МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

**«Московский технический университет связи и информатики»**

Кафедра «Информационная безопасность»

Проект по дисциплине

**«Проектный практикум»**

Вариант 3.12.1

**Детектирование фишинговых писем при помощи ML**

Выполнили: студенты группы БИБ2302

Парфенова А.С., Каюрина К.А.,

Макарова М.М., Дудоркин Д.Д.

Проверил:

к.т.н., доц. Ванюшина А.В.

Москва 2024

**Содержание**

[Задание №12 для выполнения практического задания по дисциплине «Проектный практикум» 3](#__RefHeading___1)

[Описание проекта «Детектирование фишинговых писем при помощи ML» 5](#__RefHeading___2)

[Цель проекта 5](#__RefHeading___3)

[Задачи проекта 5](#__RefHeading___4)

[Роли 6](#__RefHeading___5)

[Этапы реализации проекта: 8](#__RefHeading___6)

[1. Методы детектирования фишинговых писем и анализ их эффективности 8](#__RefHeading___7)

[2. Загрузка набора данных и его первичная обработка 10](#__RefHeading___8)

[3. Векторизация набора данных 11](#__RefHeading___9)

[4. Разделение набора данных на тестовую и обучающую выборки 12](#__RefHeading___10)

[5. Обучение и классификация модели Naive Bayes 13](#__RefHeading___11)

[Результаты 15](#__RefHeading___12)

[Распределение реальных и предсказанных результатов 15](#__RefHeading___13)

[Матрица ошибок 16](#__RefHeading___14)

[Сравнение Naive Bayes с другими методами классификации 17](#__RefHeading___15)

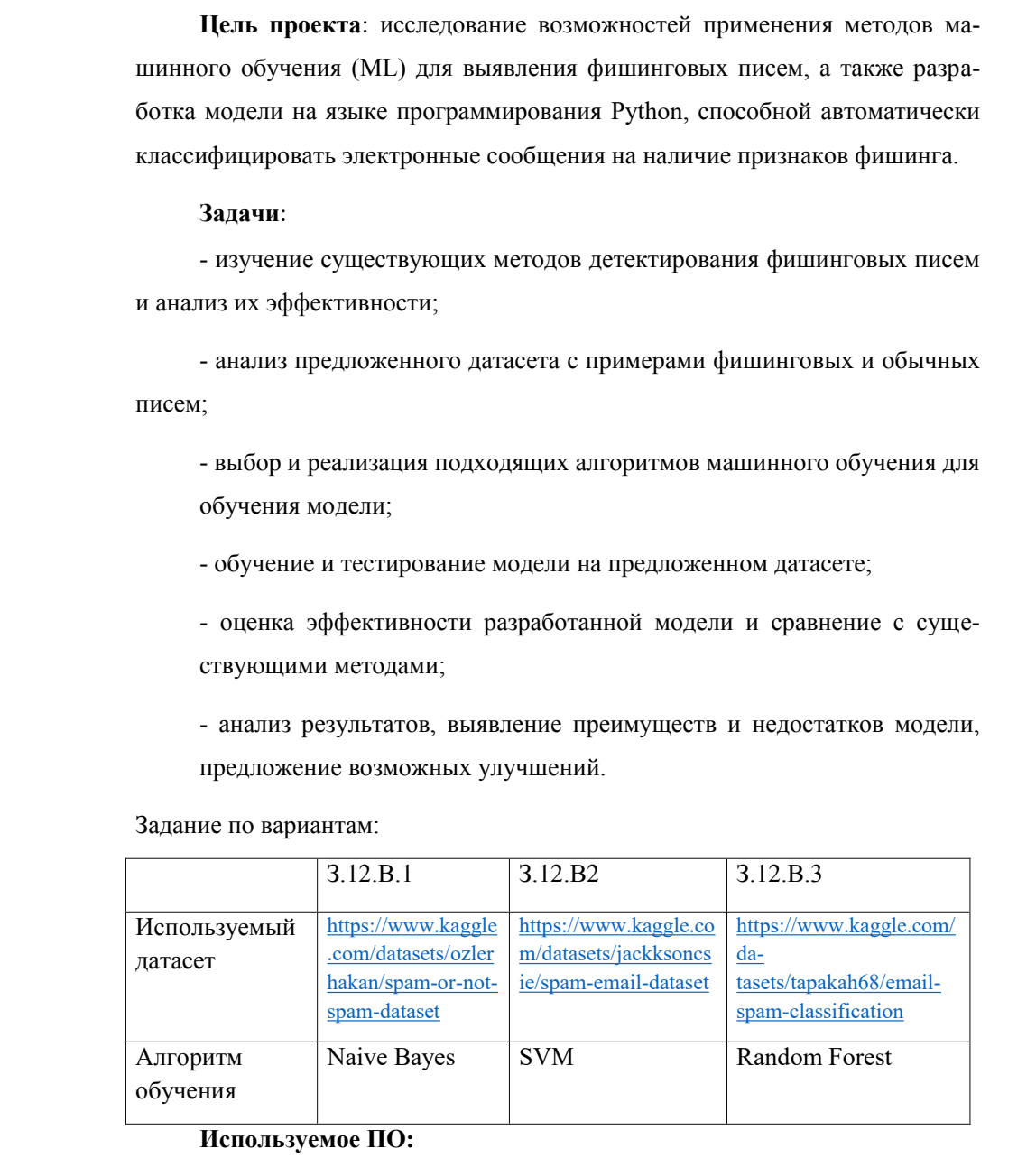
[Итог 18](#__RefHeading___16)

[Список использованных источников 19](#__RefHeading___17)

[Приложение 20](#__RefHeading___18)

# Задание №12 для выполнения практического задания по дисциплине «Проектный практикум»

Детектирование фишинговых писем при помощи ML



# Описание проекта **«Детектирование фишинговых писем при помощи ML»**

Фишинговые атаки через электронную почту остаются одним из наиболее распространенных методов кибермошенничества и компрометации конфиденциальной информации. Они представляют серьезную угрозу как для частных лиц, так и для предприятий, могут привести к утечке чувствительных данных, финансовым потерям и даже к краже личной информации.

В этом контексте разработка эффективных методов детектирования фишинговых писем становится все более важной задачей. Использование методов машинного обучения (ML) для автоматизации процесса обнаружения фишинговых писем открывает новые перспективы в борьбе с этим видом киберугроз.

Этот проект имеет ценность для защиты от фишинговых атак и обеспечения безопасности пользователей в онлайн среде, а также демонстрирует потенциал машинного обучения в решении сложных задач кибербезопасности.

## Цель проекта

Исследование возможностей применения методов машинного обучения (ML) для выявления фишинговых писем, а также разработка модели на языке программирования Python, способной автоматически классифицировать электронные сообщения на наличие признаков фишинга.

## Задачи проекта

* Изучение существующих методов детектирования фишинговых писем и анализ их эффективности;
* Анализ предложенного набора данных с примерами фишинговых и обычных писем;
* Выбор и реализация подходящих алгоритмов машинного обучения для обучения модели;
* Обучение и тестирование модели на предложенном наборе данных;
* Оценка эффективности разработанной модели и сравнение с существующими методами;
* Анализ результатов, выявление преимуществ и недостатков модели, предложение возможных улучшений.

## Роли

**Дудоркин Д.А.** – Отвечает за следующие этапы работы:

* Анализ результатов тестирования и сравнение с результатами других методов детектирования фишинга.
* Представление графиков и диаграмм для наглядной демонстрации эффективности модели.
* Выбор алгоритмов машинного обучения.
* Проведение тестов для оценки производительности модели на тестовой выборке.

**Парфенова А.С.** – Отвечает за следующие этапы работы:

* Выбор метрик эффективности: определение критериев оценки модели.
* Разделение набора данных на обучающую и тестовую выборки для проведения обучения и тестирования модели.
* Написание кода для создания и обучения модели машинного обучения.
* Проведение тестов для оценки производительности модели на тестовой выборке.
* Представление графиков и диаграмм для наглядной демонстрации эффективности модели.

**Каюрина К.А.** – Отвечает за следующие этапы работы:

* Анализ результатов тестирования и сравнение с результатами других методов детектирования фишинга.
* Проверка стабильности и надежности модели при изменениях входных данных. Определение сильных и слабых сторон разработанной модели.
* Выбор алгоритмов машинного обучения.
* Выбор метрик эффективности: определение критериев оценки модели.

**Макарова М.М.** – Отвечает за следующие этапы работы:

* Изучение научных статей и публикаций по существующим методам детектирования фишинговых писем.
* Анализ набора данных, включающего примеры фишинговых и обычных писем. Очистка и структуризация данных для последующего использования в обучении модели.
* Выбор метрик эффективности: определение критериев оценки модели.
* Выбор алгоритмов машинного обучения.

# Этапы реализации проекта:

### Методы детектирования фишинговых писем и анализ их эффективности

Детектирования фишинговых писем делятся на три типа методов: базовые методы, методы на основе контента и методы на основе машинного обучения.

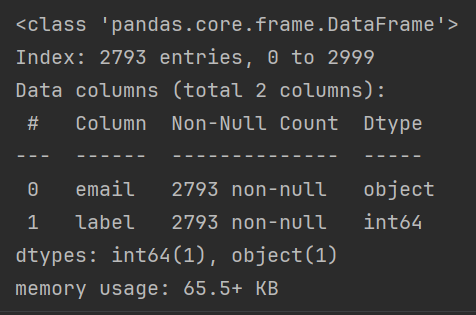
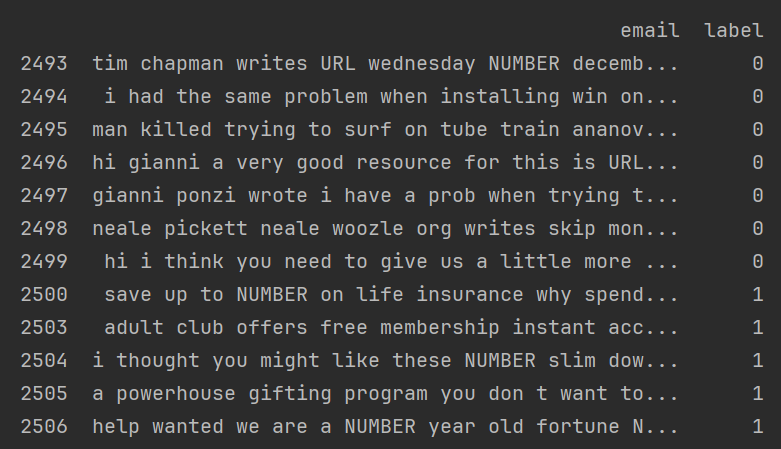
К базовым методам относятся списки заблокированных отправителей и доменов и белые списки. Метод списков заблокированных отправителей и доменов использует предопределённых списков заблокированных или так называемых черных адресов и доменов для фильтрации подозрительных писем. Преимущество данного метода является его простота реализации и мгновенный результат, а его недостаток – новые и не внесенные в черный список источники фишинга. Метод белые списки использует списки доверенных или белых адресов и доменов для пропуска легитимных писем. Преимущество метода заключается в его высокой точности для писем из надежных источников, а недостаток – требование частого обновления и ограничение диапазона допустимых отправителей.

К методам на основе контента относятся анализ ключевых слов и фраз и анализ структуры URL. Метод анализа ключевых слов и фраз заключается в поиске характерных для фишинга ключевых слов и фраз в тексте письма. Преимуществом такого метода является его простота и высокая скорость анализа, а недостатком метода - высокая вероятность ложных срабатываний, так как злоумышленники могут избегать использования определенных слов. Метод анализа структуры URL заключается в проверке ссылок в письме на предмет подозрительных структур. Преимущество метода заключается в эффективности против ссылок, перенаправляющих на фишинговые сайты или страницы, а его недостаток – злоумышленники тщательнее маскируют ссылки.

К методам на основе машинного обучения относятся байесовские классификаторы, логистическая регрессия, дерево решений, SVM и другие алгоритмы. Данные методы будут сравниваться ниже по метрика их эффективности по набору данных из предложенного к заданию.

### **Загрузка набора данных и его первичная обработка**

После загрузки имеющегося набора данные были просмотрены и убраны дублирующиеся и пустые сообщения, при помощи следующих библиотек: pandas (используется для загрузки и обработки данных), matplotlib (используется для визуализации данных в графики и матрицы), scikit-learn (используется для обучения модели и оценки их эффективности), seaborn (используется для добавления тепловых карт, которые отображают матрицы ошибок) и numpy (используется для создания массивов и выполнения математических операция). В результате чего мы получили 2793 строки (см. рисунок 1), содержащие текст сообщения и метку-идентификатор (см. рисунок 2), которое поясняло является ли сообщение фишинговым (0 – обычное, 1 – фишинговое письмо).

  
Рисунок 1 – Вывод количества оставшихся писем после удаления дубликатов и пустых сообщений  
  
Рисунок 2 – Вывод части отредактированного набора данных

### **Векторизация набора данных**

Для получения атрибутов, на основе которых осуществляется обучение и классификация, был использован векторизатор TfidVectorizer. TfidVectorizer позволяет преобразовать наборы текстовых сообщений в таблицу частот слов в сообщениях, также дополнительно было добавлено удаление стоп-слова при помощи параметра stop\_words=’english’ и приведение слова к нижнему регистру при помощи параметра lowercase=True. Вывод результата векторизатора TfidVectorizer можно посмотреть на рисунке 3.

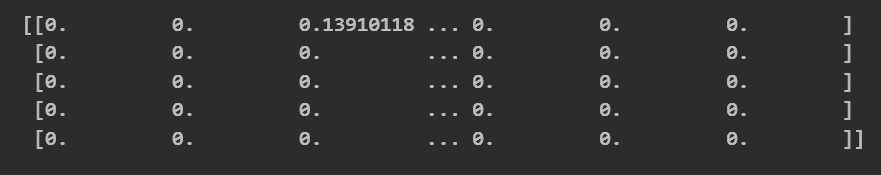


Рисунок 3 – TF-IDF матрица для каждого слова в письмах

### **Разделение набора данных на тестовую и обучающую выборки**

Чтобы проводить дальнейшие работы с набором данных, требуется разделить имеющийся набор данных на обучающую и тестовую выборки. Чтобы определить какое процентное соотношение лучше сказывается на обучении требуемого классификатора, нужно их сравнить в следующих процентных соотношениях: 80% на 20%, 50% на 50% и 20% на 80%. Затем в последствии было выбрано наилучшее разбиение. Для разделения набора данных на обучающую и тестовую выборки была использована функция train\_test\_split, которая находится в библиотеке scikit-learn и которая содержит в внутри себя следующие обязательные параметры: «X» – значения TF-IDF матрицы, «y» – метки-идентификаторы и «test\_size» – процент данных отправляемых на тестовую часть выборки (необязательный параметр).

### Обучение и классификация модели Naive Bayes

Для проведения классификации необходимо сначала инициализировать модель по следующим методам классификации эффективности: accuracy (точность, где сумма правильно классифицированных фишинговых и обычных наблюдений, деленная на общее число наблюдений), precision\_1 (точность, где считается из фишинговых писем, точно фишинговые), precision\_0 (точность, где считается из обычных писем, точно обычные), recall\_1 (полнота, которая считает сколько фишинговых писем получилось обнаружить), recall\_0 (полнота, которая считает сколько обычных писем получилось обнаружить), F1\_1 (процент сочетания точности и полноты фишинговых писем) и F1\_0 (процент сочетания точности и полноты обычных писем). Однако перед этим сначала требуется обучить модель с помощью метода fit() из библиотеки scikit-learn, который принимает следующие значения: матрица признаков (X) и их метки-идентификаторы (y).

Также было проведено сравнение и процентного соотношения обучающей и тестовой части (см. рис. 4) из 3-го пункта по метрикам эффективности и лучшие показатели метрик эффективности имеют следующее процентное соотношение: accuracy – 93% у «20% и 80% на тестировании» (и 92% у «50% на тестировании»), precision\_1 – 74% у «20% на тестировании» (57% у «50% на тестировании» и 58% у «80% на тестировании»), precision\_0 – 99% у «80% на тестировании» (96% у «20% на тестировании» и 97% у «50% на тестировании»), recall\_1 – 92% у «80% на тестировании» ( 78% «20% и 50% на тестировании»), recall\_0 – 96% «20% на тестировании» (94% у «50% и 80% на тестировании»), F1\_1 – 76% у «20% на тестировании» (66% «50% на тестировании» и 71% у «80% на тестировании») и F1\_0 – 96% у всех процентов на тестировании. В результате лучшее процентное соотношение 20% на тестирование и 80% на обучение.

Поэтому через таблицу метрик эффективности было проведено сравнение методов алгоритма Naive Bayes: мультиномиальный, Бернулли и Гауссовский методы (см. рис. 5). Сравнив три метода между собой лучшие метрики эффективности получили следующие методы: accuracy – 93% «Гаусовского метода» (и по 87% у других методах), precision\_1 – 74% у «Гаусовского метода» (12% у «мультиноминального» и 24% у «Бернулли»), precision\_0 – 100% у «мультиноминального метода» (97% у «Бернулли» и 96% у «Гаусовского»), recall\_1 – 100% у «мультиноминального» метода ( 59% у «Бернулли» и 78% у «Гаусовского»), recall\_0 – 96% у «Гаусовского метода» (87% у «мультиноминального» и 88% у «Бернулли»), F1\_1 – 76% у «Гаусовского метода» (22% у «мультиноминального» и 34% у «Бернулли») и F1\_0 – 96% у «Гаусовского метода» (по 93% у других методов). В связи с результатами таблицы Гауссовский метод показал себя наилучшим образом.

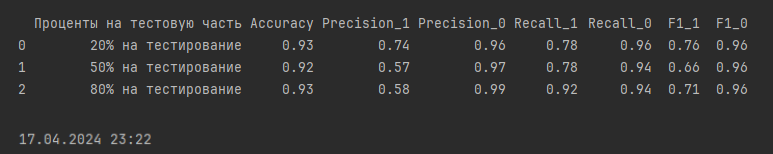


Рисунок 4 – Таблица метрик производительности с разными разделениями частей тестирования и обучения программы

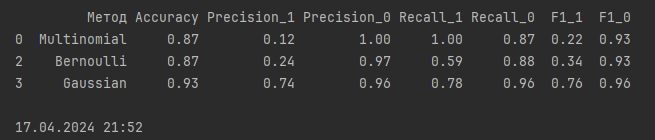


Рисунок 5 – Таблица метрик производительности с тремя методами

# Результаты

## Распределение реальных и предсказанных результатов

На рисунке 6 видно, что Naive Bayes ошибся на 4 сообщения, которые он посчитал как обычные сообщения. Можно сказать, что результат выборки имеет небольшой дисбаланс между классами. График по распределению реальных и предсказанных результатов был построен благодаря библиотеки matplotlib, в основе которго лежат следующие функции: plt.subplots() – создает фигуру и оси для построения графика, ax.bar() – создает столбчатую диаграмму для заданных данных, ax.text() – добавляет текстовые метки к столбцам на графике, ax.set\_xticks() – устанавливает позиции меток по оси X, ax.set\_xticklabels() – устанавливает подписи для меток по оси X, ax.set\_title() – устанавливает заголовок графика, ax.legend() – добавляет легенду к графику, plt.show() – отображает созданный график.

## 

Рисунок 6 – График распределения Реальных и Предсказанных сообщений

## Матрица ошибок

На рисунке 7 мы видим, что 462 обычных писем были предсказаны верно (TN), 59 фишинговых писем были предсказаны верно (TP), 17 фишинговых писем были не верно предсказаны (FP), и 21 обычное письмо было не верно предсказано (FN). Матрица была построена при помощи функции confusion\_matrix благодаря следующих импортированных библиотек: confusion\_matrix, train\_test\_split, и GaussianNB.

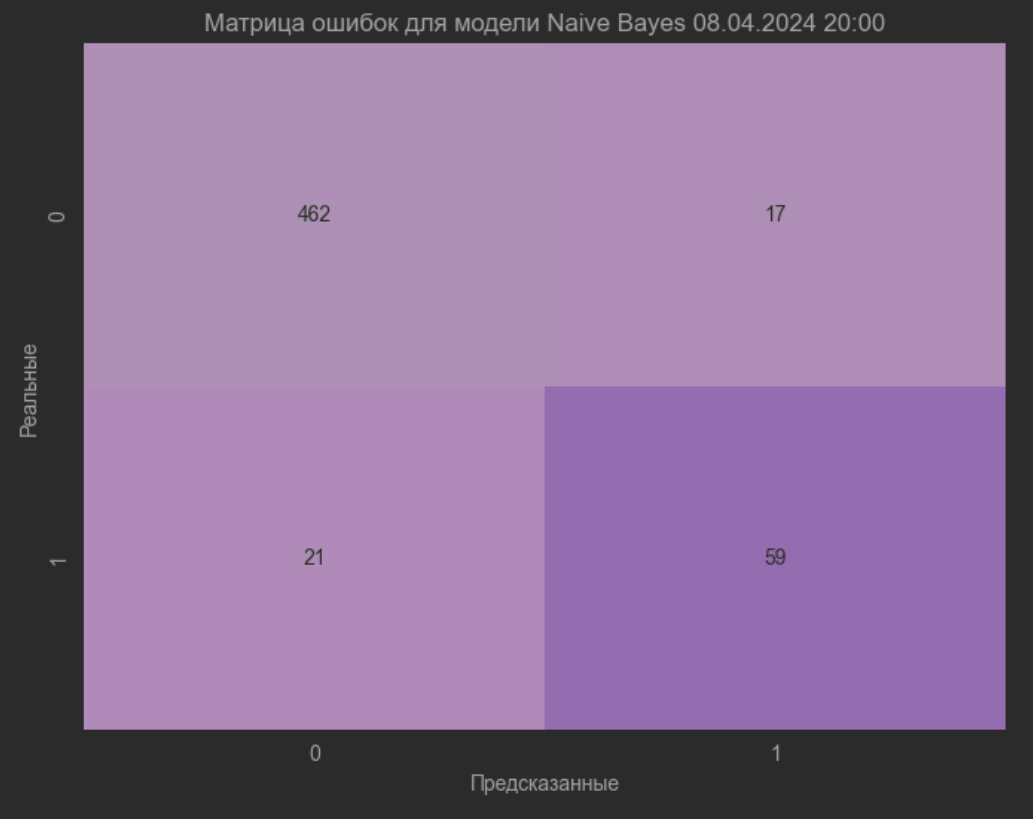


Рисунок 7 – Матрица ошибок обученной программы

## Сравнение Naive Bayes с другими методами классификации

На рисунке 8 было проведено сравнение классификаторов: Naive Bayes, логистическая регрессия (Logistic Regression), дерево решений (Decision Trees) и SVM, по метрикам эффективности и лучшие показатели метрик эффективности имеют следующее процентное соотношение: accuracy – 98% у «SVM» (и 93% у «Naive Bayes» и логистическая регрессия, 97% у дерево решений), precision\_1 – 89% у «дерево решений» (74% у «Naive Bayes» и 49% у «логистическая регрессия»), precision\_0 – 100% у «SVM» и «логистическая регрессия» (96% у «Naive Bayes» и 98% у «дерево решений»), recall\_1 – 100% у «SVM» и «логистическая регрессия» (78% у «Naive Bayes» и 89 % у «дерево решений»), recall\_0 – 98% у «SVM» и «дерево решений» (96% у «Naive Bayes» и 92% у «логистическая регрессия»), F1\_1 – 93% у «SVM» (76% у «Naive Bayes» и 66% у «логистическая регрессия», 89% у «дерево решений»), и F1\_0 – 99% у «SVM» (96% у «Naive Bayes» и «логистическая регрессия» 98% у «дерево решений»). В результате лучшим классификатором стал алгоритм SVM, Naive Bayes уступает ещё дереву решений и занимает 3-е место, а хуже всех себя показал алгоритм логистическая регрессия.

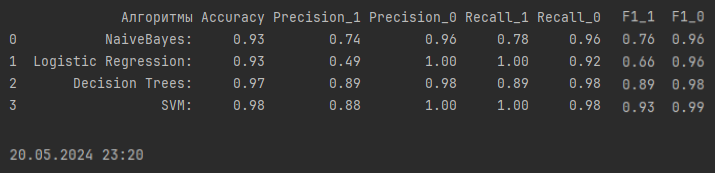


Рисунок 8 – Таблица сравнения Naive Bayes с другими методами классификации

# Итог

В результате проделанной работы была обучена модель, которая позволяет обнаруживать фишинговые письма на основе их содержания. Алгоритм Naive Bayes хорошо показывает свою эффективность на небольших объемах данных, есть возможность работать с преобразованными текстовыми данными, а также имеет относительно невысокие требования к количеству обучающих данных. Однако алгоритмы SVM и дерево решений куда лучше распределяют фишинговые и обычные письма и показывают ещё более высокую точность, чем байесовский классификатор.

В качестве улучшения уже имеющейся программы можно попробовать использовать модель мультиномиального метода сглаживания Лапласа для борьбы с проблемой нулевых вероятностей (проблема возникает, когда модель присваивает событию нулевую вероятность, так как оно не встречалось в обучающем наборе данных. Также если хотя бы одна из вероятностей равна нулю, то всё произведение становится нулевым, что делает невозможным правильную классификацию), технику отбора признаков для уменьшения влияния шумовых признаков (т.е. удаления тех признаков, которые не несут полезной информации для модели или могут даже негативно влиять на её производительность) и добавить в имеющийся набор данных больше данных с фишинговыми и обычными сообщениями.

# Список использованных источников

1. Марченко, А. Л. Python: большая книга примеров / А. Л. Марченко. — Москва : Издательство Московского университета, 2023. — 361, [1] с.
2. Бурков, А. Машинное обучение без лишних слов. — СПб.: Питер, 2020. — 192 с.: ил. — (Серия «Библиотека программиста»)
3. Васильев, А.Н. Программирование на Python в примерах и задачах. — Москва : Эксмо, 2021. — 616 с.
4. "Machine Learning Yearning" by Andrew Ng. URL:   
   https://vk.com/wall-94208167\_6035
5. Курс на платформе Stepik "Программирование на Python". URL:   
   <https://stepik.org/course/67/syllabus>
6. Сайт с документациями по библиотеке scikit-learn. URL:   
   [https://scikit-learn.ru](https://scikit-learn.ru/)
7. Сайт с документацией по библиотеке pandas. URL: https://pandas.pydata.org/docs/
8. Документация по Naive Bayes. URL:   
   <https://scikit-learn.ru/1-9-naive-bayes/?ysclid=luku4rv4c1108944510>
9. JetBrains. Getting Started. PyCharm Documentation. URL:

<https://www.jetbrains.com/help/pycharm/getting-started.html>

1. Статья про обработку текста. URL: <https://habr.com/ru/companies/otus/articles/687796/>
2. Документация по seaborn для построения графиков. URL:   
   https://seaborn.pydata.org

# Приложение

# Загрузка библиотек  
!pip install pandas  
!pip install matplotlib  
!pip install scikit-learn  
!pip install seaborn  
!pip install numpy  
  
# Импорт необходимых библиотек  
from pandas import read\_csv  
import pandas as pd  
import numpy as np  
from matplotlib import pyplot as plt  
from sklearn.metrics import confusion\_matrix  
from sklearn.linear\_model import LogisticRegression  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.svm import SVC  
from sklearn.feature\_extraction.text import TfidfVectorizer  
import seaborn as sns  
#Multinomial Naive Bayes  
from sklearn.naive\_bayes import MultinomialNB  
#Bernoulli Naive Bayes  
from sklearn.naive\_bayes import BernoulliNB  
#Gaussian Naive Bayes  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score  
# Загрузка набора данных  
  
url = "data\spam\_or\_not\_spam.csv"  
  
data = read\_csv(url)  
  
data.head()

data.describe()

data.info()

#Удаляем пустые строки  
df = data.dropna()   
  
#Удаляем повторяющиеся строки  
idx = df.loc[df.duplicated(subset=['email'], keep=False)].index  
data1 = df.drop(idx)  
  
data1.info()

# Находим индекс середины таблицы  
middle\_index = len(data1) - 380  
  
# Указываем количество строк для вывода  
num\_rows = 12  
  
# Выводим строки из середины таблицы с указанным периодом  
print(data1.iloc[middle\_index - num\_rows // 2 : middle\_index + num\_rows // 2])

# Значение, которое мы ищем  
target\_value = 1  
  
# Считаем количество строк, где значение в столбце 'B' равно target\_value  
count = (data1['label'] == target\_value).sum()  
  
print("Количество строк со значением '{}' в столбце 'B': {}".format(target\_value, count))

#Обработка текста  
vectorizer = TfidfVectorizer(stop\_words='english', lowercase=True)  
X = vectorizer.fit\_transform(data1['email'])  
y = data1['label']  
  
#Выборка на тестовые и обучающие части  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)  
X\_train\_80, X\_test\_80, y\_train\_80, y\_test\_80 = train\_test\_split(X, y, test\_size=0.8)  
X\_train\_50, X\_test\_50, y\_train\_50, y\_test\_50 = train\_test\_split(X, y, test\_size=0.5)

# Создание модели с выбранным алгоритмом  
model = DecisionTreeClassifier()  
  
# Обучение модели  
model.fit(X\_train, y\_train)  
  
# Предсказание на тестовой выборке  
y\_pred = model.predict(X\_test)  
  
# Оценка производительности модели с помощью выбранных метрик  
accuracy = accuracy\_score(y\_test, y\_pred)

# Создание модели логистической регрессии  
logistic\_model = LogisticRegression()  
  
# Обучение модели логистической регрессии на обучающих данных  
logistic\_model.fit(X\_train, y\_train)  
  
# Предсказание на тестовых данных  
y\_pred\_logistic = logistic\_model.predict(X\_test)  
  
# Оценка производительности модели логистической регрессии  
accuracy\_logistic = accuracy\_score(y\_test, y\_pred\_logistic)

# Создание модели SVM  
svm\_model = SVC(kernel='linear') # Линейное ядро  
  
# Обучение модели SVM на обучающих данных  
svm\_model.fit(X\_train, y\_train)  
  
# Предсказание на тестовых данных  
y\_pred\_svm = svm\_model.predict(X\_test)  
  
# Оценка производительности модели SVM  
accuracy\_svm = accuracy\_score(y\_test, y\_pred\_svm)

# 20% на тестирование  
X\_train\_dense = X\_train.toarray()  
X\_test\_dense = X\_test.toarray()  
  
naive\_bayes\_model\_gaussian = GaussianNB()  
  
naive\_bayes\_model\_gaussian.fit(X\_train\_dense, y\_train)  
  
y\_pred\_naive\_bayes\_gaussian = naive\_bayes\_model\_gaussian.predict(X\_test\_dense)  
  
accuracy\_gaussian = accuracy\_score(y\_test, y\_pred\_naive\_bayes\_gaussian)  
precision\_gaussian = precision\_score(y\_test, y\_pred\_naive\_bayes\_gaussian)  
recall\_gaussian = recall\_score(y\_test, y\_pred\_naive\_bayes\_gaussian)  
f1\_gaussian = f1\_score(y\_test, y\_pred\_naive\_bayes\_gaussian)  
  
# 50% на тестирование  
X\_train\_dense\_50 = X\_train\_50.toarray()  
X\_test\_dense\_50 = X\_test\_50.toarray()  
  
naive\_bayes\_model\_gaussian.fit(X\_train\_dense\_50, y\_train\_50)  
  
y\_pred\_naive\_bayes\_gaussian\_50 = naive\_bayes\_model\_gaussian.predict(X\_test\_dense\_50)  
  
accuracy\_gaussian\_50 = accuracy\_score(y\_test\_50, y\_pred\_naive\_bayes\_gaussian\_50)  
precision\_gaussian\_50 = precision\_score(y\_test\_50, y\_pred\_naive\_bayes\_gaussian\_50)  
recall\_gaussian\_50 = recall\_score(y\_test\_50, y\_pred\_naive\_bayes\_gaussian\_50)  
f1\_gaussian\_50 = f1\_score(y\_test\_50, y\_pred\_naive\_bayes\_gaussian\_50)  
  
# 80% на тестирование  
X\_train\_dense\_80 = X\_train\_80.toarray()  
X\_test\_dense\_80 = X\_test\_80.toarray()  
  
naive\_bayes\_model\_gaussian.fit(X\_train\_dense\_80, y\_train\_80)  
  
y\_pred\_naive\_bayes\_gaussian\_80 = naive\_bayes\_model\_gaussian.predict(X\_test\_dense\_80)  
  
accuracy\_gaussian\_80 = accuracy\_score(y\_test\_80, y\_pred\_naive\_bayes\_gaussian\_80)  
precision\_gaussian\_80 = precision\_score(y\_test\_80, y\_pred\_naive\_bayes\_gaussian\_80)  
recall\_gaussian\_80 = recall\_score(y\_test\_80, y\_pred\_naive\_bayes\_gaussian\_80)  
f1\_gaussian\_80 = f1\_score(y\_test\_80, y\_pred\_naive\_bayes\_gaussian\_80)  
  
# Создание модели Multinomial Naive Bayes   
naive\_bayes\_model\_multinomial = MultinomialNB()  
  
naive\_bayes\_model\_multinomial.fit(X\_train\_dense, y\_train)  
  
y\_pred\_naive\_bayes\_multinomial = naive\_bayes\_model\_multinomial.predict(X\_test\_dense)  
  
accuracy\_multinomial = accuracy\_score(y\_test, y\_pred\_naive\_bayes\_multinomial)  
precision\_multinomial = precision\_score(y\_test, y\_pred\_naive\_bayes\_multinomial)  
recall\_multinomial = recall\_score(y\_test, y\_pred\_naive\_bayes\_multinomial)  
f1\_multinomial = f1\_score(y\_test, y\_pred\_naive\_bayes\_multinomial)  
  
# Создание модели Multinomial Naive Bayes с настройкой параметра сглаживания alpha  
naive\_bayes\_model\_multinomial\_alpha = MultinomialNB( alpha=0.1 )  
  
naive\_bayes\_model\_multinomial\_alpha.fit(X\_train\_dense, y\_train)  
  
y\_pred\_naive\_bayes\_multinomial\_alpha = naive\_bayes\_model\_multinomial\_alpha.predict(X\_test\_dense)  
  
accuracy\_multinomial\_alpha = accuracy\_score(y\_test, y\_pred\_naive\_bayes\_multinomial\_alpha)  
precision\_multinomial\_alpha = precision\_score(y\_test, y\_pred\_naive\_bayes\_multinomial\_alpha)  
recall\_multinomial\_alpha = recall\_score(y\_test, y\_pred\_naive\_bayes\_multinomial\_alpha)  
f1\_multinomial\_alpha = f1\_score(y\_test, y\_pred\_naive\_bayes\_multinomial\_alpha)  
  
# Создание модели Бернолли наивного байесовского классификатора  
naive\_bayes\_model\_bernoulli = BernoulliNB()  
  
naive\_bayes\_model\_bernoulli.fit(X\_train\_dense, y\_train)  
  
y\_pred\_naive\_bayes\_bernoulli = naive\_bayes\_model\_bernoulli.predict(X\_test\_dense)  
  
accuracy\_bernoulli = accuracy\_score(y\_test, y\_pred\_naive\_bayes\_bernoulli)  
precision\_bernoulli = precision\_score(y\_test, y\_pred\_naive\_bayes\_bernoulli)  
recall\_bernoulli = recall\_score(y\_test, y\_pred\_naive\_bayes\_bernoulli)  
f1\_bernoulli = f1\_score(y\_test, y\_pred\_naive\_bayes\_bernoulli)  
  
# Вычисление матрицы ошибок Гауссовский  
cm\_naive\_bayes = confusion\_matrix(y\_pred\_naive\_bayes\_gaussian, y\_test)  
  
FP\_naive\_bayes = cm\_naive\_bayes[0, 1]  
TP\_naive\_bayes = cm\_naive\_bayes[1, 1]  
TN\_naive\_bayes = cm\_naive\_bayes[0, 0]  
FN\_naive\_bayes = cm\_naive\_bayes[1, 0]  
  
# Вычисление precision  
precision1\_naive\_bayes = TP\_naive\_bayes / (TP\_naive\_bayes + FP\_naive\_bayes)  
precision0\_naive\_bayes = TN\_naive\_bayes / (TN\_naive\_bayes + FN\_naive\_bayes)  
  
# Вычисление recall  
recall1\_naive\_bayes = TP\_naive\_bayes / (TP\_naive\_bayes + FN\_naive\_bayes)  
recall0\_naive\_bayes = TN\_naive\_bayes / (TN\_naive\_bayes + FP\_naive\_bayes)  
  
# Вычисление f1  
f11\_naive\_bayes = 2 \* precision1\_naive\_bayes \* recall1\_naive\_bayes / (precision1\_naive\_bayes + recall1\_naive\_bayes)  
f10\_naive\_bayes = 2 \* precision0\_naive\_bayes \* recall0\_naive\_bayes / (precision0\_naive\_bayes + recall0\_naive\_bayes)  
  
# Вычисление матрицы ошибок мультиномиальный  
cm\_multinomial = confusion\_matrix(y\_pred\_naive\_bayes\_multinomial, y\_test)  
  
FP\_multinomial = cm\_multinomial[0, 1]  
TP\_multinomial = cm\_multinomial[1, 1]  
TN\_multinomial = cm\_multinomial[0, 0]  
FN\_multinomial = cm\_multinomial[1, 0]  
  
# Вычисление precision  
precision1\_multinomial = TP\_multinomial / (TP\_multinomial + FP\_multinomial)  
precision0\_multinomial = TN\_multinomial / (TN\_multinomial + FN\_multinomial)  
  
# Вычисление recall  
recall1\_multinomial = TP\_multinomial / (TP\_multinomial + FN\_multinomial)  
recall0\_multinomial = TN\_multinomial / (TN\_multinomial + FP\_multinomial)  
  
# Вычисление f1  
f11\_multinomial = 2 \* precision1\_multinomial \* recall1\_multinomial / (precision1\_multinomial + recall1\_multinomial)  
f10\_multinomial = 2 \* precision0\_multinomial \* recall0\_multinomial / (precision0\_multinomial + recall0\_multinomial)  
  
# Вычисление матрицы ошибок мультиномиальный улучшенный  
cm\_alpha = confusion\_matrix(y\_pred\_naive\_bayes\_multinomial\_alpha, y\_test)  
  
FP\_alpha = cm\_alpha[0, 1]  
TP\_alpha = cm\_alpha[1, 1]  
TN\_alpha = cm\_alpha[0, 0]  
FN\_alpha = cm\_alpha[1, 0]  
  
# Вычисление precision  
precision1\_alpha = TP\_alpha / (TP\_alpha + FP\_alpha)  
precision0\_alpha = TN\_alpha / (TN\_alpha + FN\_alpha)  
  
# Вычисление recall  
recall1\_alpha = TP\_alpha / (TP\_alpha + FN\_alpha)  
recall0\_alpha = TN\_alpha / (TN\_alpha + FP\_alpha)  
  
# Вычисление f1  
f11\_alpha = 2 \* precision1\_alpha \* recall1\_alpha / (precision1\_alpha + recall1\_alpha)  
f10\_alpha = 2 \* precision0\_alpha \* recall0\_alpha / (precision0\_alpha + recall0\_alpha)  
  
# Вычисление матрицы ошибок Бернулли  
cm\_bernoulli = confusion\_matrix(y\_pred\_naive\_bayes\_bernoulli, y\_test)  
  
FP\_bernoulli = cm\_bernoulli[0, 1]  
TP\_bernoulli = cm\_bernoulli[1, 1]  
TN\_bernoulli = cm\_bernoulli[0, 0]  
FN\_bernoulli = cm\_bernoulli[1, 0]  
  
# Вычисление precision  
precision1\_bernoulli = TP\_bernoulli / (TP\_bernoulli + FP\_bernoulli)  
precision0\_bernoulli = TN\_bernoulli / (TN\_bernoulli + FN\_bernoulli)  
  
# Вычисление recall  
recall1\_bernoulli = TP\_bernoulli / (TP\_bernoulli + FN\_bernoulli)  
recall0\_bernoulli = TN\_bernoulli / (TN\_bernoulli + FP\_bernoulli)  
  
# Вычисление f1  
f11\_bernoulli = 2 \* precision1\_bernoulli \* recall1\_bernoulli / (precision1\_bernoulli + recall1\_bernoulli)  
f10\_bernoulli = 2 \* precision0\_bernoulli \* recall0\_bernoulli / (precision0\_bernoulli + recall0\_bernoulli)  
  
cm\_50 = confusion\_matrix(y\_pred\_naive\_bayes\_gaussian\_50, y\_test\_50)  
  
FP\_50 = cm\_50[0, 1]  
TP\_50 = cm\_50[1, 1]  
TN\_50 = cm\_50[0, 0]  
FN\_50 = cm\_50[1, 0]  
  
# Вычисление precision  
precision1\_50 = TP\_50 / (TP\_50 + FP\_50)  
precision0\_50 = TN\_50 / (TN\_50 + FN\_50)  
  
# Вычисление recall  
recall1\_50 = TP\_50 / (TP\_50 + FN\_50)  
recall0\_50 = TN\_50 / (TN\_50 + FP\_50)  
  
# Вычисление f1  
f11\_50 = 2 \* precision1\_50 \* recall1\_50 / (precision1\_50 + recall1\_50)  
f10\_50 = 2 \* precision0\_50 \* recall0\_50 / (precision0\_50 + recall0\_50)  
  
cm\_80 = confusion\_matrix(y\_pred\_naive\_bayes\_gaussian\_80, y\_test\_80)  
  
FP\_80 = cm\_80[0, 1]  
TP\_80 = cm\_80[1, 1]  
TN\_80 = cm\_80[0, 0]  
FN\_80 = cm\_80[1, 0]  
  
# Вычисление precision  
precision1\_80 = TP\_80 / (TP\_80 + FP\_80)  
precision0\_80 = TN\_80 / (TN\_80 + FN\_80)  
  
# Вычисление recall  
recall1\_80 = TP\_80 / (TP\_80 + FN\_80)  
recall0\_80 = TN\_80 / (TN\_80 + FP\_80)  
  
# Вычисление f1  
f11\_80 = 2 \* precision1\_80 \* recall1\_80 / (precision1\_80 + recall1\_80)  
f10\_80 = 2 \* precision0\_80 \* recall0\_80 / (precision0\_80 + recall0\_80)  
results\_df = pd.DataFrame(columns=['Метод', 'Accuracy', 'Precision\_1', 'Precision\_0', 'Recall\_1', 'Recall\_0', 'F1\_1', 'F1\_0'])  
  
results\_df.loc[0] = ['Multinomial', "{:.2f}".format(accuracy\_multinomial), "{:.2f}".format(precision1\_multinomial), "{:.2f}".format(precision0\_multinomial), "{:.2f}".format(recall1\_multinomial), "{:.2f}".format(recall0\_multinomial), "{:.2f}".format(f11\_multinomial), "{:.2f}".format(f10\_multinomial)]  
results\_df.loc[1] = ['Multinomial\_alpha', "{:.2f}".format(accuracy\_multinomial\_alpha), "{:.2f}".format(precision1\_alpha), "{:.2f}".format(precision0\_alpha), "{:.2f}".format(recall1\_alpha), "{:.2f}".format(recall0\_alpha), "{:.2f}".format(f11\_alpha), "{:.2f}".format(f10\_alpha)]  
results\_df.loc[2] = ['Bernoulli', "{:.2f}".format(accuracy\_bernoulli), "{:.2f}".format(precision1\_bernoulli), "{:.2f}".format(precision0\_bernoulli), "{:.2f}".format(recall1\_bernoulli), "{:.2f}".format(recall0\_bernoulli), "{:.2f}".format(f11\_bernoulli), "{:.2f}".format(f10\_bernoulli)]  
results\_df.loc[3] = ['Gaussian', "{:.2f}".format(accuracy\_gaussian), "{:.2f}".format(precision1\_naive\_bayes), "{:.2f}".format(precision0\_naive\_bayes), "{:.2f}".format(recall1\_naive\_bayes), "{:.2f}".format(recall0\_naive\_bayes), "{:.2f}".format(f11\_naive\_bayes), "{:.2f}".format(f10\_naive\_bayes)]  
  
# Выводим результаты  
print(results\_df)  
  
print("")  
print("17.04.2024 21:52")

results\_df = pd.DataFrame(columns=['Проценты на тестовую часть', 'Accuracy', 'Precision\_1', 'Precision\_0', 'Recall\_1', 'Recall\_0', 'F1\_1', 'F1\_0'])  
  
results\_df.loc[0] = ['20% на тестирование', "{:.2f}".format(accuracy\_gaussian), "{:.2f}".format(precision1\_naive\_bayes), "{:.2f}".format(precision0\_naive\_bayes), "{:.2f}".format(recall1\_naive\_bayes), "{:.2f}".format(recall0\_naive\_bayes), "{:.2f}".format(f11\_naive\_bayes), "{:.2f}".format(f10\_naive\_bayes)]  
results\_df.loc[1] = ['50% на тестирование', "{:.2f}".format(accuracy\_gaussian\_50), "{:.2f}".format(precision1\_50), "{:.2f}".format(precision0\_50), "{:.2f}".format(recall1\_50), "{:.2f}".format(recall0\_50), "{:.2f}".format(f11\_50), "{:.2f}".format(f10\_50)]  
results\_df.loc[2] = ['80% на тестирование', "{:.2f}".format(accuracy\_gaussian\_80),"{:.2f}".format(precision1\_80), "{:.2f}".format(precision0\_80), "{:.2f}".format(recall1\_80), "{:.2f}".format(recall0\_80), "{:.2f}".format(f11\_80), "{:.2f}".format(f10\_80)]  
  
# Выводим результаты  
print(results\_df)  
  
print("")  
print("17.04.2024 23:22")

# Вывод результатов  
print("Метрики производительности 08.04.2024 20:08:")  
print("Accuracy:", "{:.2f}".format(accuracy\_gaussian))  
print("Precision\_1:", "{:.2f}".format(precision1\_naive\_bayes))  
print("Precision\_0:", "{:.2f}".format(precision0\_naive\_bayes))  
print("Recall\_1:", "{:.2f}".format(recall1\_naive\_bayes))  
print("Recall\_0:", "{:.2f}".format(recall0\_naive\_bayes))  
print("F1\_1:", "{:.2f}".format(f11\_naive\_bayes))  
print("F1\_0:", "{:.2f}".format(f10\_naive\_bayes))

unique\_labels\_test, counts\_test = np.unique(y\_test, return\_counts=True)  
unique\_labels\_pred, counts\_pred = np.unique(y\_pred\_naive\_bayes\_gaussian, return\_counts=True)  
  
name = ['1', '2']  
  
# Создание подписей для классов  
labels = ['Обычное', 'Фишинговое']  
  
# Создание фигуры и осей  
fig, ax = plt.subplots()  
  
# Ширина столбцов  
width = 0.35  
  
# Создание столбцов для тестовых и предсказанных значений  
bars1 = ax.bar(np.arange(len(counts\_test)) - width/2, counts\_test, width, label='Реальные', color='#291048')  
bars2 = ax.bar(np.arange(len(counts\_pred)) + width/2, counts\_pred, width, label='Предсказанные', color='#7F26EF')  
  
# Добавление значений к столбцам  
for i, count in enumerate(counts\_test):  
 ax.text(i - width/2, count + 1, str(count), ha='center', va='bottom')  
  
for i, count in enumerate(counts\_pred):  
 ax.text(i + width/2, count + 1, str(count), ha='center', va='bottom')  
  
# Установка подписей и заголовка  
ax.set\_xticks(np.arange(len(labels)))  
ax.set\_xticklabels(labels)  
ax.set\_title('Распределение реальных и предсказанных результатов')  
  
# Добавление легенды  
ax.legend()  
  
# Отображение графика  
plt.show()

# Вычисление матрицы ошибок для модели Naive Bayes  
conf\_matrix\_naive\_bayes = confusion\_matrix(y\_pred\_naive\_bayes\_gaussian, y\_test)  
  
# Создание тепловой карты для матрицы ошибок  
plt.figure(figsize=(8, 6))  
sns.heatmap(conf\_matrix\_naive\_bayes, annot=True, cmap='twilight\_shifted', fmt='g', cbar=False)  
  
# Настройка названий осей и заголовка  
plt.title('Матрица ошибок для модели Naive Bayes 08.04.2024 20:00')  
  
# Показать график  
plt.show()

cm\_logistic = confusion\_matrix(y\_pred\_logistic, y\_test)  
  
FP\_logistic = cm\_logistic[0, 1]  
TP\_logistic = cm\_logistic[1, 1]  
TN\_logistic = cm\_logistic[0, 0]  
FN\_logistic = cm\_logistic[1, 0]  
  
# Вычисление precision  
precision1\_logistic = TP\_logistic / (TP\_logistic + FP\_logistic)  
precision0\_logistic = TN\_logistic / (TN\_logistic + FN\_logistic)  
  
# Вычисление recall  
recall1\_logistic = TP\_logistic / (TP\_logistic + FN\_logistic)  
recall0\_logistic = TN\_logistic / (TN\_logistic + FP\_logistic)  
  
# Вычисление f1  
f11\_logistic = 2 \* precision1\_logistic \* recall1\_logistic / (precision1\_logistic + recall1\_logistic)  
f10\_logistic = 2 \* precision0\_logistic \* recall0\_logistic / (precision0\_logistic + recall0\_logistic)  
  
cm = confusion\_matrix(y\_pred, y\_test)  
  
FP = cm[0, 1]  
TP = cm[1, 1]  
TN = cm[0, 0]  
FN = cm[1, 0]  
  
# Вычисление precision  
precision1 = TP / (TP + FP)  
precision0 = TN / (TN + FN)  
  
# Вычисление recall  
recall1 = TP / (TP + FN)  
recall0 = TN / (TN + FP)  
  
# Вычисление f1  
f11 = 2 \* precision1 \* recall1 / (precision1 + recall1)  
f10 = 2 \* precision0 \* recall0 / (precision0 + recall0)  
  
cm\_svm = confusion\_matrix(y\_pred\_svm, y\_test)  
  
FP\_svm = cm\_svm[0, 1]  
TP\_svm = cm\_svm[1, 1]  
TN\_svm = cm\_svm[0, 0]  
FN\_svm = cm\_svm[1, 0]  
  
# Вычисление precision  
precision1\_svm = TP\_svm / (TP\_svm + FP\_svm)  
precision0\_svm = TN\_svm / (TN\_svm + FN\_svm)  
  
# Вычисление recall  
recall1\_svm = TP\_svm / (TP\_svm + FN\_svm)  
recall0\_svm = TN\_svm / (TN\_svm + FP\_svm)  
  
# Вычисление f1  
f11\_svm = 2 \* precision1\_svm \* recall1\_svm / (precision1\_svm + recall1\_svm)  
f10\_svm = 2 \* precision0\_svm \* recall0\_svm / (precision0\_svm + recall0\_svm)  
  
results\_df = pd.DataFrame(columns=['Алгоритмы', 'Accuracy', 'Precision\_1', 'Precision\_0', 'Recall\_1', 'Recall\_0', 'F1\_1', 'F1\_0'])  
  
results\_df.loc[0] = ['NaiveBayes:', "{:.2f}".format(accuracy\_gaussian), "{:.2f}".format(precision1\_naive\_bayes),"{:.2f}".format(precision0\_naive\_bayes), "{:.2f}".format(recall1\_naive\_bayes), "{:.2f}".format(recall0\_naive\_bayes), "{:.2f}".format(f11\_naive\_bayes), "{:.2f}".format(f10\_naive\_bayes)]  
results\_df.loc[1] = ['Logistic Regression:', "{:.2f}".format(accuracy\_logistic), "{:.2f}".format(precision1\_logistic),"{:.2f}".format(precision0\_logistic), "{:.2f}".format(recall1\_logistic), "{:.2f}".format(recall0\_logistic), "{:.2f}".format(f11\_logistic), "{:.2f}".format(f10\_logistic)]  
results\_df.loc[2] = ['Decision Trees:', "{:.2f}".format(accuracy), "{:.2f}".format(precision1),"{:.2f}".format(precision0), "{:.2f}".format(recall1), "{:.2f}".format(recall0), "{:.2f}".format(f11), "{:.2f}".format(f10)]  
results\_df.loc[3] = ['SVM:', "{:.2f}".format(accuracy\_svm), "{:.2f}".format(precision1\_svm),"{:.2f}".format(precision0\_svm), "{:.2f}".format(recall1\_svm), "{:.2f}".format(recall0\_svm), "{:.2f}".format(f11\_svm), "{:.2f}".format(f10\_svm)]  
  
# Выводим результаты  
print(results\_df)  
  
print("")  
print("20.05.2024 23:20")