Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

Uczenie Maszynowe
(projekt)

Sprawozdanie z projektu
SK.UMA.13

Maciej Groszyk, Piotr Hondra

Warszawa, 2022

Spis treści

1. Projekt końcowy

1.1. Streszczenie projektu wstępnego

1.1.1. Opis projektu

Celem projektu jest zapoznanie się oraz implementacja metody uczenia maszynowego algorytmu generowania lasu losowego z modyfikacją zamiany klasyfikatorów z drzew decyzyjnych na naiwne klasyfikatory Bayesa.

W projekcie chcemy przeprowadzić badania eksperymentalne mające na celu ocenę ich właściwości z użyciem języka Python.

1.1.2. Plan eksperymentów

1. Implementacja klasyfikatora bayesa
2. Pórównanie własnej implementacji z gotowym rozwiązaniem z biblioteki `scikit-learn`
3. Implementacja zmodyfikowanego lasu losowego
4. Porównanie działania własnej implelmentacji lasu losowego z gotowym rozwiązaniem z biblioteki `scikit-learn`
5. Przeprowadzenie testów zaimplementowanego algorytmu dla róznych parametrów

1.2. Pełen opis funkcjonalny

1.2.1. Jak korzystać z programu?

Główny plik programu to *naive bayes.ipynb*. Jest to Jupyter Notebook. W celu przeprowadzenia symulacji programu wystarczy po kolei odpalić komórki.

1.2.2. Możliwości programu

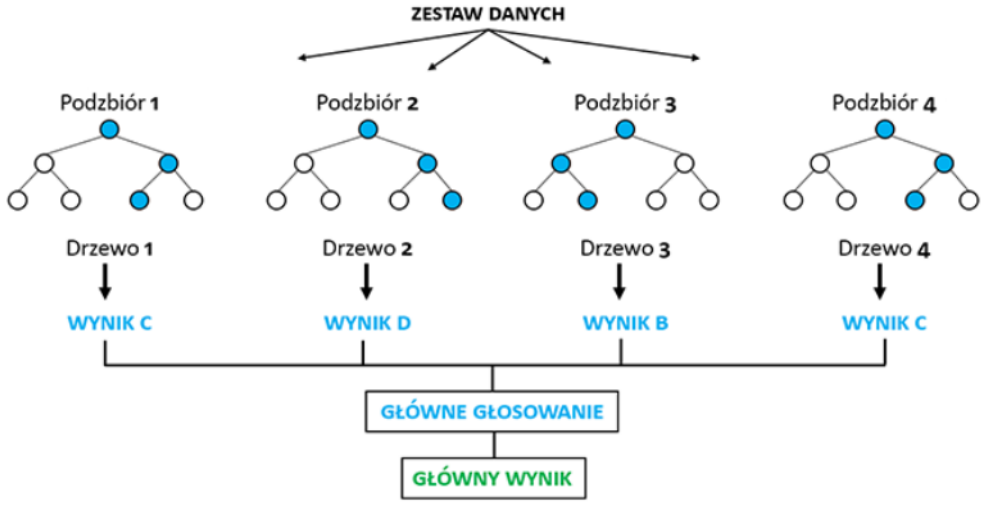
Program posiada 2 główne klasy $NaiveBayes$ (liczenie naiwnego klasyfikatora Bayesa) oraz $RandomForest$ (implementacja lasu losowego z klasyfikatorami naiwnego Bayesa). W pliku *naive bayes.ipynb* napisano wywołania metod z podanych klas w odpowiedniej kolejności. Program na podstawie danych trenujących jest w stanie przewidzieć klasę dla innych przykładów. Program w kolejności wczytuje dane, dzieli je na zbiór trenujący oraz sprawdzający, uczy wiele modeli powstałych z lasu losowego, wybiera najlepszy model, a później przewiduje klasę. Dodatkowo prorgam porównuję własną implementacje do gotowego rozwiązania z biblioteki scikit-learn.

1.3. Precyzyjny opis algorytmów oraz opis zbiorów danych

1.3.1. Las losowy

Las losowy (ang. random forest) jest to jedna z metod uczenia maszynowego wykorzystywana do problemów regresyjnych, jak i klasyfikacyjnych. Sposób działania lasu losowego polega na wytworzeniu kilku podzbiorów uczących z początkowego zbioru danych. Następnie tworzone są

drzewa dla wcześniejszych utworzonych zbiorów.W tym kroku sprawdzamy także czy dzielony zbiór nie jest zbyt mały oraz czy nie jest jednorodny. Kolejnym punktem jest wybranie zmiennych objaśniających (cech) oraz najlepszy podział dla wylosowanego podzbioru. Krok ten odbywa się na zasadzie głosowania. Jeżeli liczba drzew osiągnie zadane maksimum lub błąd w próbie testowej przestanie maleć, należy zakończyć uczenie. W przeciwnym przypadku, należy wrócić do pierwszego kroku. W ogólności: drzewa "głosują" nad rozwiązaniem, wybór następuje zwykłą większością głosów.



Rys. 1.1. Schemat lasu losowego

1.3.2. Zmodyfikowana wersję algorytmu lasu losowego

Las losowy został zmodyfikowany, tak, że zamiast drzew decyzyjnych użyto naiwnego klasyfikatora Bayesa.

1.4. Implementacja algorytmu lasu losowowego

```
from .NaiveBayesClassifier import NaiveBayes
import numpy as np


class RandomForest:
    def __init__(self, n_estimators, feature_bagging=False) -> None:
        self.clf = NaiveBayes
        self.n_estimators = n_estimators
        self.feature_bagging = feature_bagging
        self._clfs = None

    @staticmethod
    def feature_bag(X, y):
        n_rows, n_cols = X.shape
        samples_cols = np.random.choice(
            a=n_cols, size=int(np.sqrt(n_cols)), replace=True)
        return X[:, samples_cols], y
```

```
    @staticmethod
    def bag(X, y):
        n_rows, n_cols = X.shape
        samples_rows = np.random.choice(a=n_rows, size=n_rows,
                                        replace=True)
        return X[samples_rows], y[samples_rows]

    def fit(self, X, y):
        clfs = []
        for _ in range(self.n_estimators):
            clf = self.clf()
            _X, _y = self.bag(X, y)
            clf.fit(_X, _y)
            clfs.append(clf)
        self._clfs = clfs
        return self._clfs

    def predict(self, X):
        if self._clfs is None:
            raise ValueError("Fit classifier first to use predict")
        arr = []
        for clf in self._clfs:
            prediction = clf.predict(X)
            arr.append(prediction)
        arr = np.array(arr)
        result = []
        for i in range(arr.shape[1]):
            unique, counts = np.unique(arr[:, i], return_counts=True)
            result.append(unique[np.argmax(counts.shape)])

        return np.array(result)
```

Klasa `RandomForest` implementuje metody `fit(X,y)` oraz `predict(X)`, które kolejno odpowiadają za uczenie klasyfikatora oraz zwracają wartość predykcji. Jako parametry klasyfikatora możemy zmienić liczbę naiwnych klasyfikatorów Bayesa. Poza tym możemy rządać, by bagging był przeprowadzoany na atrybutach, a nie na przykładach (wierszach). W razie wywołania metody `predict()` bez wcześniejszego wywołana metody `fit(X,y)`, podniesiony zostanie wyjątek niewłaściwej inicjalizacji.

1.4.1. Naiwny klasyfikator Bayesa

Twierdzenie Bayesa jest twierdzeniem teorii prawdopodobieństwa. Wiąże prawdopodobieństwa warunkowe dwóch wpływających na siebie nawzajem zdarzeń. Wzór twierdzenia Bayesa przedstawiony został w równaniu 1.1, w którym x przedstawia wektor cech, a y to wektor etykiet dla problemu klasyfikacji.

$$P(y_k|x) = \frac{P(x|y_k)P(y_k)}{P(x)} \tag{1.1}$$

gdzie:

— $P(y_k)$ to prawdopodobieństwo wystąpienia etykiety w zbiorze
— $P(x)$ to prawdopodobieństwo wystąpienia wektora cech.

— $P(y_k|x)$ to prawdopodobieństwo wystąpienia etykiety w zbiorze na podstawie wektora zmiennych decyzyjnych
— $P(x|y_k)$ to prawdopodobieństwo wystąpienia wektora cech wiedząc, że wystąpiła etykieta.

Klasyfikator nazywamy naiwnym, ponieważ zakładamy, że wszystkie zmienne objaśniające są niezależne. Uproszczenie to pozwala w łatwy sposób implementować twierdzenie Bayesa dla rzeczywistych zbiorów danych. Równanie 1.2 przedstawia prawdopodobieństwo wystąpienia cech wiedząc, że wystąpiła etykieta.

$$P(\mathbf{x}|y_k) = P(x_1|y_k) * P(x_2|y_k) * ... * P(x_n|y_k) \tag{1.2}$$

Istnieje ryzyko wystąpienia zerowego prawdopodobieństwa dla jednej z cech. Dlatego zastosowano wygładzenie Laplace'a.

$$P(y_k|x) = \frac{P(x|y_k) * P(y_k) + \alpha}{P(x) + \alpha * K}, \tag{1.3}$$

gdzie α to współczynnik wygładzenia, najczęściej równy 1.

1.5. Implementacja algorytmu naiwnego klasyfikatora Bayesa

```
import numpy as np
from collections import defaultdict


class NaiveBayes:
    def __init__(self) -> None:
        self.prior = None
        self.likelihood = None

    def get_label_indices(self, y):
        label_indices = defaultdict(list)
        for idx, label in enumerate(y):
            label_indices[label].append(idx)
        return label_indices

    def get_prior(self, label_indices):
        prior = {label: len(indices)
                 for label, indices in label_indices.items()}
        total = sum(prior.values())
        for label in prior:
            prior[label] /= total
        return prior

    def get_likelihood(self, X, y, smoothing=1):
        label_indices = self.get_label_indices(y)
        likelihood = {}
        for label, indices in label_indices.items():
            likelihood[label] = X[indices, :].sum(axis=0)
            + smoothing
            total_count = len(indices)
            likelihood[label] = likelihood[label] / \
```

```
                (total_count + 2 * smoothing)
        return likelihood

    def get_posterior(self, X, prior, likelihood):
        posteriors = []
        for x in X:
            posterior = prior.copy()
            for label, likelihood_label in likelihood.items():
                for index, bool_value in enumerate(x):
                    posterior[label] *= likelihood_label[index] \
                    if bool_value else (
                        1 - likelihood_label[index])
            sum_posterior = sum(posterior.values())
            for label in posterior:
                if posterior[label] == float('inf'):
                    posterior[label] = 1.0
                else:
                    posterior[label] /= sum_posterior
            posteriors.append(posterior.copy())
        return posteriors

    def fit(self, X, y):
        label_indices = self.get_label_indices(y)
        self.prior = self.get_prior(label_indices)
        self.likelihood = self.get_likelihood(X, y)

    def predict(self, X):
        if self.prior is None or self.likelihood is None:
            raise ValueError("Fit classifier first to use predict")
        posterior = self.get_posterior(X, self.prior, self.likelihood)
        result = [max(prediction, key=prediction.get)
                  for prediction in posterior]
        return np.array(result)
```

Metoda *get_prior*() wylicza prawdopodobieństwo występienia wektora cech.

Metoda *get_likelihood*() wylicza prawdopodobieństwo wystąpienia wektora cech przy wystąpieniu etykiety.

Metoda *get_posterior*() wylicza prawdopodobieństwo wystąpienia etykiety w zbiorze na podstawie wektora cech.

Metoda *fit*() realizuje zadanie dopasowania.

Metoda *predict*() przewiduje otrzymaną klasę.

1.6. Zbiór danych

Do projektu użyto zbiór https://archive.ics.uci.edu/ml/datasets/Mushroom

Zestaw danych opisuje hipotetyczne próbki 23 gatunków grzybów z rodziny Agaricus i Lepiota. Gatunki są zidentyfikowane jako zdecydowanie jadalnie, trujące. Zbiór danych służy do klasyfikacji binarnej. Dokładny opis atrybutów danych podany jest w powyższym linku. Do wyboru są 22 atrybuty. Dane przedstawione są jako kategoryczne, zakodowane słownie. Pierwszy element wiersza jest przewidywaną klasą, a pozostałe są cechami.

1.7. Raport z przeprowadzonych testów oraz wnioski

Na początku testu sprawdzono poprawność działania naiwnego klasyfikatora Bayesa. Otrzymano wartość wskaźnika f1 na poziomie 0.994. Następnie porównano wynik z gotowym rozwiązaniem biblioteki Sklearn, które dało wynik około 0.94897. Powyższe testy wykonano dla współczynnika wygładzenia Laplace'a równego 1. W kolejnych testach sterowano trzema współczynnikami i obserwowano ich wpływ na dokładność rozwiązania. Możliwe parametry do zmiany: liczba wygenerowanych modeli za pomocą lasu losowego, las losowy możliwy w 2 trybach losowanie po wierszach lub kolumnach, parametr wygładzenia Laplace'a.

Na początku sprawdzono jak działa zmiana liczby wygenerowanych modeli za pomocą lasu losowego. Za pomocą jednego modelu naiwnego klasyfikatora Bayesa otrzymano wysoką poprawność, więc nie spodziewano się dużej poprawy w działaniu algorytmu. Wyniki testu ukazano w tabeli 1.7.

Numer symulacji	Liczba modeli wygenerowanych przez las losowy	Las losowy wiersz/kolumna	Wartość współczynnika wygładzenia Laplace'a	Otrzymana poprawność modelu	Czas [s]
1	1	wiersz	1	0.994	0.3
2	5	wiersz	1	0.99448	1.1
3	10	wiersz	1	0.99448	2.3
4	15	wiersz	1	0.9964	5.4
5	25	wiersz	1	0.99347	6.8
6	50	wiersz	1	0.99548	11.3
7	100	wiersz	1	0.99497	24.9
8	200	wiersz	1	0.99497	48.1

Tab. 1.1. Wyniki testu 1

Następnie przeprowadzono podobny eksperyment tylko dla lasu losowego generującego nowe modele losując na podstawie kolumn. Wyniki przedstawiono w tabeli 1.7. Zaobserwowano, że las losowy generujący nowe modele na podstawie kolumn daje minimalnie lepsze wyniki niż las losowy generujący nowe modele na podstawie wierszy.

Numer symulacji	Liczba modeli wygenerowanych przez las losowy	Las losowy wiersz/kolumna	Wartość współczynnika wygładzenia Laplace'a	Otrzymana poprawność modelu	Czas [s]
1	1	kolumna	1	0.994	0.1
2	5	kolumna	1	0.99799	1.2
3	10	kolumna	1	0.99497	2.2
4	15	kolumna	1	0.995988	3.4
5	25	kolumna	1	0.99598	5.4
6	50	kolumna	1	0.99396	11.9
7	100	kolumna	1	0.99648	22.9
8	200	kolumna	1	0.9949698	50.8

Tab. 1.2. Wyniki testu 2

W kolejnym doświadczeniu postanowiono sprawdzić jak wpływa wartość współczynnika wygładzenia Laplace'a na otrzymaną poprawność modelu. Wyniki przedstawiono w 1.7.

Numer symulacji	Liczba modeli wygenerowanych przez las losowy	Las losowy wiersz/kolumna	Wartość współczynnika wygładzenia Laplace'a	Otrzymana poprawność modelu
1	15	kolumna	0.0	0.9956
2	15	kolumna	0.001	0.9958
3	15	kolumna	0.01	0.9974
4	15	kolumna	0.1	0.996966
5	15	kolumna	0.2	0.99794
6	15	kolumna	0.5	0.996985
7	15	kolumna	1	0.99648
8	15	kolumna	5	0.994396
9	15	kolumna	20	0.994918

Tab. 1.3. Wyniki testu 3

1.7.1. Wnioski do algorytmu

Dla wybranego zbioru danych otrzymano wysokie wyniki poprawności dla jednego modelu naiwnego klasyfikatora Bayesa. Można wnioskować, że do nauki tych danych algorytm nie potrzebował koniecznie lasu losowego, jednak jego wykorzystanie z pewnością poprawiło wynik. Otrzymana poprawność modelu z wykorzystaniem lasu losowego nieznacznie poprawiła się od tego bez wykorzystania lasu losowego. Na podstawie przeprowadzonych testów parametrów algorytmu lasu losowego z klasyfikatorem naiwnego Bayesa można stwierdzić w uproszczeniu, że im więcej modeli tym poprawność algorytmu wzrasta. Wzrastająca liczba wyliczonych modeli za pomocą lasu losowego wydłuża czas działania algorytmu, lecz nie powinna pogorszyć wyniku działania algorytmu.

1.7.2. Wnioski do projektu

Implementacja algorytmu lasu losowego z klasyfikatorem naiwnego Bayesa oraz analiza algorytmu z różnymi parametrami pomogła autorom wgłębić się w zagadnienia związane z podanym algorytmem oraz ogólnie uczeniem maszynowym. Dodatkowo było to dobre ćwiczenie na poznanie dodatkowych bibliotek języka *Python* takich jak *Sklearn*, *Pandas* czy *Numpy*.

1.8. Opis wykorzystanych narzędzi, bibliotek, itp

Do projektu wykorzystaliśmy bibliotekę numpy, sklearn, pandas.