolnej klasy jest teoretycznie możliwe. W późniejszym etapie przeprowadzimy porównanie skuteczności modelu dla różnych wartości tego współczynnika. Zakładamy, że w D znajduje się przynajmniej jedna próbka z każdej rozważanej klasy, inaczej model nie byłby w stanie niczego się o danej klasie nauczyć.

Dla każdego nowego zestawu cech model oblicza, która z wartości $p(y = 2|x_1, x_2, ...x_9)$ oraz $p(y = 4|x_1, x_2, ...x_9)$ jest większa, a następnie wybiera odpowiednią klasę.

Poniższy kod przedstawia funkcję, która na podstawie zbioru treningowego oraz współczynnika wygładzenia Laplace'a produkuje funkcję decyzyjną, której następnie będziemy używać przy klasyfikacji nowych rekordów.

```python
# 01 - Naive binary bayes classificator

import math

def GetDecisiveFunction(trainingDataset, laplaceSmoothingCoefficient):

    presentClasses = list({record[9] for record in trainingDataset})

    if len(presentClasses) < 2:
        raise ValueError("01: Too few classes are present in the training dataset")


    #priorProbabilities[y]
    #p(y==?)
    priorProbabilities = ...


    #conditionalProbabilities[j][k][y]
    #p(x_j==k+1|y==?)
    conditionalProbabilities = ...

    #Takes [x_1,x_2,...,x_9], returns (predicted class, probability of that class)
    def DecisiveFunction(X):

        if len(X) != 9:
            raise ValueError("DecisiveFunction: Wrong arguments")

        calculatedClassProbabilities = {y: (priorProbabilities[y] * math.prod(conditionalProbabilities[j][X[j]-1][y] for j in range(9))) for y in presentClasses}

        totalProb = sum(calculatedClassProbabilities.values())

        maxProb = max(calculatedClassProbabilities.values())/totalProb

        bestClass = max(calculatedClassProbabilities, key=calculatedClassProbabilities.get)

        return bestClass, maxProb

    return DecisiveFunction
```

Rysunek 1: Kod naiwnego klasyfikatora bayesowskiego

1.2 Wyznaczanie wartości współczynnika wygładzenia

Wybór jak najlepszego współczynnika wygładzenia wpływa znacząco na skuteczność modelu. Zbyt małe wartości mogą prowadzić do przetrenowania modelu, zbyt duże mogą z kolei za bardzo generalizować jego predykcje

Model przedstawiony w poprzedniej sekcji dla klasy, którą wskazał poprzez obliczenie odpowiednich wartości prawdopodobieństwa, podaje również z jakim prawdopodobieństwem szacuje, że jego predykcja jest właściwa. Załóżmy, że dla zestawu cech ze zbioru testowego $x_1, x_2, ..., x_9, y$ model podał klasę $\hat{y}$ z prawdopodobieństwem p. Możemy zdefiniować w prosty sposób funkcję straty:

$$J(X_{test}) = \frac{1}{m} \sum_{i=1}^{m} \mathbf{1}[\hat{y}_i = y_i](1-p) + \mathbf{1}[\hat{y}_i \neq y_i]p$$

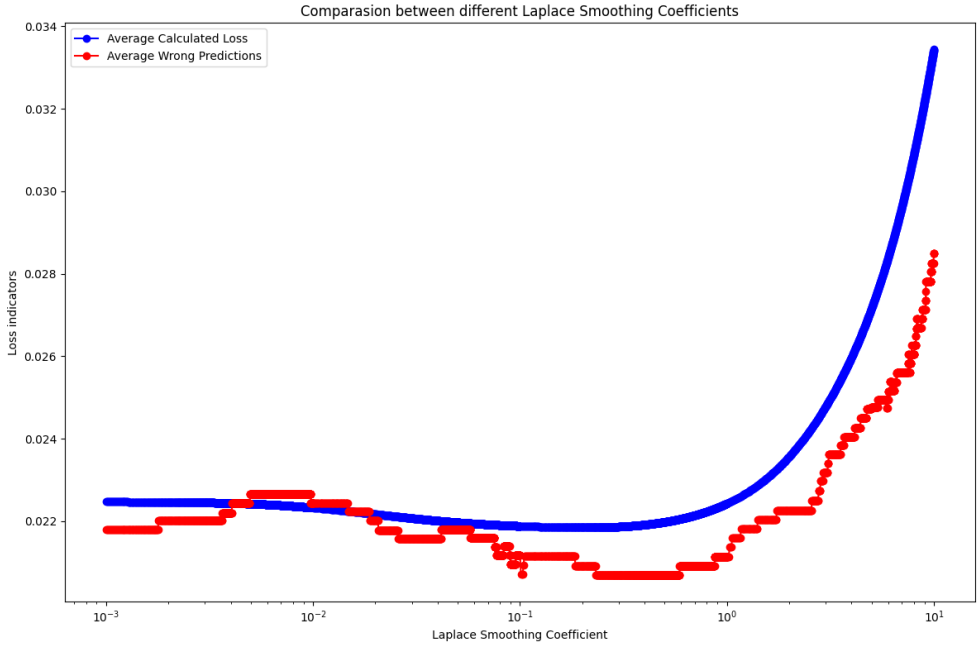
gdzie $m = |X_{test}|$.

Jest to dokładniejszy sposób na sprawdzenie modelu niż samo porównywanie jego liczby dobrych i złych predykcji dla zbioru testowego. W eksperymencie stworzymy funkcje decyzyjne dla różnych współczynników wygładzenia dla tych samych podziałów danych do zbiorów treningowych i testowych.

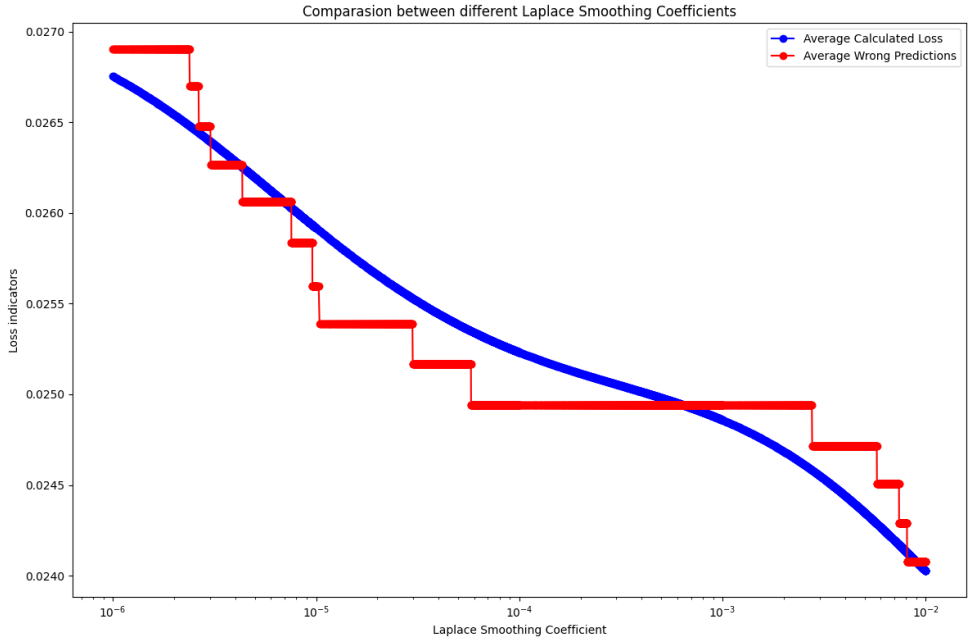
Wykresy widoczne na kolejnej stronie przedstawiają uzyskane wyniki uśrednione na kilkudziesięciu niezależnych próbach. Niebieskie punkty oznaczają średnią wartość opisanej wcześniej funkcji straty, czerwone oznaczają średnią liczbę nieprawidłowych obstawień.

Jak widać, optymalna wartość współczynnika mocno zależy od wylosowanych zbiorów danych, w szczególności, że wielkość zbioru treningowego jest stosunkowo niewielka. Widzimy jednak, że dla dowolnego zbioru z wysokim prawdopodobieństwem optymalna wartość znajduje się w przedziale $[10^{-4}, 3]$, poza nim jest ona już dużo większa. Możemy więc w naszym modelu zastosować metodę walidacji krzyżowej na zbiorze treningowym na tym przedziale w celu wyznaczania optymalnego współczynnika, którym będziemy starać się objaśniać zbiór testowy.

Co ciekawe, wartości obydwu parametrów porównywania skuteczności modeli są do siebie zbliżone. Dzieje się tak, ponieważ model oblicza prawdopodobieństwo porównując iloczyny 10 prawdopodobieństw warunkowych, często o innych rzędach wielkości. Zazwyczaj jeden z nich jest nieporównywalnie większy od drugiego, przez co model obstawia klasę z prawdopodobieństwem $p \approx 1$, zatem w każdym przypadku ponosi stratę bliską 0 lub bliską 1.



Rysunek 2: Porównanie skuteczności współczynników



Rysunek 3: Porównanie skuteczności współczynników - część 2

1.3 Testowanie

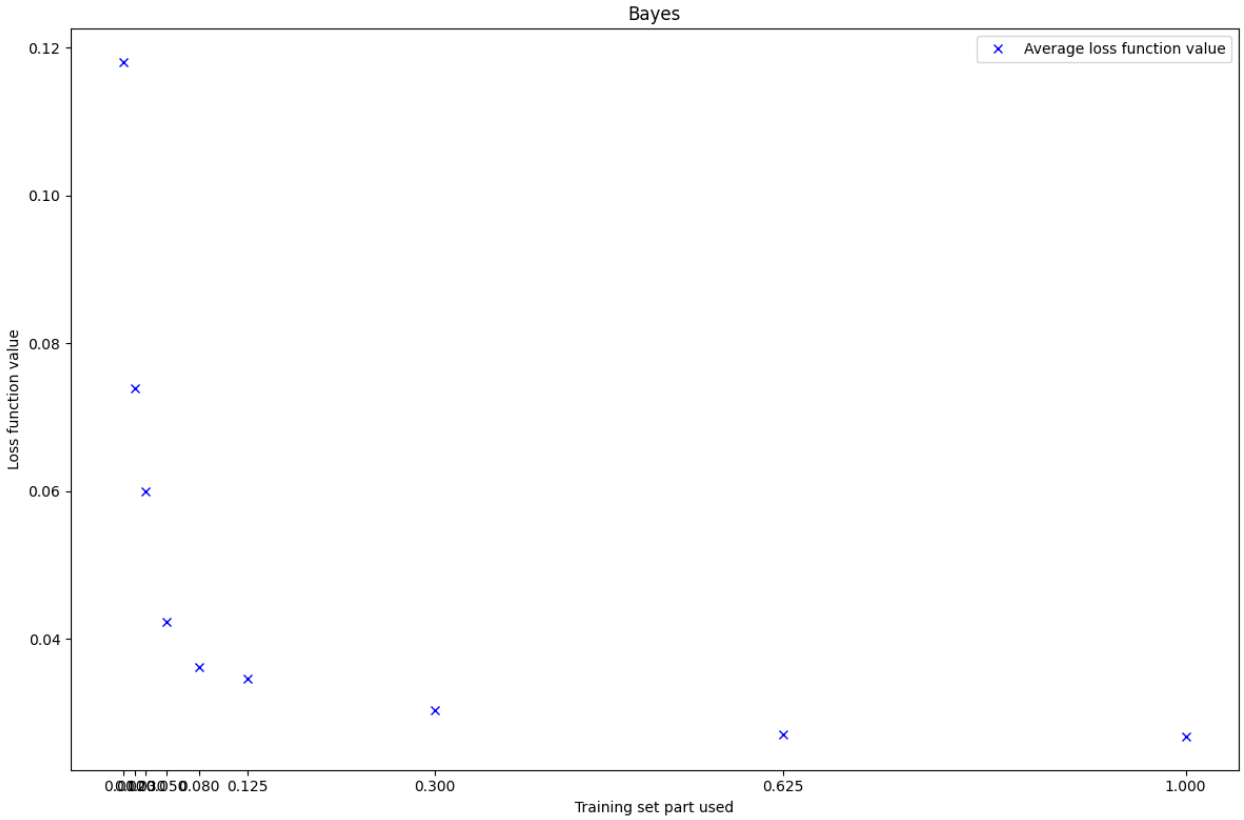
Po otrzymaniu zbioru treningowego wykorzystamy technikę walidacji krzyżowej w celu wyznacznia najlepiej objaśniającego go współczynnika wygładzenia. Algorytm wygląda następująco:

- Weź listę kilku możliwych wartości współczynnika (początkowo 9 liczb w przedziale $[10^{-4}, 3]$ oddalonych od siebie równo na skali logarytmicznej
- Podziel zbiór treningowy na k mniej więcej równych co do wielkości podzbiorów (u nas $k = 5$)
- Dla każdego potencjalnego współczynnika k razy: Wybierz jeden podzbiór jako walidacyjny, wytrenuj funkcję decyzyjną na podstawie pozostałych k-1 podzbiorów oraz rozważanego współczynnika, oblicz wartość funkcji straty i dopisz do listy
- Wybierz współczynnik, który uzyskał najlepsze wyniki
- Ponów poszukiwanie na przedziale wokół tego współczynnika

Jako funkcję błędu stosowaną do analizy trafności danego modelu wykorzystamy funkcję opisaną w poprzednim podrozdziale.

Warto zaznaczyć, że cały zbiór danych jest relatywnie mały (zawiera kilkaset próbek), dla niewielkich części zbioru treningowego może się zdarzyć, że nie będzie on zawierał ani jednej próbki z żadnej klasy. Nie zawsze będzie więc możliwe zebranie danych do prezentacji krzywej uczenia się dla wszystkich wskazanych części zbioru treningowego. Po sprawdzeniu empirycznym nie dzieję się to jednak zbyt często, dalej zbieramy wystarczająco dużo danych z prób dla niewielkich części zbioru, nie zaburzy to więc kształtu krzywej uczenia.

Poniższy wykres przedstawia krzywą uczenia dla opisanego modelu:



Rysunek 4: Krzywa uczenia naiwnego klasyfikatora bayesowskiego

2 Regresja logistyczna

2.1 Opis teoretyczny modelu

Mówiąc o przypadku pozytywnym mamy na myśli sytuację, w której $y = 4$, czyli wykryto nowotwór złośliwy.

W modelu regresji logistycznej zakładamy, że prawdopodobieństwo bycia przypadkiem pozytywnym dla rekordu o danym wektorze cech jest postaci:

$$p(y = 4|x) = \sigma(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}}$$

gdzie x to wektor cech. Jeśli obliczone prawdopodobieństwo wyjdzie większe niż $\frac{1}{2}$, to model wskaże przypadek pozytywny, w przeciwnym wypadku negatywny. Chcemy znaleźć jak najlepsze w oraz b.

Zdefiniujemy nieco inną funkcję straty niż tą wykorzystaną w klasyfikatorze bayesowskim. Załóżmy od teraz, że dla klasy pozytywnej $y = 1$ (zamiast $y = 4$) oraz, że dla klasy negatywnej $y = 0$ (zamiast $y = 2$). Nasz model dla wektora cech x będzie podawał wartość $\hat{y} \in [0, 1] = p(y = 1|x)$. Dla zbioru $Y = \{y_1, y_2, ..., y_m\}$ uwzględniając prawdopodobieństwa $\hat{Y} = \{\hat{y_1}, \hat{y_2}, ..., \hat{y_m}\}$, prawdopodobieństwo zaobserwowania następującego zbioru jest postaci

$$L = \prod_{i=1}^{m} (\hat{y_i})^{y_i} (1 - \hat{y_i})^{(1-y_i)}$$

Wyciągnięcie logarytmu zachowa porządek wartości używanych do porównywania oraz ułatwi obliczenia:

$$log(L) = \sum_{i=1}^{m} y_i log(\hat{y_i}) + (1 - y_i) log((1 - \hat{y_i}))$$

Funkcją straty, którą wykorzystamy do ewaluacji modelu będzie ujemna wartość funkcji prawdopodobieństwa dla danego zbioru danych podzielona przez wielkość zbioru:

$$J(w, b) = -\frac{L}{m} = -\frac{1}{m} \sum_{i=1}^{m} y_i log(\hat{y_i}) + (1 - y_i) log((1 - \hat{y_i}))$$

gdzie $\hat{y_i} = \sigma(w^T x_i + b)$ jest wartością klasy przewidywanej przez model na podstawie wektora cech.

W celu wyznaczenia najlepszych wartości w i b posłużymy się metodą gradientową. W tym celu musimy znać pochodną używanej funkcji straty. Obliczamy:

Niech $z_i = w^T x_i + b$. Wtedy $\hat{y_i} = \sigma(z_i)$. Obliczamy kolejno:

$$\frac{dJ}{d\hat{y_i}} = -\frac{1}{m}(\frac{y_i}{\hat{y_i}} - \frac{1 - y_i}{1 - \hat{y_i}})$$

$$\frac{d\hat{y_i}}{dz_i} = \frac{d}{dz_i} \frac{1}{1 + e^{z_i}} = \frac{-e^{z_i}}{(1 + e^{z_i})^2} = \hat{y_i}(1 - \hat{y_i})$$

$$\frac{dz_i}{dw} = x_i$$

$$\frac{dz_i}{db} = 1$$

Następnie korzystamy z reguły łańcuchowej:

$$\frac{dJ}{dw} = \sum_{i=1}^{m} \frac{dJ}{d\hat{y_i}} \frac{d\hat{y_i}}{dz_i} \frac{dz_i}{dw} = -\frac{1}{m} \sum_{i=1}^{m} (\frac{y_i(1 - \hat{y_i}) - \hat{y_i}(1 - y_i)}{\hat{y_i}(1 - \hat{y_i})})(\hat{y_i}(1 - \hat{y_i}))x_i = \frac{1}{m} \sum_{i=1}^{m} (\hat{y_i} - y_i)x_i$$

$$\frac{dJ}{db} = \sum_{i=1}^{m} \frac{dJ}{d\hat{y_i}} \frac{d\hat{y_i}}{dz_i} \frac{dz_i}{db} = -\frac{1}{m} \sum_{i=1}^{m} (\frac{y_i(1 - \hat{y_i}) - \hat{y_i}(1 - y_i)}{\hat{y_i}(1 - \hat{y_i})})(\hat{y_i}(1 - \hat{y_i}))1 = \frac{1}{m} \sum_{i=1}^{m} (\hat{y_i} - y_i)$$

Wiemy, że wartości przyjmowane przez każdą z cech na wejściu znajdują się w przedziale $[1, 10]$ i przyjmują wartości całkowite, nie musimy więc skalować danych przed zastosowaniem metody gradientowej.

Algorytm wygląda więc następująco. Zaczynamy z wektorem zerowym w oraz parametrem $b = 0$. Następnie w każdym kroku metody gradientowej, obliczamy:

$$w \leftarrow w - \alpha \frac{dJ}{dw}$$

$$b \leftarrow b - \alpha \frac{dJ}{db}$$

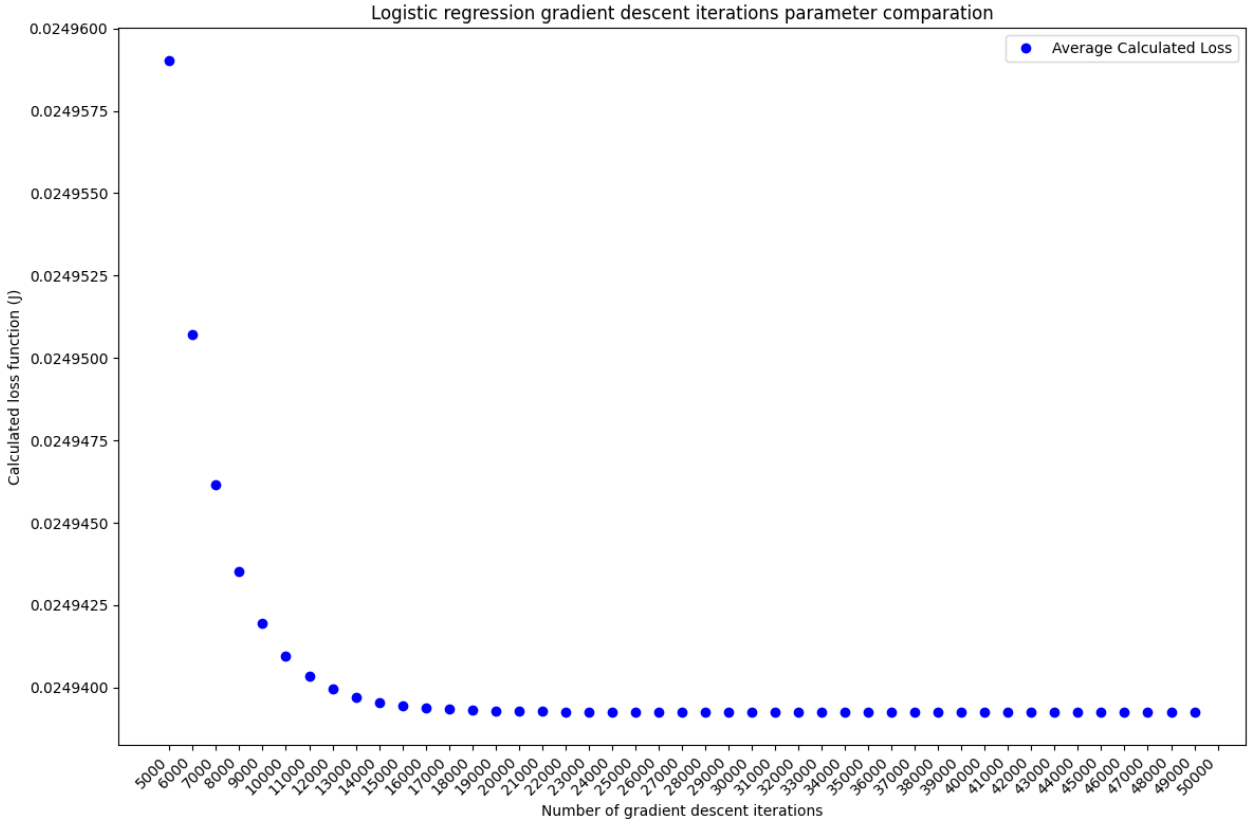
gdzie α to współczynnik uczenia się gradientu. Możemy modyfikować liczbę iteracji samej metody oraz sam współczynnik α aby uzyskiwać lepsze wyniki. Poniżej przestawiony jest fragment kodu który dla zbioru treningowego i parametrów metody gradientowej zwraca funkcję decyzyjną opartą o metodę regresji logistycznej:

```python
def GetDecisiveFunction(trainingDataset, gradientDescentIterations, gradientDescentRate):

    POSITIVE_CLASS = 4
    NEGATIVE_CLASS = 2

    m = len(trainingDataset)

    trainingDataset = np.array(trainingDataset)

    w = np.array([0.0 for _ in range(9)])
    b = 0.0

    #Helper function
    def GetPredictedY(w, b, x):
        return 1 / (1 + np.exp(-np.dot(w, x) - b))

    #Gradient descent method
    for _ in range(gradientDescentIterations):

        w_derivative = sum((GetPredictedY(w,b,trainingDataset[i][0:9])-(1 if trainingDataset[i][9] == POSITIVE_CLASS else 0))*trainingDataset[i][0:9] for i in range(m))/m
        b_derivative = sum((GetPredictedY(w,b,trainingDataset[i][0:9])-(1 if trainingDataset[i][9] == POSITIVE_CLASS else 0)) for i in range(m))/m

        w -= gradientDescentRate * w_derivative
        b -= gradientDescentRate * b_derivative

    #Takes [x_1,x_2,...,x_9], returns (predicted class, probability of that class)
    def DecisiveFunction(X):
        if len(X) != 9:
            raise ValueError("DecisiveFunction: Wrong arguments")

        positiveProbability = GetPredictedY(w, b, X)

        if positiveProbability > 0.5:
            return POSITIVE_CLASS, positiveProbability
        else:
            return NEGATIVE_CLASS, 1-positiveProbability

    return DecisiveFunction
```

Rysunek 5: Kod algorytmu regresji logistycznej

2.2 Parametry metody gradientowej

Zbadamy, jaka liczba iteracji metody gradientowej jest konieczna, aby uzyskiwała ona wyniki zbliżone do idealnych. Poniższy wykres przedstawia zależność wyników osiąganych przez dany model od liczby iteracji metody optymalizacji wzdłuż gradientu. Warto dodać, że uczenie modelu startuje od wektora zerowego w oraz $b = 0$. Zastosowaliśmy współczynnik $\alpha = 0.5$, nie okazał się on za duży, algorytm nie "skakał wokół ekstremum" pod koniec swojego działania. Wyniki zostały uśrednione na kilkudziesięciu próbach.



Rysunek 6: Porównanie skuteczności metody gradientowej

Jak widzimy, funkcja straty jest praktycznie stała od około kilkudziesięciu tysięcy iteracji. W najlepszym modelu służącym do porównania z klasyfikatorem generatywnym wykorzystamy więc parametr liczby iteracji równy 20000 przy współczynniku zbieżności względem gradientu $\alpha = 0.5$.

2.3 Regularyzacja

Do naszej funkcji straty możemy również dodać parametr regularyzacji. Wykorzystamy regularyzację l_2:

$$J_{reg}(w,b) = -\frac{1}{m}\sum_{i=1}^{m} y_i log(\hat{y}_i) + (1-y_i)log((1-\hat{y}_i)) + \frac{\lambda}{m}||w||_2^2$$

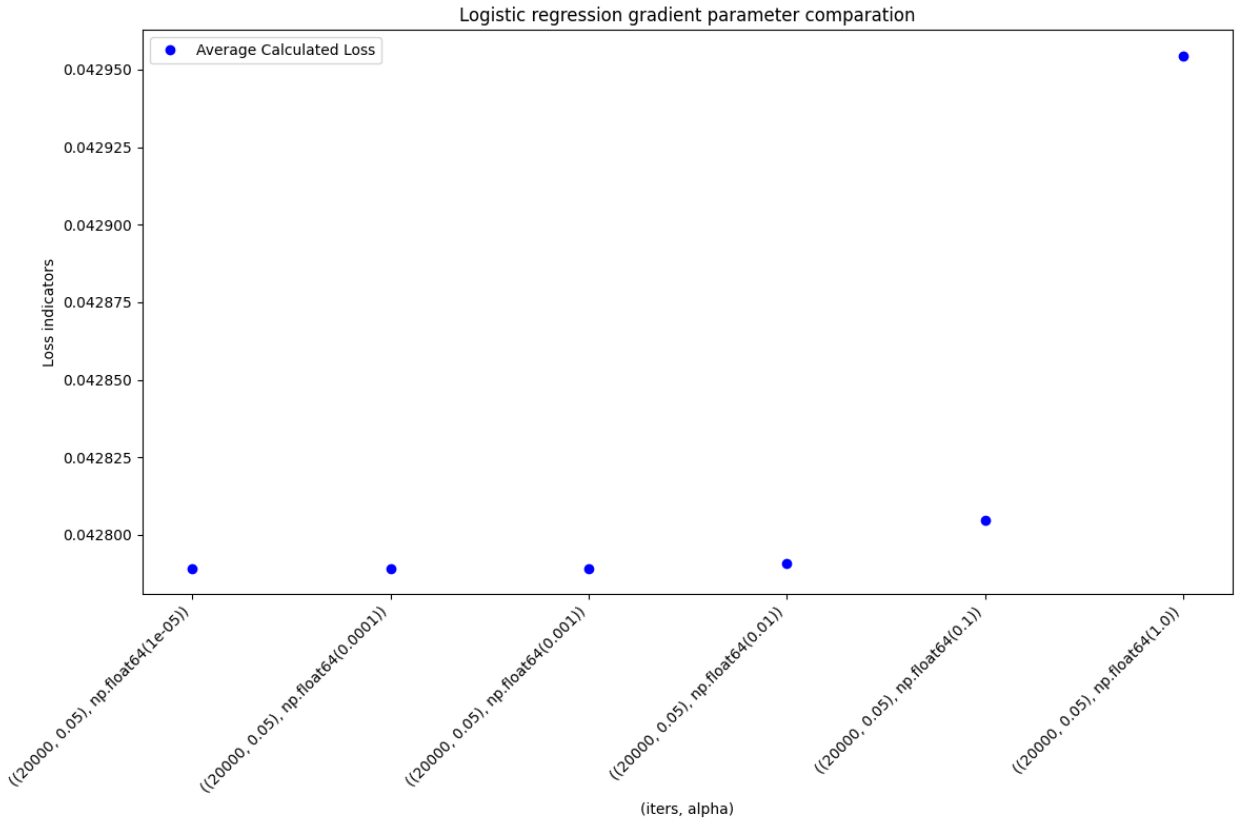
Pochodna po w zmienia się następująco:

$$\frac{dJ_{reg}}{dw} = \frac{dJ}{dw} + \frac{2\lambda}{m}w$$

Pochodna po b pozostaje bez zmian względem wersji bez regularyzacji:

$$\frac{dJ_{reg}}{db} = \frac{dJ}{db}$$

Porównamy średnie wyniki osiągane przez funkcje decyzyjne wyprodukowane przez modele w zależności od zastosowanego parametru λ. Wykres przedstawia uśrednione rezultaty uzyskane dla 10 niezależnych przebiegów algorytmu.



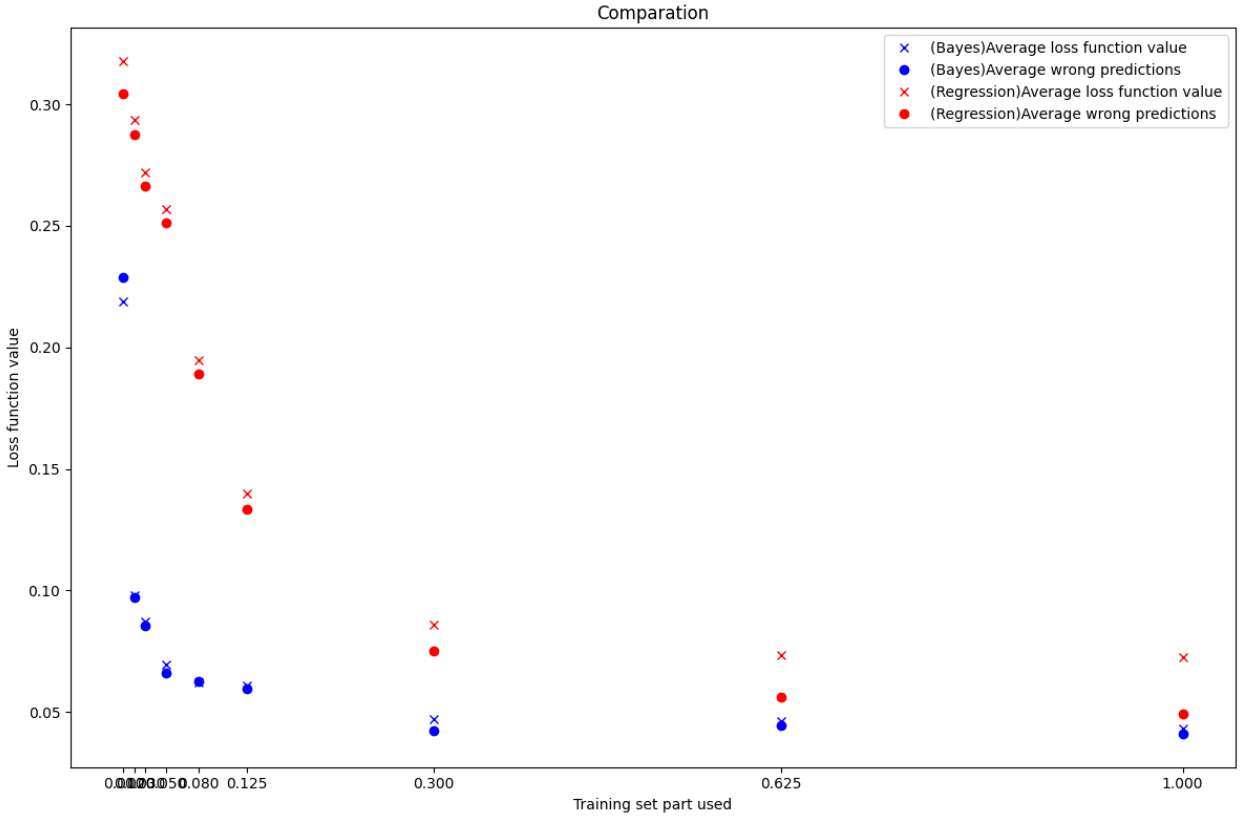
Rysunek 7: Porównanie różnych parametrów regularyzacji

Jak widać, w działaniu modelu nie zachodzi zjawisko overfittingu. Zwiększanie parametru regularyzacji prowadzi do pogorszenia skuteczności modelu, który zamiast dopasowywać się do dostarczonych danych skupia się na zmniejszaniu wektora w. Przy porównywaniu skuteczności modeli zastosujemy więc $\lambda = 0$.

3 Porównanie działania algorytmów

3.1 Krzywa uczenia

Podziały danych do zbiorów treningowych i testowych będą wspólne dla obydwu modeli. Wykres przedstawia uśrednione wyniki uzyskane przez modele w zależności od części użytego zbioru testowego. Punkty niebieskie odpowiadają naiwnemu klasyfikatorowi bayesowskiemu, czerwone odpowiadają algorytmowi regresji logistycznej. Krzyżyki opisują średnią wartość funkcji straty zdefiniowanej w opisie teoretycznym naiwnego klasyfikatora, kropki to stosunek liczby rekordów, dla których model podał nieprawidłową klasę, do liczby wszystkich rekordów w zbiorze testowym



Rysunek 8: Porównanie krzywych uczenia się dwóch opisanych modeli

Jak widać, obydwa modele uzyskały średnią skuteczność klasyfikacji na poziomie $95\% - 96\%$.

Co prawda algorytm regresji logistycznej uzyskał wyraźnie gorsze wyniki jeśli rozważać wspomnianą funkcję straty. Wynika to jednak z tego, że nie została w pełni dostosowana do porównywania tych dwóch modeli.

Tak jak wspominałem przy testowaniu klasyfikatora bayesowskiego, model ten często wskazuje prawdopodobieństwa bardzo bliskie 1, traci bardzo niewiele na dobrych przewidywaniach. Natomiast model regresji logistycznej jest nieco bardziej stonowany, przez co nawet uzyskując podobną skuteczność klasyfikacji więcej "traci" na dobrych predykcjach.

Nie jest to jednak statystyka istotna z punktu widzenia zewnętrznego działania modelu. Najważniejszą cechą jest poprawne przyporządkowywanie klas, w którym obydwa modele uzyskują podobną skuteczność.

3.2 Wnioski - porównanie z wynikami artykułu

Według autorów artykułu *On Discriminative vs. Generative Classifiers: A comparasion of logistic regression and naive Bayes*, popularnym przekonaniem jest fakt, że klasyfikatory dyskryminatywne generalnie uzyskują lepsze wyniki od generatywnych. Autorzy zwracają jednak uwagę na istotny czynnik jakim jest liczba danych treningowych i porównują zbieżność krzywej uczenia się dla tych dwóch klas klasyfikatorów.

Klasyfikatory generatywne, mimo uzyskiwania gorszych wyników dla większych zbiorów danych, są w stanie znacznie szybciej zbliżyć się do swoich optymalnych osiągów, osiągają zadowalające wyniki nawet dla niewielkich zbiorów danych. Tę tezę potwierdził nasz eksperyment. Z wykresu widać, że naiwny klasyfikator bayesowski, przykład klasyfikatora generatywnego, już dla niewielkich zbiorów treningowych (o wielkości mniej więcej $10 - 15$ elementów) osiągał dość zadowalające wyniki, które w przypadku regresji logistycznej były możliwe dopiero po zastosowaniu kilkusetelementowych zbiorów treningowych.

Wraz z dalszym wzrostem liczby danych treningowych widoczna staje się przewaga klasyfikatorów dyskryminatywnych. Przykładowo, na naszym wykresie widać ogromną różnicę dla regresji logistycznej dla frakcji 0.3 oraz 1, natomiast dla klasyfikatora bayesowskiego wyniki są bardzo podobne. Zbiór danych treningowych w naszym eksperymencie i tak był stosunkowo nieduży (liczył około 500 rekordów). Podejrzewam, że dla większej ilości dostępnych danych zobaczylibyśmy, że algorytm regresji logistycznej osiągnąłby lepsze wyniki.

Podsumowując, wybór stosowanego klasyfikatora nie jest tak oczywisty jak wynikałoby z popularnej opinii. Warto rozważać klasyfikatory generatywne w przypadku problemów o niewielkiej ilości dostępnych danych lub w których konieczne jest uzyskiwanie jak najlepszych wyników przy mocno ograniczonym czasie.