Отчет по лабораторной работе №2
по курсу «Исскуственный интеллект»

Выполнила студентка группы 8О-304б Лаар Марина

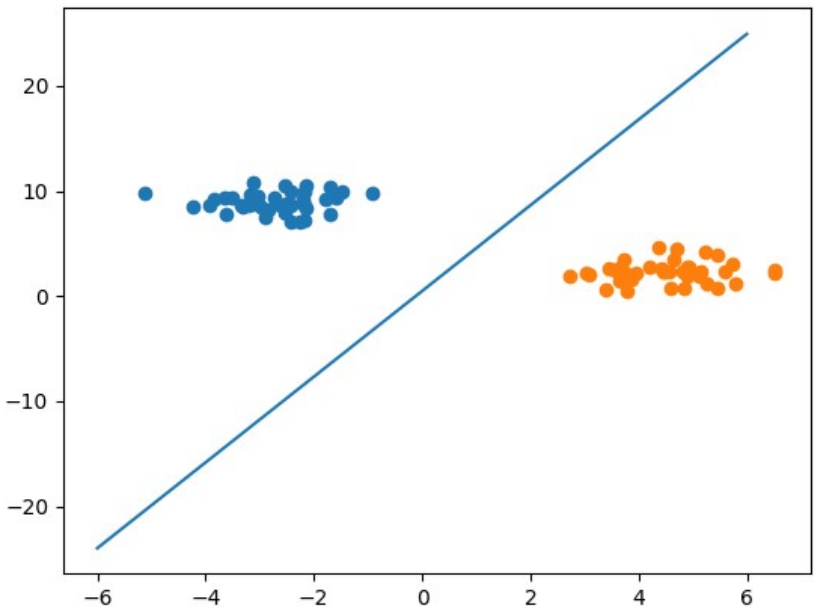
Тема: Алгоритмы машинного обучения
Задание: Реализовать на выбранном языке программирования один из алгоритмов машинного обучения. Провести предобработку набора данных из лабораторной работы №0 и проверить на нем работу алгоритма. Сравнить результаты с готовой реализацией.
Вариант: Логистическая регрессия
Ход работы:

1. Реализация алгоритма:

```
class LogReg(object):

      def __init__(self):
            self.learning_rate = 0.1
      def _sigmoid(self, z):
            return 1/(1 + np.exp(-z))

      def _loss_fuction(self, X, y):
            return (1/X.shape[0])*np.sum(np.log(1+np.exp(-y*np.dot(self.w, X.T))))

      def _loss_function_grad(self, X, y):
            M = y * np.dot(self.w, X.T)
            return -(1/X.shape[0]) * np.dot(np.exp(-M) * y / (1 + np.exp(-M)), X)

      def fit(self, X, y, iterations=10, verbose=False):
            samples = X.shape[0]
            X = np.hstack((np.ones((samples, 1)), X))
            self.w = np.zeros(shape=X.shape[1])
            for i in range(iterations):
                  dw = self._loss_function_grad(X, y)
                  # dw = dw / np.linalg.norm(dw)
                  self.w = self.w - (self.learning_rate * dw)

                  if verbose:
                        cost = self._loss_fuction(X, y)
                        print(f'Iter: {i} Loss: {cost} dW:{dw}')

            return self

      def predict_proba(self, X):
            return self._sigmoid(np.dot(self.w[1:], X.T) + self.w[0])

      def predict(self, X):
            return np.array([1 if proba > 0.5 else -1 for proba in
                                    self.predict_proba(X)], dtype=np.int8)
```

2. Для проверки работоспособности алгоритма была сгененрирована линейно-разделимая выборка с двумя числовыми признаками. Так как признака всего два, то разделяющая гиперплоскость примет вид прямой, а значит результат работы можно будет отобразить на графике:

```
def test_log_reg():
      RANDOM_SEED = 42
      CLUSTER_STD = 1

      blobs = make_blobs(centers=2,cluster_std=CLUSTER_STD,
                                                  random_state=RANDOM_SEED)
      blobs[1][blobs[1] == 0] = -1
      X_train, X_test, y_train, y_test = train_test_split(blobs[0], blobs[1],
                                        test_size=0.2, random_state=RANDOM_SEED)
      model = LogReg().fit(X_train, y_train, iterations=100)

      plt.scatter(X_train[y_train == -1, 0], X_train[y_train == -1, 1])
      plt.scatter(X_train[y_train == 1, 0], X_train[y_train == 1, 1])
      plt.plot([-6, 6], [6*model.w[1]/model.w[2] – model.w[0]/model.w[2],
                         -6*model.w[1]/model.w[2]-model.w[0]/model.w[2]])
      plt.show()
```



Как видно из графика, разделяющая гиперплоскость построилась правильно. А значит алгоритм справился с линейно-разделимой выборкой.

3. Перед тем как выполнять задачу классификации для датасета из ЛР№0 его неообходимо предобработать:

- Провести нормализацию числовых признаков
- Выполнить dummy-кодирование для категориальных признаков
- Отобразить целевую переменною на множество {-1, 1}

```
data = pd.read_csv('german_credit_data.csv', index_col=0)

      # Normalization
      data['Age'] = (data['Age'] - data['Age'].min()) /
                    (data['Age'].max() - data['Age'].min())
      data['Credit amount'] = (data['Credit amount']-data['Credit amount'].min()) /
                         (data['Credit amount'].max() - data['Credit amount'].min())
      data['Duration'] = (data['Duration'] - data['Duration'].min()) /
                          (data['Duration'].max() - data['Duration'].min())

      #dummy-code
      data = pd.get_dummies(data, columns=['Sex','Job','Housing','Saving accounts',
'Checking account', 'Purpose'])

      # map target to -1, 1
      data['Risk'] = data['Risk'].map({'bad': -1, 'good': 1})
```

4. Разделим выборку на обучающую и тестовую

```
features = list(data.columns)
features.remove('Risk')

X_train = data[features].values
y_train = data['Risk'].values
# Split to train and test sets
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train,
                                                    test_size=0.2, random_state=42)
```

5. Обучим модели и сравним точность

```
# Fit model
model1 = LogReg().fit(X_train, y_train, verbose=False, iterations=10000)
print(f'[LogReg] Точность на тестовой выборке: {accuracy_score(y_test,
                                                        model1.predict(X_test))}')

# Compare with sklearn
model2 = LogisticRegression(solver='liblinear', C=1).fit(X_train, y_train)
print(f'[Sklearn] Точность на тестовой выборке: {accuracy_score(y_test,
                                                        model2.predict(X_test))}')
```

Результат работы:

```
[LogReg] Точность на тестовой выборке: 0.76
[Sklearn] Точность на тестовой выборке: 0.75
```

Примечание: Подобрав параметр регуляризации для логистической регрессии из библиотеки sklearn можно добиться большей точности. Однако, в данном случае это ± 0.01

Выводы:

В ходе выполнения лабораторной работы мною был реализован алгоритм машинного обучения — Логистическая регрессия. Нельзя сказать, что данный алгоритм сложен для реализации, так как для него не нужно изобретать определеннных способов оптимизации — обычный градиентный спуск отлично показал себя в случае линейно-разделимой и линейно-неразделимой выборки.

Я сравнила работу алгоритма со стандартной реализации в sklearn. Результат — примерно одинаковая точность. Различие в точности обусловлено тем, что в стандартной реализации используется L2-регуляризация, которая довльно сильно может влиять на весовые коэффициенты. Так как датасет имеет небольшой размер, разница во времени работы не видна.

Таким образом, цель лабораторной работы выполнена полностью.