Федеральное государственное автономное

образовательное учреждение высшего образования

«Пермский государственный национальный исследовательский университет» (ПГНИУ)

Региональный институт непрерывного образования (РИНО ПГНИУ)

Цифровая кафедра

Выпускная аттестационная (квалификационная) работа

по курсу профессиональной переподготовки «Анализ данных и машинное обучение»

**КЛАССИФИКАЦИЯ ПАРОЛЕЙ ПО ИХ СЛОЖНОСТИ**

Разработчики проекта:

Мазунин Иван Вадимович,

Любов Кирилл Дмитриевич,

Щедрина Наталья Сергеевна

Пермь, 2024

**Оглавление**

[ПАСПОРТ ПРОЕКТА 3](#_Toc185368129)

[СОДЕРЖАНИЕ ПРОЕКТА 4](#_Toc185368130)

[Анализ проблемы исследования 4](#_Toc185368131)

[Исходные данные 6](#_Toc185368132)

[Реализация проекта 7](#_Toc185368133)

[Этап 1. Подготовка данных к анализу 7](#_Toc185368134)

[Этап 2. Предварительный анализ данных 10](#_Toc185368135)

[Этап 3. Корреляционный анализ данных 12](#_Toc185368136)

[Этап 4. Создание и обучение моделей классификаторов. 14](#_Toc185368137)

[Этап 5. Задача NLP 21](#_Toc185368138)

[ЗАКЛЮЧЕНИЕ 27](#_Toc185368139)

[СПИСОК ЛИТЕРАТУРЫ 28](#_Toc185368140)

[ПРИЛОЖЕНИЕ 30](#_Toc185368141)

# ПАСПОРТ ПРОЕКТА

**Название проекта:**Классификация паролей по их сложности.

**Сведения об авторах:** Мазунин Иван Вадимович, Любов Кирилл Дмитриевич, Щедрина Наталья Сергеевна

**Цель:**выполнить анализ данных о паролях и построить модель классификации от различных факторов с помощью некоторых классификаторов, позволяющую делать прогнозы с высокой точностью. Также, реализовать задачу NLP и сравнить полученные результаты.

**Задачи:**

1. Выполнить анализ проблемы, обосновать ее актуальность.
2. Осуществить загрузку данных и подготовку их к анализу.
3. Выполнить предварительный анализ данных, в том числе корреляционный анализ.
4. Осуществить моделирование зависимости целевого признака от остальных с помощью различных классификаторов, в том числе подобрать наилучшую модель, оценить ее качество и выполнить прогнозирование.
5. Осуществить моделирование с помощью методов NLP.
6. Выбрать лучшую модель и сделать выводы о достижении цели.

**Краткое описание проекта:**

Требуется построить и обучить модели, создав признаки на основе паролей и реализовав задачу NLP. Дать интерпретацию полученным результатам и выбрать лучшую модель.

**Конкретные ожидаемые результаты:**

Построенные модели классификации и рекомендации по ее использованию.

# СОДЕРЖАНИЕ ПРОЕКТА

Анализ проблемы исследования

В современном мире, где информационные технологии стремительно развиваются, тема безопасных паролей становится особенно актуальной и значимой. Это связано с несколькими факторами, которые определяют важность исследования и классификации паролей по их надежности:

1. **Увеличение числа киберугроз**

С каждым годом количество кибератак значительно возрастает. Хакеры разрабатывают все более сложные методы для осуществления взломов аккаунтов и систем. Наиболее уязвимым местом для подобных атак часто становятся пароли. По данным различных исследований, многие пользователи продолжают использовать простые и легко запоминаемые пароли, что делает их лёгкой мишенью для злоумышленников.

1. **Необходимость защиты личных данных**

В эпоху цифровизации каждый из нас оставляет массу личной информации в сети — от фотографий и переписки до финансовых данных и учетных записей. Безопасность этой информации зачастую зависит от надежности паролей. Утечка данных может привести не только к потере личной информации, но и к финансовым потерям, нарушению репутации и другим негативным последствиям.

1. **Социальные и правовые последствия**

Не только индивидуальные пользователи, но и организации сталкиваются с последствиями утечки данных. В большинстве стран существуют законы о защите данных, которые обязывают компании обеспечивать безопасность информации пользователей. Слабые пароли могут привести к огромным штрафам и судебным искам, а также к потере доверия клиентов.

1. **Развитие технологий аутентификации**

В ответ на угрозы кибербезопасности разрабатываются новые механизмы аутентификации, такие как двухфакторная аутентификация и биометрические методы. Однако даже лучшие технологии не будут эффективными, если пользователи продолжают использовать слабые пароли. Классификация надежности паролей может служить основой для улучшения пользовательских практик и повышения общей безопасности.

Таким образом, актуальность исследования классификации паролей по их надежности обуславливается сочетанием факторов — от увеличения киберугроз до юридических и финансовых последствий для организаций. В условиях быстрой эволюции технологий, понимание и внедрение принципов создания надежных паролей является важным шагом к обеспечению безопасности в мире цифровых услуг. Разработка эффективных методов классификации паролей может значительно повысить уровень защиты информации и снизить риски, связанные с киберпреступностью.

**Цель:**выполнить анализ данных о паролях и построить модель классификации от различных факторов с помощью некоторых классификаторов, позволяющую делать прогнозы с высокой точностью. Также, реализовать задачу NLP и сравнить полученные результаты.

**Задачи:**

1. Выполнить анализ проблемы, обосновать ее актуальность.
2. Осуществить загрузку данных и подготовку их к анализу.
3. Выполнить предварительный анализ данных, в том числе корреляционный анализ.
4. Осуществить моделирование зависимости целевого признака от остальных с помощью различных классификаторов, в том числе подобрать наилучшую модель, оценить ее качество и выполнить прогнозирование.
5. Осуществить моделирование с помощью методов NLP.
6. Выбрать лучшую модель и сделать выводы о достижении цели.

Исходные данные

В дальнейшей работе используется набор данных, который был выдан в «**Конкурс красоты кода от Сбера**», где нужно было классифицировать сложность пароля на 3 класса.

Набор данных состоит из 2 столбцов:

1. password - пароль в виде строки;
2. strength - класс сложности (0, 1, 2).

Где 0 — ненадежный пароль, 1 – средней надежности, 2 — очень надежный.

Необходимо проанализировать пароли и определить, какими признаками можно описать эти пароли и на основе них создать набор данных для дальнейшего обучения на различных моделях классификации. А также предобработать пароли для реализации задачи NLP.

Реализация проекта

### Этап 1. Подготовка данных к анализу

Подключаем необходимые библиотеки:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.utils import resample

import re

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import StratifiedKFold

from sklearn.model\_selection import KFold

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn.metrics import accuracy\_score

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from imblearn.under\_sampling import RandomUnderSampler

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.preprocessing.text import Tokenizer

from sklearn.preprocessing import LabelEncoder

from imblearn.pipeline import Pipeline

Загрузим данные в объект DataFrame:

data = pd.read\_csv('passwords.csv')

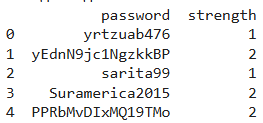


Рисунок 1. Исходный датафрейм

Убедимся, что все количественные столбцы имеют числовой тип. Если это не так, выполним преобразование типа столбца к числовому.

print(data.info())

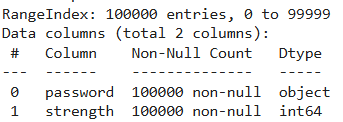


Рисунок 2. Типы данных колонок

Видим, что целевой признак strength является числовым, а пароли password являются типом object(по факту пароли – строка). Нам необходимо преобразовать этот датафрейм в числовые значения, чтобы мы могли применить классификаторы. Для этого мы создаем отдельную функцию, которая добавляет в исходный датафрейм столбцы, обозначающие следующие признаки пароля:

1. length - длина пароля
2. uppercase - количество заглавных букв
3. lowercase - количество строчных букв
4. special\_chars - количество специальных символов
5. digits - количество цифр
6. max\_consecutive - максимальное количество подряд идущих символов одной категориии
7. uniq\_chars\_% - отношение количества уникальных символов к длине пароля
8. pass\_is\_popular? - является пароль популярным или нет

def add\_features(df):

df['length'] = df['password'].apply(len)

    df['special\_chars'] = df['password'].apply(lambda x: len(re.findall(r'[\W]', x)))

    df['uppercase'] = df['password'].apply(lambda x: len(re.findall(r'[A-Z]', x)))

    df['lowercase'] = df['password'].apply(lambda x: len(re.findall(r'[a-z]', x)))

    df['digits'] = df['password'].apply(lambda x: len(re.findall(r'[0-9]', x)))

    # Максимальная длина последовательных символов одной категории

    def max\_sequence\_length(password):

        max\_length = 0

        current\_length = 1

        for i in range(1, len(password)):

            if (password[i].islower() and password[i-1].islower()) or \

               (password[i].isupper() and password[i-1].isupper()) or \

               (password[i].isdigit() and password[i-1].isdigit()) or \

               (not password[i].isalnum() and not password[i-1].isalnum()):

                current\_length += 1

            else:

                max\_length = max(max\_length, current\_length)

                current\_length = 1

        max\_length = max(max\_length, current\_length)  # Проверка на конец строки

        return max\_length

    df['max\_consecutive'] = df['password'].apply(max\_sequence\_length)

    def unique\_chars(password):

        # Находим уникальные символы

        unique\_characters = set(password)

        # Вычисляем соотношение уникальных символов к общему количеству символов

        uniq = len(unique\_characters) / len(password)

        return uniq

    df['uniq\_chars\_%'] = df['password'].apply(unique\_chars)

    #Загружаем список популярных паролей и сравниваем с нашими из датасета

    with open('worst\_passwords.txt', 'r') as file:

        typical\_passwords = {line.strip() for line in file}

    #Проверяем, является ли пароль популярным

    df['pass\_is\_popular?'] = df['password'].apply(lambda x: 1 if x in typical\_passwords else 0)

    return df

new\_data = add\_features(data)

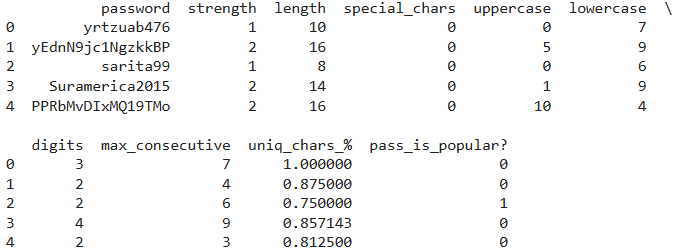


Рисунок 3. Дополненный датафрейм

Так как все значения у нас не пустые и дополнительной обработки нам не требуется, поэтому мы переходим к следующему этапу.

### Этап 2. Предварительный анализ данных

Для классификации очень важным является наличие или отсутствие баланса в количестве значений различных классов. Построим гистограмму распределения классов в нашем наборе данных.

# Строим гистограмму целевой переменной 'strength'

plt.figure(figsize=(8, 5))

data['strength'].value\_counts().plot(kind='bar')

plt.title('Распределение классов')

plt.xlabel('Класс')

plt.ylabel('Количество')

plt.xticks(ticks=[0,1,2], labels=[1,0,2], rotation=0)

# Показываем график

plt.show()

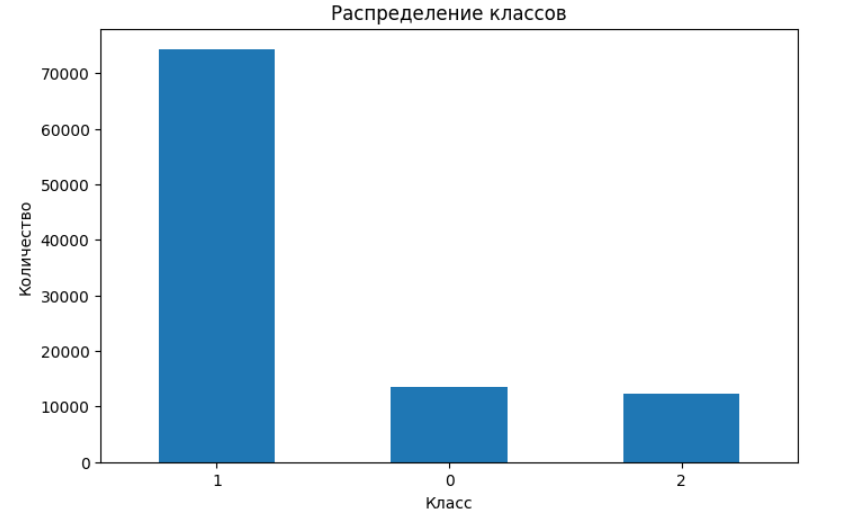


Рисунок 4. Гистограмма распределения классов.

Мы наблюдаем, что классы несбалансированные – класс «1» сильно доминирует над «0» и «2» по количеству значений. Дополнительно выведем количество паролей в каждом классе.

data['strength'].value\_counts()

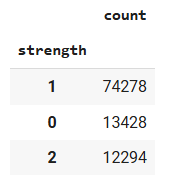


Рисунок 5. Количество паролей в классах.

Далее в работе нам будет необходимо устранить дисбаланс любыми способами. Мы выбрали метод RandomUnderSampler, так как разница небольшая (примерно в 7 раз). Также этот метод уменьшит общий объём выборки и ускорит обучение. Описание и реализация метода описана в этапе 4.

### Этап 3. Корреляционный анализ данных

**Корреляционный анализ** – это совокупность методов оценивания степени тесноты статистической связи между анализируемыми переменными.

Выполним корреляционный анализ данных с помощью матрицы корреляции. В нашем случае мы будем использовать метод Пирсона.

**Парный коэффициент** корреляции характеризует взаимосвязь двух переменных на фоне действия остальных показателей и является самым распространенным показателем тесноты связи при статистическом анализе данных.

Парный коэффициент корреляции между количественными случайными переменными и носит название *выборочного коэффициента корреляции* Пирсона (*sample correlation coefficient*) (или просто коэффициента корреляции) и находится по формуле

где и  — *выборочные дисперсии (sample variances*) переменных  и , а — *выборочная ковариация* или выборочный ковариационный момент, и соответствующие *средние (means)* определяются по формулам



Коэффициент корреляции обладает следующими свойствами:

1. Принимает значения от –1 до +1.
2. Если , то связь между переменными  и  считается сильной. Если , то связь слабая.
3. Если , то корреляционное поле наблюдений представляет собой совокупность точек, которые можно расположить на одной прямой. Знак «+» свидетельствует о прямой линейной зависимости между переменными и , а знак «—» − об обратной линейной зависимости.
4. При  линейная корреляционная связь отсутствует.

Построим тепловую карту для визуализации матрицы корреляции по методу Пирсона для нашего набора данных.

plt.figure(figsize=(7,6))

sns.heatmap(new\_data.corr(numeric\_only = True,method='pearson'), annot=True)

plt.show()

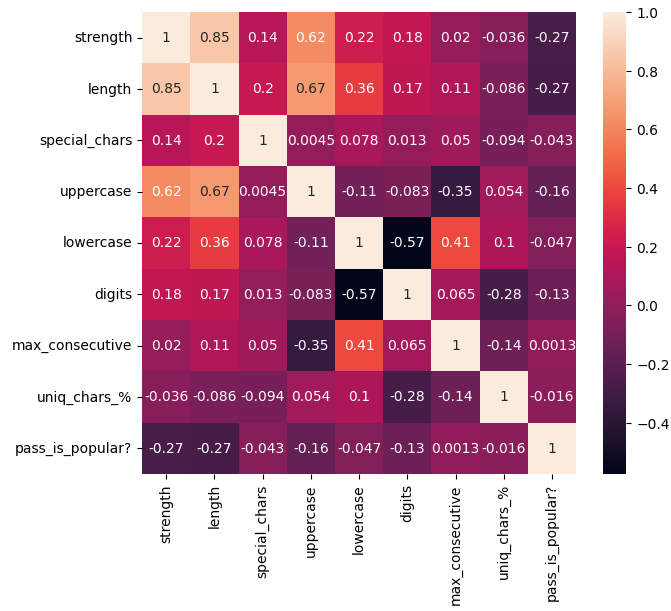


Рисунок 6. Матрица корреляции для выделенных признаков.

Наиболее коррелирующими с целевым признаком является длина пароля и количество символов верхнего регистра (length и uppercase). Так как корреляция с остальными признаками слабая, то мы можем их убрать из нашего набора данных для ускорения обучения, не теряя особо в точности модели.

### Этап 4. Создание и обучение моделей классификаторов.

Сначала мы определяемся с теми классификаторами, которые мы хотим обучить, чтобы выбрать из них наилучший вариант. Мы остановились на следующих вариантах:

1. Логистическая регрессия (LogisticRegression)
2. Метод К-ближайших соседей (KNeighborsClassifier)
3. Дерево решений (DecisionTreeClassifier)
4. Метод опорных векторов (SVC)

tuned\_classifiers = {

    "LogisiticRegression": LogisticRegression(random\_state=42),

    "KNearest": KNeighborsClassifier(n\_neighbors=5),

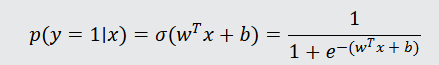
    "DecisionTreeClassifier": DecisionTreeClassifier(random\_state=42),

    "Support Vector Classifier": SVC()

}

1. **Logistic Regression**

Логистическая регрессия применяется для бинарной классификации (хотя ее можно адаптировать для многоклассовых задач). Основная идея заключается в моделировании вероятности принадлежности наблюдения к классу, используя функцию, описанную ниже. Она использует линейную комбинацию входных признаков и функцию сигмоида для преобразования этой комбинации в значение вероятности принадлежности к тому или иному классу. То есть, для заданных признаков x:



где w - вектор весов, b - сдвиг, а σ - сигмоидальная функция. Вектор весов определяется по тестовой выборке с помощью методов оптимизации. й

1. **KNeighborsClassifier**

Метод k-ближайших соседей (KNN) — это алгоритм, который классифицирует объекты на основе их ближайших соседей в пространстве признаков. Для нового (неизвестного) наблюдения KNN находит K ближайших обучающих объектов и классифицирует новое наблюдение на основе большинства классов соседей. Расстояние между объектами может измеряться различными способами, например, с помощью евклидова расстояния или манхэттенского расстояния.

1. **DecisionTreeClassifier**

Деревья решений используют структуру, напоминающую дерево, где каждый узел представляет собой вопрос о значениях признака, а каждое ответвление — результат этого вопроса. Процесс деления продолжается до тех пор, пока не будут достигнуты условия остановки (например, максимальная глубина или минимальное количество образцов в листовом узле). Основные критерии разбиения: «информационная выгода» (information gain), Gini impurity и другие.

1. **SVC (Support Vector Classifier)**

Опорные векторы (SVM) — это метод, который находит гиперплоскость, максимально разделяющую классы в многомерном пространстве. Он ищет гиперплоскость, которая максимизирует расстояние до ближайших точек (опорных векторов) каждого из классов. Для нелинейно разделимых данных SVM может использовать ядерные функции (kernels) для величайшего преобразования данных в более высокое измерение, где данные могут быть линейно разделимыми.

Разбиваем данные на тренировочную и тестовые выборки в соотношении 4:1.

Также устраняем дисбаланс в распределении классов с помощью метода **RandomUnderSampler**. Этот метод случайным образом выбирает экземпляры из класса с более высоким количеством, чтобы уменьшить его размер до уровня минимального класса. Это означает, что выбираются случайные подмножества данных, чтобы сбалансировать классы.

# Определяем признаки и целевую переменную из несбалансированного набора данных

X = new\_data[['length','uppercase']]  # Признаки (все столбцы кроме целевой переменной)

y = new\_data['strength']  # Целевая переменная

original\_Xtrain, original\_Xtest, original\_ytrain, original\_ytest = train\_test\_split(X, y, test\_size=0.2,random\_state = 42)

# Создаем объект для уменьшения выборки для большинства

under = RandomUnderSampler()

**Кросс-валидация** — это метод оценки качества модели, который позволяет проверить ее способность обобщать, избегая переобучения. Этот метод заключается в следующем:

* Разделение данных: Исходный набор данных делится на несколько частей (или "фолдов"). Обычно используется от 5 до 10 фолдов.
* Обучение и тестирование: Модель обучается на части данных (например, на 4 фолдах), а затем тестируется на оставшейся части (1 фолде). Этот процесс повторяется для каждого из фолдов.
* Средний результат: после завершения всех итераций рассчитывается средний показатель качества (например, точность), чтобы получить более надежную оценку производительности модели.

Преимущества кросс-валидации:

* Позволяет избежать переобучения, так как модель проверяется на разных подмножестве данных.
* Обеспечивает более надежную оценку производительности модели по сравнению с простым разделением на тренировочный и тестовый наборы.
* Кросс-валидация помогает лучше понять, как модель будет вести себя на новых, ранее невидимых данных, что критически важно для задач машинного обучения.

# Используем StratifiedKFold для обеспечения стратифицированной кросс-валидации

# Это объект кросс-валидации, который позволяет сохранить процент образцов для каждого класса в каждом фоне.

sss = StratifiedKFold(n\_splits=5, random\_state=None, shuffle=False)  # 5 образцов для кросс-валидации

**Перейдем к обучению моделей:**

**Pipeline** — это инструмент для организации последовательной обработки данных, упрощающий написание, отладку и сопровождение кода, особенно в контексте машинного обучения.

import time

# Итерируемся по словарю с настроенными классификаторами

for key, classifier in tuned\_classifiers.items():

    time1 = time.time()

    # Создаем список шагов для пайплайна машинного обучения

    steps = [('u', under), ('model', classifier)]  # Шаги: уменьшенная выборка и модель

    # Создаем пайплайн машинного обучения

    FiPipeline = Pipeline(steps=steps)  # Инициализация пайплайна с указанными шагами

    # Кросс-валидация классификаторов с использованием определенного пайплайна

    scores = cross\_val\_score(FiPipeline, original\_Xtrain, original\_ytrain, cv=sss)  # Оценка модели с помощью кросс-валидации

    time2 = time.time()

    # Выводим средний коэффициент точности

    print(f"Название: {key}, оценка точности: {round(scores.mean() \* 100, 2)} %, время обучения : {(time2-time1):0.2f} секунд")  # Печать результатов

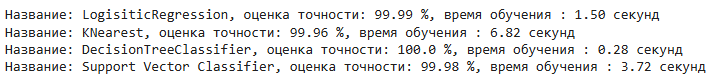


Рисунок 7. Результат обучения различных моделей классификации.

По итогам обучения все модели обучились замечательно – точность вплоть до 100%. Но модель «дерево решений» обучилась быстрее всех, поэтому будем считать, что это наилучший вариант. Обучим эту модель и выведем результаты обучения в виде отчета о классификации и матрицы ошибок.

# Создание модели

model = Pipeline([

    ('u', RandomUnderSampler()),  # Шаг для уменьшения выборки большинства

    (‘classification’, DecisionTreeClassifier()) ])

model.fit(original\_Xtrain, original\_ytrain)  # Обучаем модель

smote\_prediction = model.predict(original\_Xtest)

# Вывод метрик

labels = ['0', '1','2']

print(classification\_report(original\_ytest, smote\_prediction, target\_names=labels))

conf\_matrix = confusion\_matrix(original\_ytest, smote\_prediction)

plt.figure(figsize=(6, 4))

sns.heatmap(conf\_matrix, annot=True, cmap=”Blues”)

plt.xlabel(“Predicted”)

plt.ylabel(“Actual”)

plt.title(“Confusion Matrix Plot”)

plt.show()

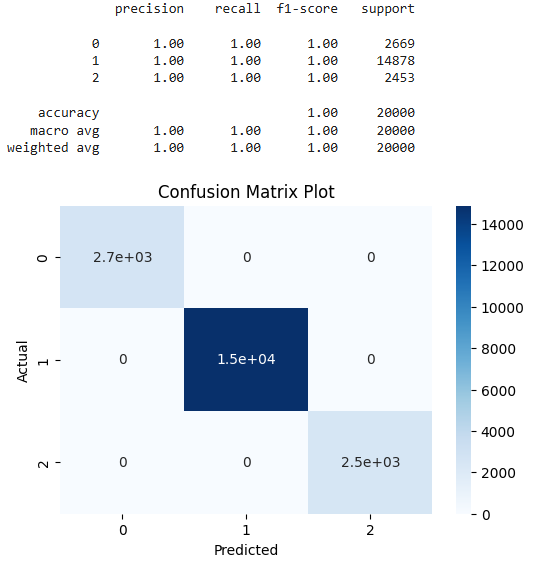


Рисунок 8. Результат обучения модели дерева решений.

**Матрица ошибок** показывает, как модель классифицирует данные, сравнивая предсказанные и фактические классы. Она отображает количество верно и неверно классифицированных объектов по категориям: истинно положительные (TP), истинно отрицательные (TN), ложноположительные (FP) и ложноотрицательные (FN). Матрица ошибок является основой для вычисления метрик, таких как точность, полнота, F1-мера. Анализ матрицы помогает понять сильные и слабые стороны модели, а также выявить проблемы классификации.

**Метрики отчета о классификации:**

* + 1. Precision (Точность):

Доля верно предсказанных положительных результатов среди всех предсказанных положительных результатов. Другими словами, насколько мы можем доверять предсказаниям модели, когда она говорит, что объект принадлежит к данному классу.

Precision = TP/(TP+FP)

* + 1. Recall (Полнота):

Доля верно предсказанных положительных результатов среди всех фактически положительных результатов. Другими словами, насколько хорошо модель способна “находить” все объекты, принадлежащие к данному классу.

Recall = TP / (TP + FN)

* + 1. F1-score (F1-мера):

Среднее гармоническое между точностью и полнотой. F1-score является хорошим показателем, когда требуется сбалансировать точность и полноту.

F1-score = 2 \* (Precision \* Recall) / (Precision + Recall)

* + 1. Support (Количество):

Количество экземпляров каждого класса в тестовых данных.

* + 1. Accuracy (Общая точность):

Доля правильно предсказанных результатов среди всех результатов.

Формула: Accuracy = (TP + TN) / (TP + TN + FP + FN)

* + 1. Macro avg (Макро-усреднение):

Среднее значение каждой метрики (precision, recall, f1-score) по всем классам. Каждый класс имеет одинаковый вес.

* + 1. Weighted avg (Взвешенное усреднение):

Среднее значение каждой метрики по всем классам, взвешенное количеством экземпляров каждого класса.

Анализируя полученные результаты, можно сделать вывод, что наша модель идеально обучилась. Это возможно благодаря тому, что изначальный набор данных был специально создан для обучения классификаторами и идеально размечен.

### Этап 5. Задача NLP

Несмотря на то, что мы получили идеальную модель на предыдущем этапе, попробуем создать модель, используя методы **NLP**, и обучить её.

**NLP** (Natural Language Processing, обработка естественного языка) — это направление в машинном обучении, посвященное распознаванию, генерации и обработке устной и письменной человеческой речи.

Чтобы наша модель могла работать с исходными паролями, их необходимо предобработать. Воспользуемся методом токенизации.

# Разделение данных

X = data['password']

y = data['strength']

# Токенизация паролей

tokenizer = Tokenizer(char\_level=True)  # Работа с символами

tokenizer.fit\_on\_texts(X)

X\_sequences = tokenizer.texts\_to\_sequences(X)

**Токенизатор** (tokenizer) — это инструмент, который разбивает текст на более мелкие единицы, называемые токенами. Токены могут быть словами, частями слов, символами или другими значимыми единицами текста. Токенизаторы являются ключевым компонентом в обработке естественного языка (**NLP**), так как они преобразуют неструктурированный текст в структурированный формат, который может быть обработан компьютером.

С помощью tokenizer приводим пароли в виде списка, где каждый элемент — это целое число.

# Все пароли в виде списков чисел добиваем до длины max\_length

max\_length = max(len(seq) for seq in X\_sequences)

X\_padded = pad\_sequences(X\_sequences, maxlen=max\_length, padding='post'

Далее мы добиваем все полученные токены до одинаковой длины, которая равна максимальной длине токенов. Дополнение длины идет после исходного токена, дополнение идет нулями.

Выравнивание токенов по длине необходимо для более быстрой и точной работы нейронных сетей, которые будут использоваться в обучении модели.

Основой модели является рекуррентная нейронная сеть **LSTM** (Long Short-Term Memory), которая оптимизирована под входные токены одинаковой длины. Эта сеть работает с последовательными данными и решает проблемы «забывания» долгосрочных зависимостей.

Архитектура LSTM основана на таких понятиях, как:

* Ячейка памяти (Cell state): представляет собой “конвейер” информации, проходящий через всю последовательность. Может сохранять информацию на протяжении длительного времени без значительного затухания.
* Вентили (Gates): это нейронные сети, которые управляют потоком информации в ячейку памяти. Состоят из сигмоидных функций, которые выдают значения от 0 до 1, что позволяет “закрывать” или “открывать” проход для информации. Существует три основных вентиля:
  + Вентиль забывания (Forget gate): определяет, какую информацию из старой ячейки памяти нужно отбросить.
  + Вентиль ввода (Input gate): определяет, какую новую информацию нужно добавить в ячейку памяти.
  + Вентиль вывода (Output gate): определяет, какую информацию из ячейки памяти нужно передать в скрытое состояние.
* Скрытое состояние (Hidden state): передает информацию от текущего шага к следующему.

**Принцип работы LSTM:**

1. Ввод данных: LSTM принимает текущий входной вектор и скрытое состояние с предыдущего шага.
2. Вентиль забывания: определяет, какую информацию нужно удалить из старого состояния ячейки.
3. Вентиль ввода: определяет, какую новую информацию нужно добавить в состояние ячейки.
4. Обновление ячейки памяти: новое состояние ячейки формируется на основе старого состояния, вентиля забывания и новой информации.
5. Вентиль вывода: определяет, какая информация из ячейки памяти будет использоваться для формирования нового скрытого состояния.
6. Вывод: формируется новое скрытое состояние, которое передается на следующий шаг.
7. Повторение: Этот процесс повторяется для каждого элемента в последовательности.

Так же для нашей модели мы должны использовать входной слой **Embedding**: этот слой преобразует подаваемые на вход токены в вектора фиксированной длины. Этот вектор будет использоваться в дальнейших слоях.

Защитой от переобучения нашей модели будет слой исключения **Dropout**: этот слой случайно исключает заданную долю нейронов на шаге обучения.

И для того, чтобы наша модель являлась классификатором, нам необходимо использовать жесткий полносвязный слой **Dense**: указанное количество нейронов для этого слоя и будет количеством определяемых классов. На выходе из этого слоя мы получаем вероятности принадлежности к тому или иному классу.

**Общий принцип работы модели:**

* + 1. Embedding: преобразует входные последовательности токенов в плотные векторные представления.
    2. Первый LSTM: обрабатывает последовательности и выдает на выходе всю последовательность скрытых состояний, которые используются для дальнейшей обработки.
    3. Первый Dropout: уменьшает переобучение.
    4. Второй LSTM: обрабатывает последовательности, полученные из предыдущего слоя, и возвращает только последнее скрытое состояние.
    5. Второй Dropout: еще один слой Dropout для снижения переобучения.
    6. Dense (softmax): выполняет классификацию на основе полученных скрытых состояний и выдает вектор вероятностей для каждого класса.

Программная реализация модели:

def create\_model(tokenizer,max\_length):

# Создаем модель

model = Sequential([

Embedding(input\_dim=len(tokenizer.word\_index) + 1, output\_dim=64),

LSTM(64, return\_sequences=True), # рекуррентная нейронная сеть, выход 64 нейрона

Dropout(0.5), # отбрасываем половину нейронов, исключаем переобучение

LSTM(32), #  возвращаем только выходные данные для последнего временного шага

Dropout(0.5),

Dense(3, activation='softmax') # полносвязный слой, 3 - количество нейронов (равно кол-ву наших классов)

])

# подготавливаем модель к обучению

model.compile(optimizer='adam',loss='categorical\_crossentropy', metrics=['accuracy'])

return model

Компилируем нашу модель, применяя оптимизатор Adam, функцию потерь для многоклассовой классификации и выводя метрики точности.

**Adam** — это популярный алгоритм оптимизации, который является адаптивным методом, основанным на градиентном спуске. Он адаптирует скорость обучения для каждого параметра нейросети на основе прошлых градиентов. Это означает, что отдельные веса могут двигаться быстрее или медленнее, в зависимости от их истории обучения.

Функция **categorical\_crossentropy** использует вероятности, полученные с помощью функции активации **softmax** в последнем слое нейросети. Она измеряет разницу между истинным распределением вероятностей классов и предсказанным распределением.

Обучаем нашу модель:

model = create\_model(tokenizer,max\_length)

# Обучаем модель

model.fit(X\_train, y\_train, epochs=3, batch\_size=32, validation\_split=0.2)

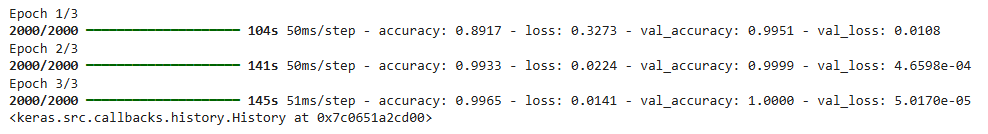
****

Рисунок 8. Обучение модели методом NLP.

По итогу обучения можем наблюдать, что значение точности стремится к 100%, что означает замечательное обучение модели. Минусом этой модели является то, что она в разы дольше обучается и в разы больше потребляет ресурсов по сравнению с прошлыми моделями.

Также выведем отчет о классификации и матрицу ошибок:

# Прогнозирование

y\_pred = np.argmax(model.predict(X\_test), axis=1)

y\_true = np.argmax(y\_test, axis=1)

print(classification\_report(y\_true, y\_pred))

conf\_matrix = confusion\_matrix(y\_true, y\_pred)

plt.figure(figsize=(6, 4))

sns.heatmap(conf\_matrix, annot=True, cmap="Blues")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix Plot")

plt.show()

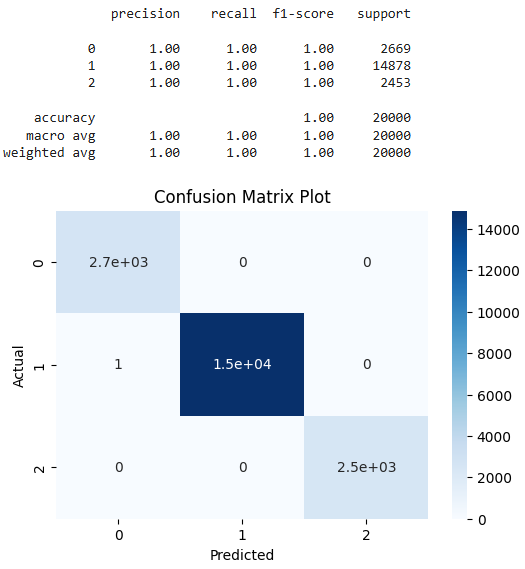


Рисунок 9. Результат обучения модели методом NLP.

Созданной нами моделью допущена всего лишь одна ошибка из 20000 наборов тестовых данных, что не может не радовать. Но, как отмечалось выше, модель на основе подхода NLP обучается намного дольше обычных классификаторов. Можно сказать, что этот метод слишком сложен и избыточен для стоящей перед нами задачи.

# ЗАКЛЮЧЕНИЕ

На основе проведенного анализа можно сделать вывод, что в первом нашем способе наиболее важными признаками для обучения являются длина пароля и количество заглавных букв. Обучив модели на этих двух признаках, мы получили идеальные метрики (близкие или равные единице). Для нашей задачи наиболее подошла модель дерева решений. Она показала 100% точность и наименьшее время обучения (меньше 1 секунды).

Также мы решили попробовать решить нашу задачу с помощью методов NLP. Мы смогли подобрать такие параметры и слои для обучения модели, что точность модели также стремится к 100%. Но при этом модель обучается в несколько раз дольше (порядка 5 минут) в сравнении с обычными классификаторами (меньше 10 секунд). Это означает, что этот способ имеет место быть, но он является более сложным, медленным и избыточным для нашей задачи.

Таким образом, поставленная цель была достигнута. Была выполнена задача классификации паролей по их сложности несколькими методами.

Для достижения цели были решены следующие задачи:

1. Выполнен анализ проблемы и обоснована ее актуальность.
2. Осуществлена загрузка данных и подготовка их к анализу.
3. Выполнен предварительный анализ данных, в том числе корреляционный анализ.
4. Осуществлено моделирование зависимости целевого признака от остальных с помощью различных классификаторов, в том числе подобрана наилучшая модель, оценено ее качество и выполнено прогнозирование.
5. Осуществлено моделирование с помощью методов NLP и оценено её качество и выполнено прогнозирование.
6. Выбрана лучшая модель и сделаны выводы о достижении цели.

Построенная модель классификации «дерево решений» точно прогнозирует класс сложности паролей, что может быть полезно в любых системах, где используется аутентификация и авторизация.

# СПИСОК ЛИТЕРАТУРЫ

1. Password Security: Sber Dataset <https://www.kaggle.com/datasets/morph1max/password-security-sber-dataset>
2. Документация по библиотеке scikit-learn <https://scikit-learn.org/stable/api/sklearn.html>
3. Учебные пособия по машинному обучению <https://colab.research.google.com/drive/1YIOGf38skLQ-RX5TFojpyXdaBy9NlvlF?usp=sharing>
4. Документация по библиотеке TensorFlow <https://www.tensorflow.org/api_docs/python/tf>

# ПРИЛОЖЕНИЕ

Программный код

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.utils import resample

import re

from sklearn.model\_selection import train\_test\_split

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import StratifiedKFold

from sklearn.model\_selection import KFold

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn.metrics import accuracy\_score

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from imblearn.under\_sampling import RandomUnderSampler

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.preprocessing.text import Tokenizer

from sklearn.preprocessing import LabelEncoder

from imblearn.pipeline import Pipeline

import warnings

warnings.filterwarnings("ignore")

def add\_features(df):

    df['length'] = df['password'].apply(len)

    df['special\_chars'] = df['password'].apply(lambda x: len(re.findall(r'[\W]', x)))

    df['uppercase'] = df['password'].apply(lambda x: len(re.findall(r'[A-Z]', x)))

    df['lowercase'] = df['password'].apply(lambda x: len(re.findall(r'[a-z]', x)))

    df['digits'] = df['password'].apply(lambda x: len(re.findall(r'[0-9]', x)))

    # Максимальная длина последовательных символов одной категории

    def max\_sequence\_length(password):

        max\_length = 0

        current\_length = 1

        for i in range(1, len(password)):

            if (password[i].islower() and password[i-1].islower()) or \

               (password[i].isupper() and password[i-1].isupper()) or \

               (password[i].isdigit() and password[i-1].isdigit()) or \

               (not password[i].isalnum() and not password[i-1].isalnum()):

                current\_length += 1

            else:

                max\_length = max(max\_length, current\_length)

                current\_length = 1

        max\_length = max(max\_length, current\_length)  # Проверка на конец строки

        return max\_length

    df['max\_consecutive'] = df['password'].apply(max\_sequence\_length)

    def unique\_chars(password):

        # Находим уникальные символы

        unique\_characters = set(password)

        # Вычисляем соотношение уникальных символов к общему количеству символов

        uniq = len(unique\_characters) / len(password)

        return uniq

    df['uniq\_chars\_%'] = df['password'].apply(unique\_chars)

    #Загружаем список популярных паролей и сравниваем с нашими из датасета

    with open('worst\_passwords.txt', 'r') as file:

        typical\_passwords = {line.strip() for line in file}

    #Проверяем, является ли пароль популярным

    df['pass\_is\_popular?'] = df['password'].apply(lambda x: 1 if x in typical\_passwords else 0)

    return df

data = pd.read\_csv('passwords.csv')

print('Исходный датасет')

print(data.head(),'\n')

print('Информация об датасете')

print(data.info())

# Строим гистограмму целевой переменной 'strength'

plt.figure(figsize=(8, 5))

data['strength'].value\_counts().plot(kind='bar')

plt.title('Распределение классов')

plt.xlabel('Класс')

plt.ylabel('Количество')

plt.xticks(ticks=[0,1,2], labels=[1,0,2], rotation=0)

# Показываем график

plt.show()

data['strength'].value\_counts()

new\_data = add\_features(data)

print('Новый датасет')

print(new\_data.head(),'\n')

new\_data[new\_data['pass\_is\_popular?']==1]['password']

plt.figure(figsize=(7,6))

sns.heatmap(new\_data.corr(numeric\_only = True,method='pearson'), annot=True)

plt.show()

# Выбираем следующие классификаторы:

tuned\_classifiers = {

    "LogisiticRegression": LogisticRegression(random\_state=42),

    "KNearest": KNeighborsClassifier(n\_neighbors=5),

    "DecisionTreeClassifier": DecisionTreeClassifier(random\_state=42),

    "Support Vector Classifier": SVC()

}

# Определяем признаки и целевую переменную из несбалансированного набора данных

X = new\_data[['length','uppercase']]  # Признаки для обучения

y = new\_data['strength']  # Целевая переменная

original\_Xtrain, original\_Xtest, original\_ytrain, original\_ytest = train\_test\_split(X, y, test\_size=0.2,

                                                                                    random\_state = 42)

# Используем StratifiedKFold для обеспечения стратифицированной кросс-валидации

# Это объект кросс-валидации, который позволяет сохранить процент образцов для каждого класса в каждом фоне.

sss = StratifiedKFold(n\_splits=5, random\_state=None, shuffle=False)  # 5 образцов для кросс-валидации

# Создаем объект для уменьшения выборки для большинства

under = RandomUnderSampler()

import time

# Итерируемся по словарю с настроенными классификаторами

for key, classifier in tuned\_classifiers.items():

    time1 = time.time()

    # Создаем список шагов для пайплайна машинного обучения

    steps = [ ('u', under), ('model', classifier)]  # Шаги: увеличенная выборка, уменьшенная выборка и модель

    # Создаем пайплайн машинного обучения

    FiPipeline = Pipeline(steps=steps)  # Инициализация пайплайна с указанными шагами

    # Кросс-валидация классификаторов с использованием определенного пайплайна

    scores = cross\_val\_score(FiPipeline, original\_Xtrain, original\_ytrain, cv=sss)  # Оценка модели с помощью кросс-валидации

    time2 = time.time()

    # Выводим средний коэффициент точности

    print(f"Название: {key}, оценка точности: {round(scores.mean() \* 100, 2)} %, время обучения : {(time2-time1):0.2f} секунд")  # Печать результатов

# Создание модели

model = Pipeline([

    ('u', RandomUnderSampler()),  # Шаг для уменьшения выборки большинства

    ('classification', DecisionTreeClassifier()) ])

model.fit(original\_Xtrain, original\_ytrain)  # Обучаем модель

smote\_prediction = model.predict(original\_Xtest)

# Вывод метрик

labels = ['0', '1','2']

print(classification\_report(original\_ytest, smote\_prediction, target\_names=labels))

conf\_matrix = confusion\_matrix(original\_ytest, smote\_prediction)

plt.figure(figsize=(6, 4))

sns.heatmap(conf\_matrix, annot=True, cmap="Blues")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix Plot")

plt.show()

# Разделение данных

X = data['password']

y = data['strength']

# Токенизация паролей

tokenizer = Tokenizer(char\_level=True)  # Работа с символами

tokenizer.fit\_on\_texts(X)

X\_sequences = tokenizer.texts\_to\_sequences(X)

# Все пароли в виде списков чисел добиваем до длины max\_length

max\_length = max(len(seq) for seq in X\_sequences)

X\_padded = pad\_sequences(X\_sequences, maxlen=max\_length, padding='post')

# Преобразуем метки

y\_categorical = to\_categorical(y, num\_classes=3)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_padded, y\_categorical, test\_size=0.2, random\_state=42)

def create\_model(tokenizer):

    # Создаем модель

    model = Sequential([

      Embedding(input\_dim=len(tokenizer.word\_index) + 1, output\_dim=64),

      LSTM(64, return\_sequences=True), # рекуррентная нейронная сеть, выход 64 нейрона

      Dropout(0.5), # отбрасываем половину нейронов, исключаем переобучение

      LSTM(32), #  возвращаем только выходные данные для последнего временного шага

      Dropout(0.5),

      Dense(3, activation='softmax') # полносвязный слой, 3 - количество нейронов (равно кол-ву наших классов)

      ])

    # подготавливаем модель к обучению

    model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

    return model

model = create\_model(tokenizer)

# Обучаем модель

model.fit(X\_train, y\_train, epochs=3, batch\_size=32, validation\_split=0.2)

# Прогнозирование

y\_pred = np.argmax(model.predict(X\_test), axis=1)

y\_true = np.argmax(y\_test, axis=1)

print(classification\_report(y\_true, y\_pred))

conf\_matrix = confusion\_matrix(y\_true, y\_pred)

plt.figure(figsize=(6, 4))

sns.heatmap(conf\_matrix, annot=True, cmap="Blues")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix Plot")

plt.show()

<https://colab.research.google.com/drive/1Qvv_U08GL3GmZQGIynC-RoJw31gBFEtG#scrollTo=vmSCzWIUYpXC> - ссылка на блокнот с кодом в Google Colab.