**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Домашнее задание

«Библиотека ML алгоритмов»

Выполнил: Проверил:

Студент ИУ5-34Б Преподаватель каф. ИУ5

Бурдуковский И.О. Гапанюк Ю. Е.

Подпись и дата: Подпись и дата:

Москва, 2023 г.

Задание

Реализация алгоритмов машинного обучения на Python.

Код программы

import numpy as np

from scipy.stats import multivariate\_normal

import pandas as pd

from abc import ABC

class BaseLoss(ABC):

def calc\_loss(X:np.ndarray, y:np.ndarray, w:np.ndarray) -> float:

raise NotImplementError

def calc\_grad(X:np.ndarray, y:np.ndarray, w:np.ndarray) -> np.ndarray:

raise NotImplementError

class LogisticLoss(BaseLoss):

def calc\_loss(self, X:np.ndarray, y:np.ndarray, w:np.ndarray) -> float:

Q = 0

for i in range(len(y)):

a = 1/(1+np.e\*\*(-np.dot(w,X[i])))

Q += y[i]\*np.log(a)+(1-y[i])\*np.log(1-a)

return -Q/len(y)

def calc\_grad(self, X:np.ndarray, y:np.ndarray, w:np.ndarray) -> np.ndarray:

grad = 0

for i in range(len(y)):

a = 1/(1+np.e\*\*(-np.dot(w,X[i])))

grad += X[i] \* (y[i]-a)

return -grad

class Hinge(BaseLoss):

def calc\_loss(X:np.ndarray, y:np.ndarray, w:np.ndarray) -> float:

Q = 0

for i in range(len(y)):

Q += max(0, 1 - y[i]\* np.dot(X[i], w))

return -Q/len(y)

def calc\_grad(X:np.ndarray, y:np.ndarray, w:np.ndarray) -> np.ndarray:

grad = 0

for i in range(len(y)):

if y[i]\*(np.dot(X[i], w)) > 0:

continue

else:

grad += y[i]\*X

return -grad/len(y)

class Rozenblatt(BaseLoss):

def calc\_loss(X:np.ndarray, y:np.ndarray, w:np.ndarray) -> float:

Q = 0

for i in range(len(y)):

Q += max(0, y[i]\* np.dot(X[i], w))

return -Q/len(y)

def calc\_grad(X:np.ndarray, y:np.ndarray, w:np.ndarray) -> np.ndarray:

grad = 0

for i in range(len(y)):

if y[i]\*(np.dot(X[i], w)) > 0:

continue

else:

grad += y[i]\*X

return -grad/len(y)

def PCA(X: np.ndarray, n\_components: int) -> np.ndarray:

mean = np.mean(X, axis=0)

centered\_X = X - mean

cov\_matrix = np.cov(centered\_X.T)

eigenvalues, eigenvectors = np.linalg.eig(cov\_matrix)

sorted\_indices = np.argsort(eigenvalues)[::-1]

top\_eigenvectors = eigenvectors[:, sorted\_indices[:n\_components]]

transformed\_X = np.dot(centered\_X, top\_eigenvectors)

return transformed\_X

class GaussianBayesianClassifier:

def fit(self, X, y):

self.classes = np.unique(y)

self.class\_priors = {}

self.mean\_vectors = {}

self.cov\_matrices = {}

for c in self.classes:

X\_c = X[y == c]

self.class\_priors[c] = len(X\_c) / len(X)

self.mean\_vectors[c] = np.mean(X\_c, axis=0)

self.cov\_matrices[c] = np.cov(X\_c, rowvar=False)

def predict(self, X):

predictions = []

for x in X:

posteriors = []

for c in self.classes:

prior = self.class\_priors[c]

mean = self.mean\_vectors[c]

cov = self.cov\_matrices[c]

likelihood = multivariate\_normal(mean=mean, cov=cov).pdf(x)

posterior = prior \* likelihood

posteriors.append(posterior)

predictions.append(np.argmax(posteriors))

return np.array(predictions)

class MSELoss(BaseLoss):

def calc\_loss(self, X: np.ndarray, y: np.ndarray, w: np.ndarray) -> float:

Q = ((np.linalg.norm(np.dot(X,w) - y))\*\*2)/len(y)

return Q

def calc\_grad(self, X: np.ndarray, y: np.ndarray, w: np.ndarray) -> np.ndarray:

L = np.dot(X,w) - y

Xt = np.transpose(X)

Grad = 2\*np.dot(Xt, L)/len(y)

return Grad

def gradient\_descent(w\_init: np.ndarray, X: np.ndarray, y: np.ndarray,

loss: BaseLoss, lr: float, n\_iterations: int = 100000):

W = []

for i in range(n\_iterations):

w\_init = w\_init - lr\*loss.calc\_grad(X,y, w\_init)

W.append(w\_init)

return W

class LogReg1:

def \_\_init\_\_(self, loss: BaseLoss, lr: float = 0.1) -> None:

self.loss = loss

self.lr = lr

self.w = None

self.g = None

def fit(self, X: np.ndarray, y: np.ndarray) -> 'LogReg':

X = np.asarray(X)

y = np.asarray(y)

X = np.hstack([X, np.ones([X.shape[0], 1])])

shape\_X = X.shape

self.w = np.ones(shape\_X[-1])

self.g = gradient\_descent(self.w, X, y, self.loss, lr=self.lr, n\_iterations=100000)

return self.g[-1]

def predict(self, X: np.ndarray) -> np.ndarray:

assert hasattr(self, "w"), "Log regression must be fitted first"

assert hasattr(self, "g"), "Log regression must be fitted first"

X = np.hstack([X, np.ones([X.shape[0], 1])])

y =[]

for i in range(X.shape[0]):

a = 1/(1+np.e\*\*(-np.dot(self.w,X[i])))

y.append(a)

return y

class LinearRegression1:

def \_\_init\_\_(self, loss: BaseLoss, lr: float = 0.1) -> None:

self.loss = loss

self.lr = lr

self.w = None

self.g = None

def fit(self, X: np.ndarray, y: np.ndarray) -> 'LinearRegression':

X = np.asarray(X)

y = np.asarray(y)

X = np.hstack([X, np.ones([X.shape[0], 1])])

shape\_X = X.shape

self.w = np.arange(1, shape\_X[-1] + 1)

self.g = gradient\_descent(self.w, X, y, self.loss, lr=self.lr, n\_iterations=100000)

return self.g[-1]

def predict(self, X: np.ndarray) -> np.ndarray:

# Проверяем, что регрессия обучена, то есть, что был вызван fit и в нём был установлен атрибут self.w

assert hasattr(self, "w"), "Linear regression must be fitted first"

assert hasattr(self, "g"), "Linear regression must be fitted first"

# добавляем столбец из единиц для константного признака

X = np.hstack([X, np.ones([X.shape[0], 1])])

y = np.dot(X, self.g[-1])

return y

Код для тестирования:

np.random.seed(1337)

n\_features = 2

n\_objects = 300

batch\_size = 10

num\_steps = 43

w\_true = np.random.normal(size=(n\_features, ))

X = np.random.uniform(-5, 5, (n\_objects, n\_features))

X \*= (np.arange(n\_features) \* 2 + 1)[np.newaxis, :]

y = X.dot(w\_true) + np.random.normal(0, 1, (n\_objects))

w\_init = np.random.uniform(-2, 2, (n\_features))

print(X.shape)

print(y.shape)

linregr = LinearRegression1(MSELoss(), lr=0.01)

linregr.fit(X, y)

xs = np.hstack([X, np.ones([X.shape[0], 1])])

MSELoss().calc\_loss(xs, linregr.predict(X), linregr.w)

X, y = make\_classification(

n\_samples=10000, n\_features=10, n\_informative=5, n\_redundant=5,

random\_state=42)

scl = StandardScaler()

scl.fit(X)

X = scl.transform(X)

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3)

lreg = LogReg1(LogisticLoss(), 0.1)

lreg.fit(x\_train, y\_train)

xs = np.hstack([x\_train, np.ones([x\_train.shape[0], 1])])

LogisticLoss().calc\_loss(xs, lreg.predict(x\_train), lreg.w)

X\_train = np.array([[1, 2], [2, 3], [3, 4], [4, 5], [1, 3], [2, 4]])

y\_train = np.array([0, 0, 1, 1, 0, 1])

classifier = GaussianBayesianClassifier()

classifier.fit(X\_train, y\_train)

X\_test = np.array([[1.5, 2.5], [3.5, 4.5]])

predictions = classifier.predict(X\_test)

print(predictions)