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)
```