**Московский авиационный институт (национальный исследовательский университет) Кафедра 806**

**“Вычислительная математика и программирование”**

**Лабораторная работа №1-2 по Машинному Обучению**

Группа: М8О-308Б-18

Студент: Дубов А.В.

Преподаватель: Ахмед Самир Халид

Москва, 2021

**Задание**

**Lab 1**

Найти себе набор данных (датасет), для следующей лабораторной работы, и проанализировать его. Выявить проблемы набора данных, устранить их. Визуализировать зависимости, показать распределения некоторых признаков. Реализовать алгоритмы К ближайших соседа с использованием весов и Наивный Байесовский классификатор и сравнить с реализацией библиотеки sklearn.

**Lab 2**

Необходимо реализовать алгоритмы машинного обучения. Применить данные алгоритмы на наборы данных, подготовленных в первой лабораторной работе. Провести анализ полученных моделей, вычислить метрики классификатора. Произвести тюнинг параметров в случае необходимости. Сравнить полученные результаты с моделями реализованными в scikit-learn. Аналогично построить метрики классификации. Показать, что полученные модели не переобучились. Также необходимо сделать выводы о применимости данных моделей к вашей задаче.

**Вариант 2**

1. Логистическая регрессия
2. Дерево решений
3. Random Forest

**Алгоритмы**

**KNN**

Один из простейших алгоритмов классификации.

Алгоритм KNN можно разделить на два этапа:

• Обучение: алгоритм запоминает векторы признаков наблюдений и их метки классов. Также задаётся параметр алгоритма n, который задаёт число «соседей», которые будут использоваться при классификации.

• Классификация: предъявляется новый объект, для которого метка класса не задана. Для него определяются n ближайших (в смысле некоторой метрики) предварительно классифицированных наблюдений. Затем выбирается класс, которому принадлежит большинство из n ближайших примеров-соседей, и к этому же классу относится классифицируемый объект.

В моей реализации используется стандартный алгоритм с использованием Евклидовой нормы.

Выбранная мною метрика – Евклидова норма

**Наивный Байесовский классификатор**

* основе лежит формула условной вероятности Байеса. Наивным алгоритм делает предположение о независимости признаков. Данное предположение позволяет рассчитать функции правдоподобия классов

**Логистическая регрессия**

Данная модель, аналогично линейной регрессии, рассчитывает взвешенную сумму, однако определяет принадлежность объекта классу по значению сигмоидной функции:



Если результат больше 0,5, мы можем классифицировать результат как 1 (или положительный), а если он меньше 0,5, мы можем классифицировать его как 0 (или отрицательный).

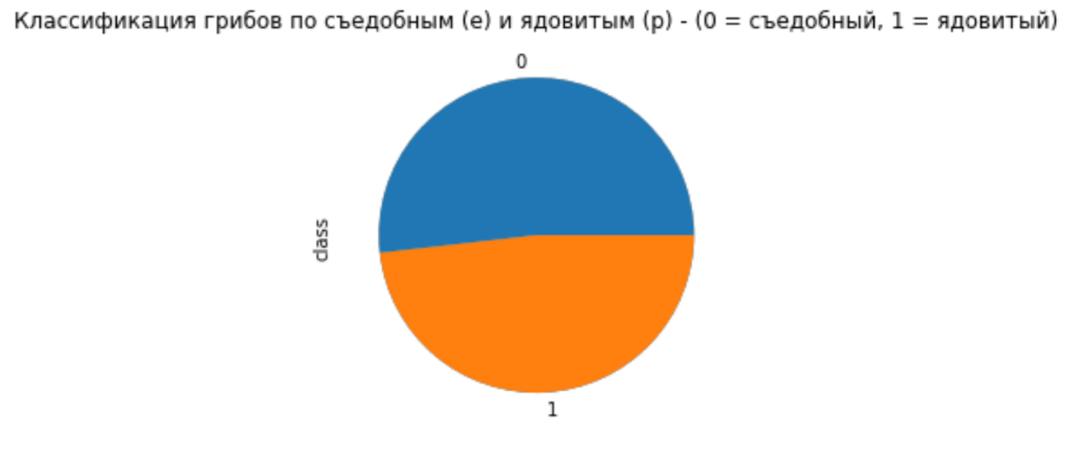
**Random Forest**

Это метод, использующий ансамбль деревьев решений, созданных на случайно разделенном датасете. Набор таких деревьевклассификаторов образует лес. Каждое отдельное дерево решений генерируется с использованием метрик отбора показателей, таких как критерий прироста информации, отношение прироста и индекс Джини для каждого признака. Случайный лес считается высокоточным и надежным методом, поскольку в процессе прогнозирования

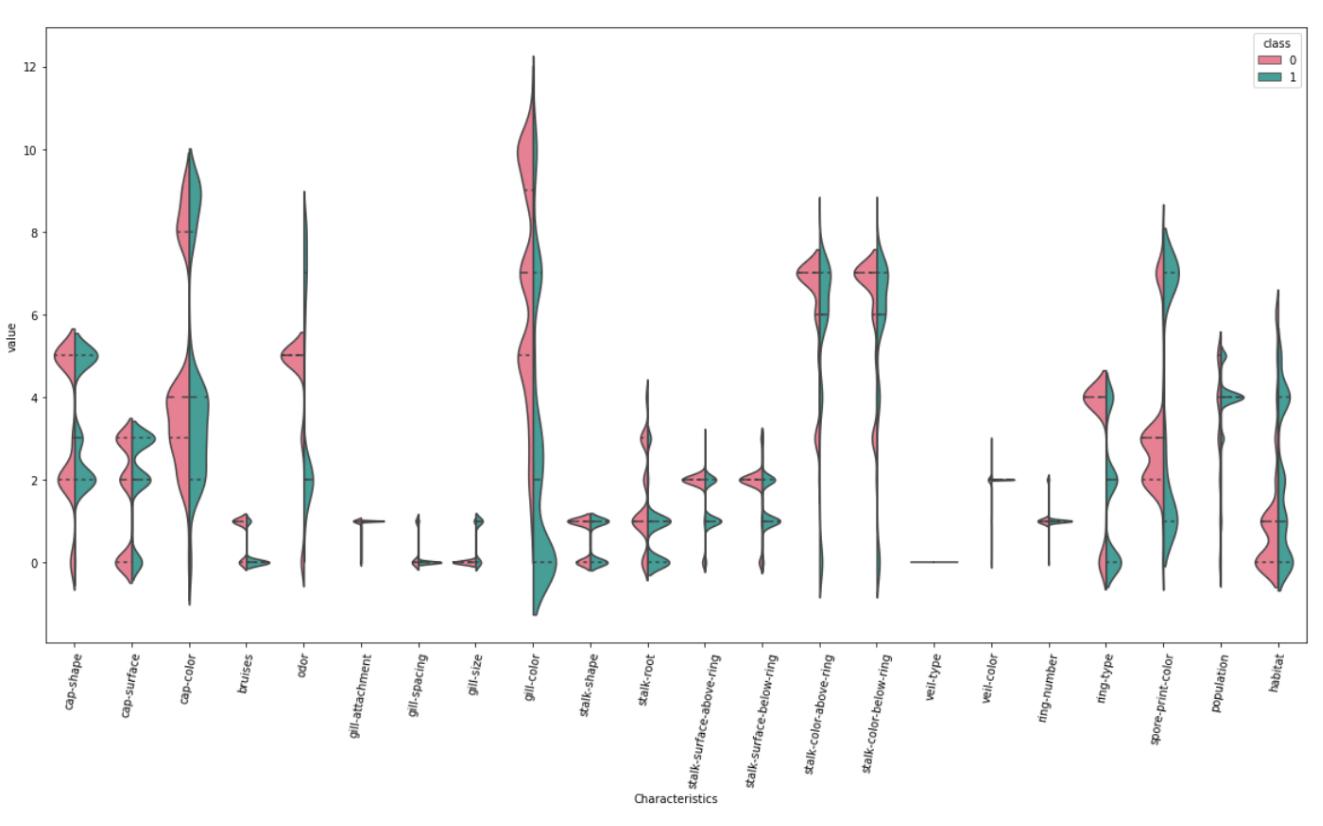
участвует множество деревьев решений

**Используемые метрики**

Для оценки качества классификации использованы метрика accuracy, так как в обеих задачах отсутсвует дисбаланс. Валидирование проводилось посредством кросс-валидации.

Мною был выбран mushroom датасет. Данные были проверены на баланс по классу:

Из построенной зависимости видно, что класс не нуждается в балансировке.

Распределение по признакам:

**Результаты**

Все алгоритмы оказались достаточно медленнее в собственной реализации по сравнению с алгоритмами библиотеки scikit-learn, поскольку в моих реализациях нет оптимизации.



In [15]:

In [16]:

**from** sklearn.neighbors **import** KNeighborsClassifier **from** sklearn.linear\_model **import** LogisticRegression **from** sklearn.naive\_bayes **import** GaussianNB

**from** sklearn.ensemble **import** RandomForestClassifier **from** sklearn.tree **import** DecisionTreeClassifier **import** ML **as** algo

**from** sklearn.model\_selection **import** KFold

**from** sklearn.metrics **import** accuracy\_score

**from** sklearn.metrics **import** precision\_score

**from** sklearn.metrics **import** recall\_score

**import** warnings

warnings**.**filterwarnings('ignore')



**def** cv(model, X, y, k\_folds**=**5):

kf **=** KFold(n\_splits**=**k\_folds, random\_state**=**16, shuffle**=True**)

scores **=** np**.**zeros(k\_folds)

precisions **=** np**.**zeros(k\_folds)

recalls **=** np**.**zeros(k\_folds)

**for** i, (train\_index, val\_index) **in** enumerate(kf**.**split(X, y)):

X\_train, y\_train **=** X**.**loc[train\_index]**.**to\_numpy(), y**.**loc[train\_index

X\_val, y\_val **=** X**.**loc[val\_index]**.**to\_numpy(), y**.**loc[val\_index]**.**to\_num

model**.**fit(X\_train, y\_train)

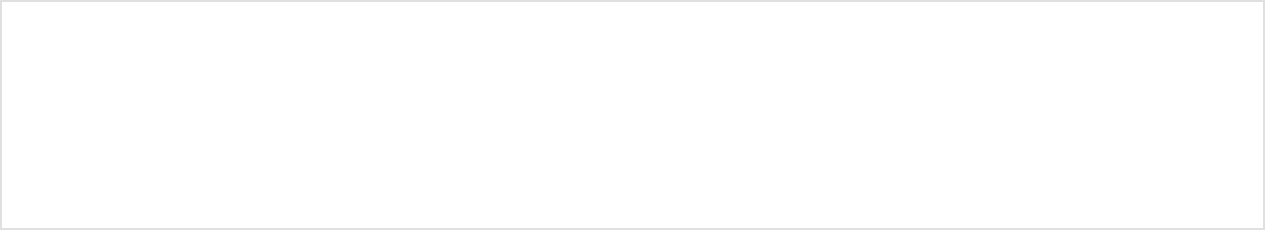
y\_pred **=** model**.**predict(X\_val)

scores[i] **=** accuracy\_score(y\_val, y\_pred)

precisions[i] **=** precision\_score(y\_val, y\_pred)

recalls[i] **=** recall\_score(y\_val, y\_pred)

**return** (scores, precisions, recalls)



In [17]:

In [18]:

**%%time**

model **=** LogisticRegression()

values **=** cv(model, X, y)

print("Accuracy: ", values[0]**.**mean())

print("Precision: ", values[1]**.**mean())

print("Recall: ", values[2]**.**mean())

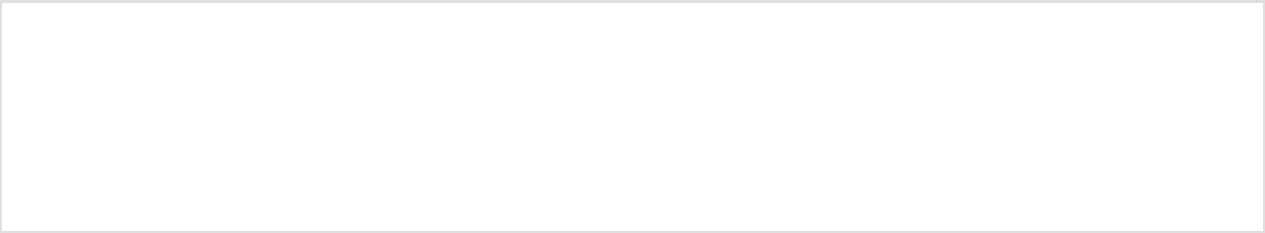
Accuracy: 0.9497783251231526

Precision: 0.9546315700623771

Recall: 0.9404389431928346

CPU times: user 1.09 s, sys: 94 ms, total: 1.18 s

Wall time: 628 ms



**%%time**

model **=** algo**.**LR()

values **=** cv(model, X, y)

print("Accuracy: ", values[0]**.**mean())

print("Precision: ", values[1]**.**mean())

print("Recall: ", values[2]**.**mean())

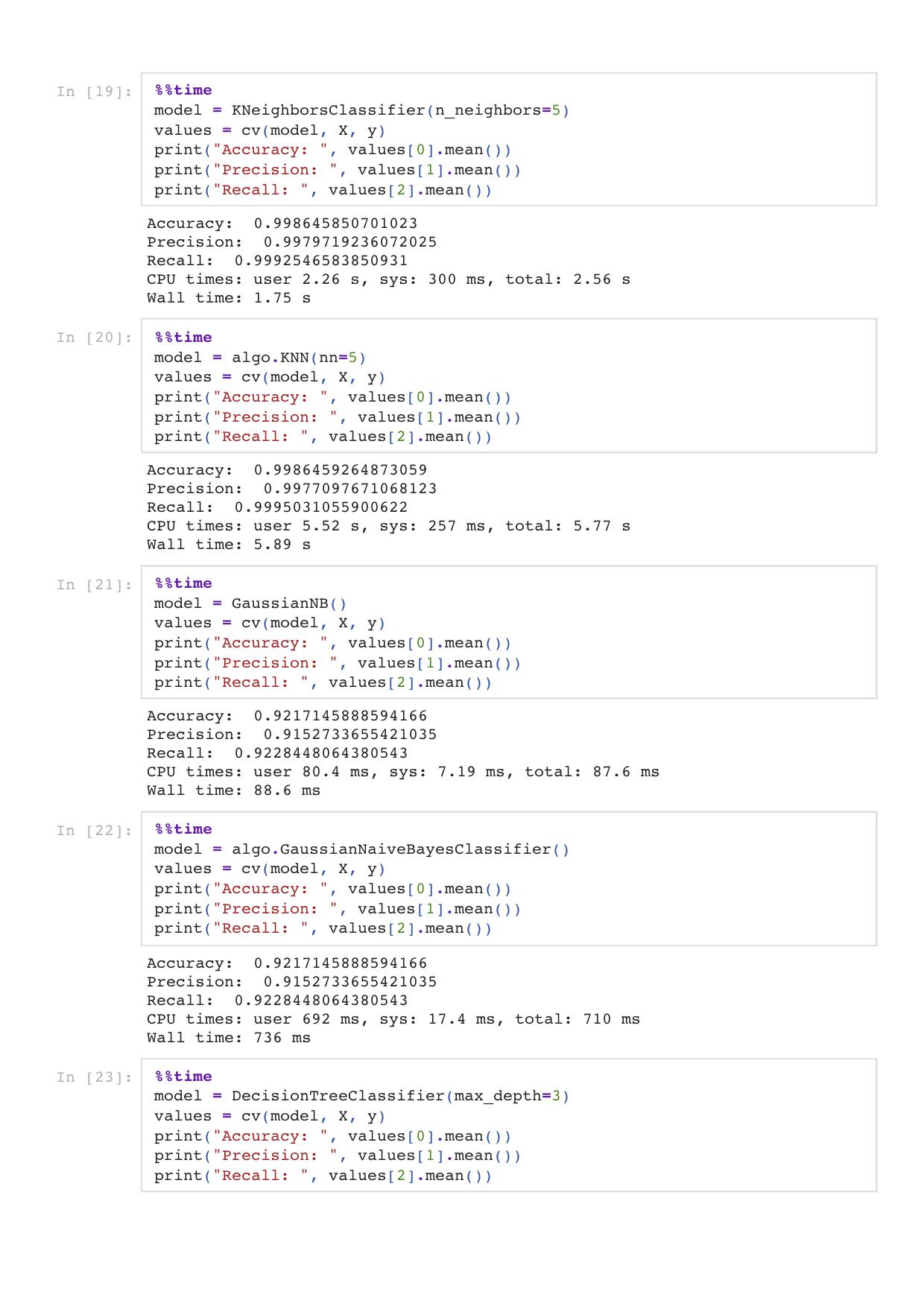
Accuracy: 0.9164206138688897

Precision: 0.9455713392470543

Recall: 0.8772857046305106

CPU times: user 8.35 s, sys: 519 ms, total: 8.87 s

Wall time: 4.56 s



|  |
| --- |
|  |

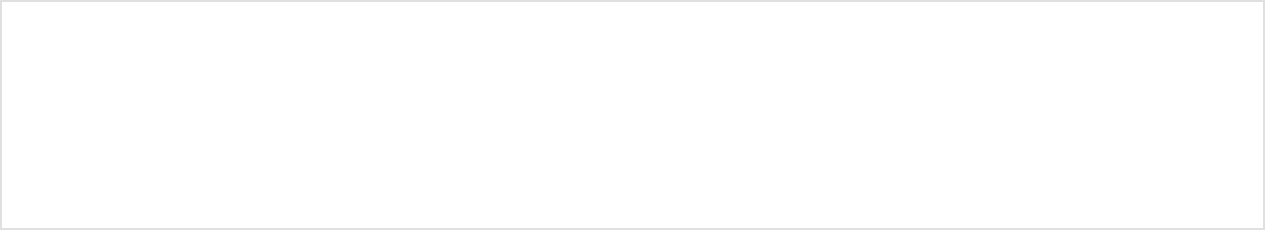
Accuracy: 0.9586414550966275

Precision: 0.9433898271980151

Recall: 0.9722973135250184

CPU times: user 84.1 ms, sys: 5.58 ms, total: 89.7 ms

Wall time: 93.7 ms



In [24]: **%%time**

model **=** algo**.**DTC()

values **=** cv(model, X, y)

print("Accuracy: ", values[0]**.**mean())

print("Precision: ", values[1]**.**mean())

print("Recall: ", values[2]**.**mean())

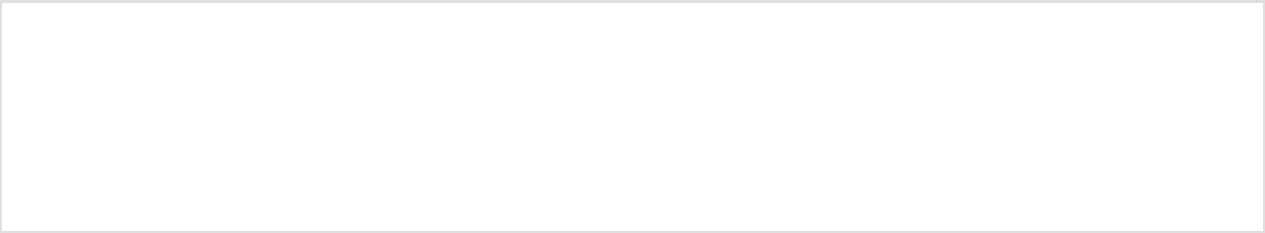
Accuracy: 0.7902507010231148

Precision: 0.8343044863880655

Recall: 0.704640176834282

CPU times: user 4.63 s, sys: 47.4 ms, total: 4.68 s

Wall time: 4.72 s



In [25]: **%%time**

model **=** algo**.**RFC()

values **=** cv(model, X, y)

print("Accuracy: ", values[0]**.**mean())

print("Precision: ", values[1]**.**mean())

print("Recall: ", values[2]**.**mean())

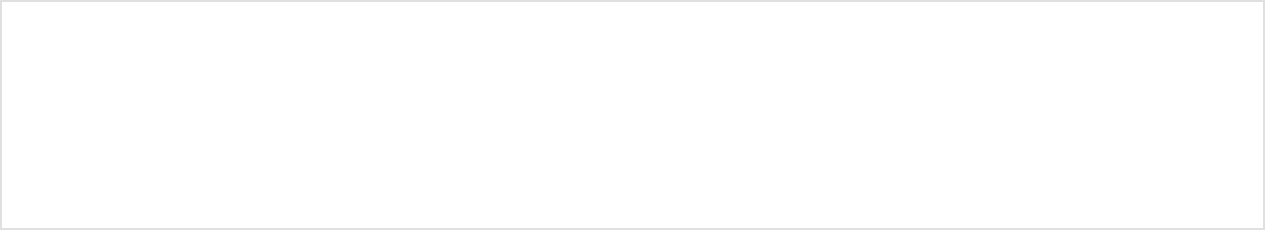
Accuracy: 0.8878710875331566

Precision: 0.8763113498415525

Recall: 0.8741603085014997

CPU times: user 350 ms, sys: 5.82 ms, total: 355 ms

Wall time: 357 ms



In [26]: **%%time**

model **=** RandomForestClassifier()

values **=** cv(model, X, y)

print("Accuracy: ", values[0]**.**mean())

print("Precision: ", values[1]**.**mean())

print("Recall: ", values[2]**.**mean())

Accuracy: 1.0

Precision: 1.0

Recall: 1.0

CPU times: user 1.69 s, sys: 17.8 ms, total: 1.71 s

Wall time: 1.73 s

При выбранных параметрах алгоритмов видно, что модель не переобучалась, поскольку разница в точности классификации на обучающей и тестовой выборках в основном достаточно мала.

Что касается полученной точности – по результатам Байесовского классификатора можно сделать предположение, что дата сет не совсем подходит для этого алгоритма. Поскольку реализация была улучшена и очень близка к библиотечной.

**Выводы**

При “ручной” реализации алгоритмов машинного обучения нужно использовать максимально эффективные алгоритмы, чтобы достичь таких же высоких результатов, какие предоставляют нам библиотечные реализации.