Минобрнауки России

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Волгоградский государственный технический университет»

Факультет \_Электроники и вычислительной техники\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра \_\_ Электронно-вычислительные машины и системы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовой работе (проекту)**

по дисциплине \_\_Системы обработки больших данных\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

на тему\_ Разведочный анализ данных и машинное обучение на больших данных

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент\_\_\_\_\_Борисов Данил Алексеевич \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество)

Группа\_\_\_\_\_\_САПР-1.3\_\_\_\_\_\_\_

Руководитель работы (проекта) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_П.Д. Кравченя\_\_\_\_

(подпись и дата подписания) (инициалы и фамилия)

Члены комиссии:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись и дата подписания) (инициалы и фамилия)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись и дата подписания) (инициалы и фамилия)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись и дата подписания) (инициалы и фамилия)

Нормоконтролер \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата подписания) (инициалы и фамилия)

Волгоград 2023 г.

Минобрнауки России

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Волгоградский государственный технический университет»

Факультет \_\_\_ Электроники и вычислительной техники\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Направление (специальность)\_\_Информатика и вычислительная техника\_\_\_\_\_\_  
Кафедра \_\_ Электронно-вычислительные машины и системы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дисциплина\_\_ Системы обработки больших данных \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |
| --- | --- |
|  | Утверждаю  Зав. кафедрой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | «\_\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 \_\_\_ г. |

**Задание**

**на курсовую работу (проект)**

Студент\_\_\_\_ Борисов Данил Алексеевич \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия, имя, отчество)

Группа\_\_\_\_\_ САПР-1.3\_\_\_\_\_\_\_\_\_\_

1. Тема: Разведочный анализ данных и машинное обучение на больших данных \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Утверждена приказом от «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г. № \_\_\_\_\_\_\_\_\_

2. Срок представления работы (проекта) к защите «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ г.

3. Содержание расчетно-пояснительной записки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_1. Разведочный анализ данных с помощью API PySpark \_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_2. Машинное обучение с помощью библиотеки SparkML\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Перечень графического материала: \_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Дата выдачи задания «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 \_\_\_ г.

Руководитель работы (проекта)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_П.Д. Кравченя\_\_\_

подпись, дата инициалы и фамилия

Задание принял к исполнению\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_Д.А. Борисов\_\_\_

подпись, дата инициалы и фамилия

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc156280867)

[1 Разведочный анализ данных с помощью API PySpark 5](#_Toc156280868)

[Характеристика данных 5](#_Toc156280869)

[Постановка задач 6](#_Toc156280870)

[Типы признаков в датасете 7](#_Toc156280871)

[Выявление пропущенных значений и их устранение 9](#_Toc156280872)

[Определение выбросов и их устранение 10](#_Toc156280873)

[Расчет статистических показателей признаков 12](#_Toc156280874)

[Визуализация распределения наиболее важных признаков 14](#_Toc156280875)

[Корреляции между признаками 18](#_Toc156280876)

[Выводы 19](#_Toc156280877)

[2 Машинное обучение с помощью библиотеки SparkML 21](#_Toc156280878)

[Постановка задач 21](#_Toc156280879)

[Задача регрессии – Linear Regression 21](#_Toc156280880)

[Задача бинарной классификации – RandomForest 25](#_Toc156280881)

[Анализ результатов 30](#_Toc156280882)

[Вывод 31](#_Toc156280883)

[ЗАКЛЮЧЕНИЕ 32](#_Toc156280884)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 33](#_Toc156280885)

[ПРИЛОЖЕНИЕ А — РАЗВЕДОЧНЫЙ АНАЛИЗ ДАННЫХ С ПОМОЩЬЮ API PYSPARK 35](#_Toc156280886)

[ПРИЛОЖЕНИЕ Б – МАШИННОЕ ОБУЧЕНИЕ С ПОМОЩЬЮ БИБЛИОТЕКИ SPARKML 44](#_Toc156280887)

ВВЕДЕНИЕ

Разведочный анализ данных представляет собой важный этап в изучении данных, который помогает нам понять их структуру, выявить закономерности, аномалии, и интересные зависимости.

В ходе курсовой работы мы можем изучить различные шаги разведочного анализа, такие как предварительное изучение данных, обработка пропущенных значений, анализ распределений и корреляций, и многое другое. Мы сможем научиться работать с данными в распределенной среде, использовать возможности параллельных вычислений, и создавать информативные визуализации, чтобы лучше понять наши данные.

Apache Spark — это мощный инструмент для обработки больших объемов данных, который предоставляет возможности распределенных вычислений. Этот инструмент позволяет эффективно обрабатывать и анализировать данные, которые не укладываются в память одного компьютера.

Цель работы на Pyspark заключается в обработке и анализе больших объемов данных с использованием распределенной обработки, которые не могут быть обработаны с помощью обычных методов работы с данными из-за их объема или сложности. В курсовой работе можно выделить два этапа в выполнении целей работы.

Первым этапом является проведение разведочного анализа данных с помощью ApacheSpark, а именно на API PySpark, и подготовка датасета для дальнейшей работы с ним.

Вторым этапом является машинное обучение подготовленного датасета из первого этапа.

1. Разведочный анализ данных с помощью API PySpark

Характеристика данных

Для курсовой работы был выбран вариант с постами из социальной сети Twitter (на данный момент носящей наименование 𝕏), а именно датасет под названием «china\_082019\_3\_tweets\_csv\_hashed\_part2.csv». Он представляет из себя небольшой объём данных постов и пользователей, собранных из китайского сообщества за август 2019 года, хотя и является всего лишь частью, на что намекает заключительная часть в названии csv файла.

Данный файл был выбран из-за его обширности и размера более 1GB, что выделяло его на фоне аналогичных датасетов, собранных в китайском сообществе.

Выбранный датасет имел 31 колонку, многие из которых, по моему мнению, являлись бесполезными при разведочном анализе данных. Было решено удалить колонки, которые не представляли в себе интереса для изучения. В основном в датасете сохранились количественные признаки, такие как:

1. «tweetid» – уникальное значение для поста;
2. «userid» – уникальное значение для каждого пользователя;
3. «follower\_count» – количество подписчиков для каждого пользователя;
4. «following\_count» – количество подписок для каждого пользователя;
5. «quote\_count» – количество цитирования определённого поста;
6. «reply\_count» – количество ответов определённого поста;
7. «like\_count» – количество лайков определённого поста;
8. «retweet\_count» – количество ретвитов определённого поста.

Также в датасете сохранился категориальный признак, такой как «tweet\_language», который явно поможет дополнить картину разведочного анализа данных за счёт сбора информации о языке, на котором был написан тот или иной пост из Китая.

В заключении относительно характеристики данных, представляется важным упомянуть, что дополнительный интерес в рамках разведочного анализа может вызывать анализ бинарных признаков. В исходном наборе данных выделяется только один такой признак, а именно «is\_retweet», который указывает на факт принадлежности данного поста к категории ретвитов, что, в свою очередь, может оказать влияние на восприятие информации, содержащейся в посте.

Стоит упомянуть о ещё нескольких бинарных признаках, которые были добавлены «вручную» для расширения восприятия заданного набора данных. К таким признакам относятся:

1. «is\_like» – признак, означающий наличие одного лайка или более на посте;
2. «is\_en\_language» – признак, означающий то, что пост был написан на английском языке;
3. «is\_popular» – признак, означающий то, что у конкретного пользователя имеется более 15 тыс. подписчиков.

Постановка задач

Необходимо провести разведочный анализ датасета с определением:

1. типов признаков в датасете;
2. пропущенных значений и их устранением;
3. выбросов и их устранением;
4. расчетом статистических показателей признаков;
5. визуализацией распределения наиболее важных признаков;
6. корреляций между признаками.

Типы признаков в датасете

Для дальнейшего анализа и использования данных признаков важно выявить их типы, что позволит обнаружить пропущенные значения, выбросы, провести статистический анализ, а также построить визуализации для наглядного представления данных и определения корреляций.

В заданном наборе данных все колонки имели тип данных «string», что явно не отвечало названиям колонок и данных, что в них содержались.

df = df.withColumn("tweetid", col("tweetid").cast(LongType()))

df = df.withColumn("userid", col("userid").cast(LongType()))

df = df.withColumn("follower\_count", col("follower\_count").cast(IntegerType()))

df = df.withColumn("following\_count", col("following\_count").cast(IntegerType()))

df = df.withColumn("tweet\_language", col("tweet\_language").cast(StringType()))

df = df.withColumn("quote\_count", col("quote\_count").cast(IntegerType()))

df = df.withColumn("reply\_count", col("reply\_count").cast(IntegerType()))

df = df.withColumn("like\_count", col("like\_count").cast(IntegerType()))

df = df.withColumn("retweet\_count", col("retweet\_count").cast(IntegerType()))

df = df.withColumn("is\_retweet", col("is\_retweet").cast("boolean"))

df = df.withColumn("is\_like", F.when(df["like\_count"] > 0, True).otherwise(False).cast("boolean"))

df = df.withColumn("is\_en\_language", F.when(df["tweet\_language"] == "en", True).otherwise(False).cast("boolean"))

df = df.withColumn("is\_popular", F.when(df["follower\_count"] > 15000, True).otherwise(False).cast("boolean"))

На рисунке 1 показаны типы данных каждой колонки в заданном наборе данных.

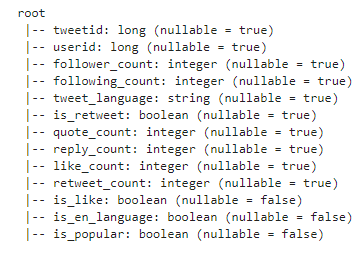


Рисунок 1 — Типы данных каждой колонки

Стоит также обратить внимание на рисунок 2. На нём отлично видно, что колонки, ранее обозначенные как количественные, категориальные и бинарные, представляют из себя именно такой тип данных.

Колонки с количественным типом данных имеют в своих границах числовые значения.

Колонка с категориальным типом данных имеет краткую запись языка, на котором был написан пост.

Колонки с бинарным типом данных имеют в своих границах лишь два бинарных значения: true и false.

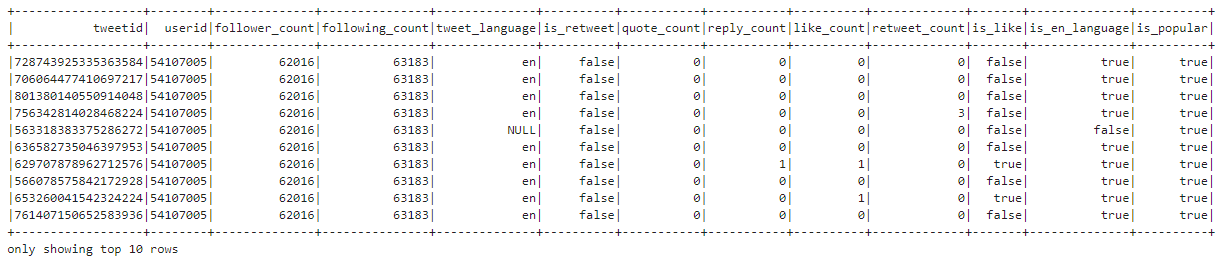


Рисунок 2 — Первые 10 строк датасета

Выявление пропущенных значений и их устранение

Для корректной работы с наборами данных и их дальнейшем анализе, необходимо выявить пропущенные значения, которые могли образоваться в датасете по различным причинам. Для начала мы выявим количество таких значений.

for index, column in enumerate(df.columns):

if column in string\_columns:

missing\_count = df.filter(col(column).eqNullSafe(None) | col(column).isNull()).count()

missing\_values.update({column:missing\_count})

if column in numeric\_columns:

missing\_count = df.where(col(column).isin([None,np.nan])).count()

missing\_values.update({column:missing\_count})

if column in binary\_columns:

missing\_count = df.where(col(column).isNull()).count()

missing\_values.update({column:missing\_count})

return pd.DataFrame.from\_dict([missing\_values])

На рисунке 3 можно увидеть количество пропущенных значений для каждой колонки.

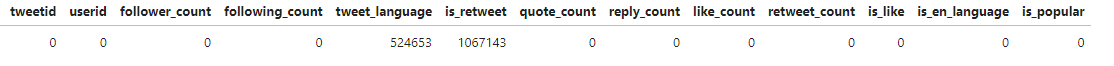


Рисунок 3 — Количество пропущенных значений

Так как такие данные могут помешать адекватному анализу данных, то было принято решение об их удалении из датасета.

df = df.dropna()

Как можем заметить на рисунке 4, все пропущенные значения были удалены.

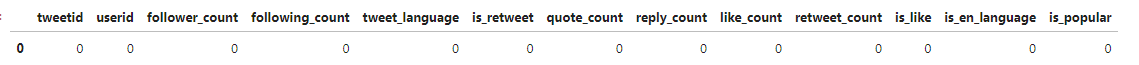


Рисунок 4 — Количество пропущенных значений после их удаления

Определение выбросов и их устранение

Определение выбросов будет происходить с помощью метода межквартильного интервала, который представляет собой вычисление 25-го и 75-го процентиля, и их последующая разница (IQR).

Нижней границей для определения выбросов будет являться значение, равное 25-му процентилю минус IQR, умноженная на 1,5.

Верхней границей для определения выбросов будет являться значение, равное 75-му процентилю плюс IQR, умноженная на 1,5.

quantiles = df.approxQuantile(column, [0.25, 0.75], 0) # 25-й и 75-й процентили

IQR = quantiles[1] - quantiles[0]

lower\_bound = quantiles[0] - 1.5 \* IQR

upper\_bound = quantiles[1] + 1.5 \* IQR

df = df.filter((col(column) >= lower\_bound) & (col(column) <= upper\_bound))

Данные, полученные в диапазоне lower\_bound и upper\_bound, остаются нетронутыми. Те данные, что остались за пределами диапазона, удаляются.

Было принято решение отфильтровать значения из колонок «follower\_count» и «following\_count», так как именно эти колонки, после удаления выбросов, показывали адекватную реакцию и не удаляли бо́льшую часть датасета.

На рисунках 5–8, мы можем наблюдать, как выбросы из обозначенных ранее колонок пропадают из нашего датасета.

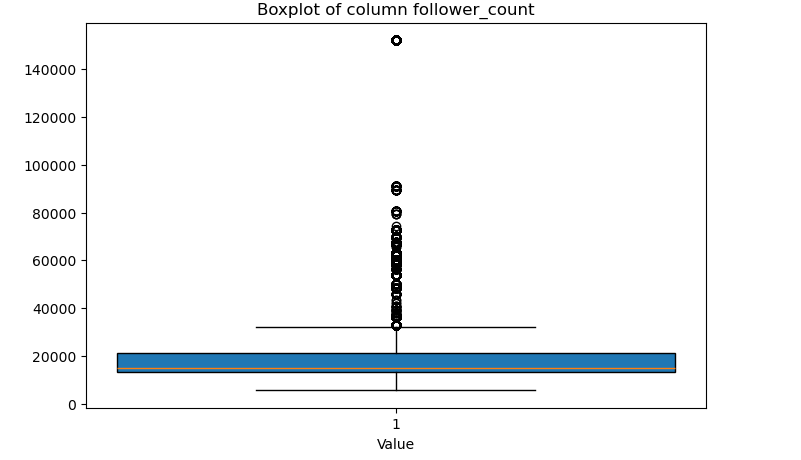


Рисунок 5 — Боксплот для колонки «follower\_count» до удаления выбросов

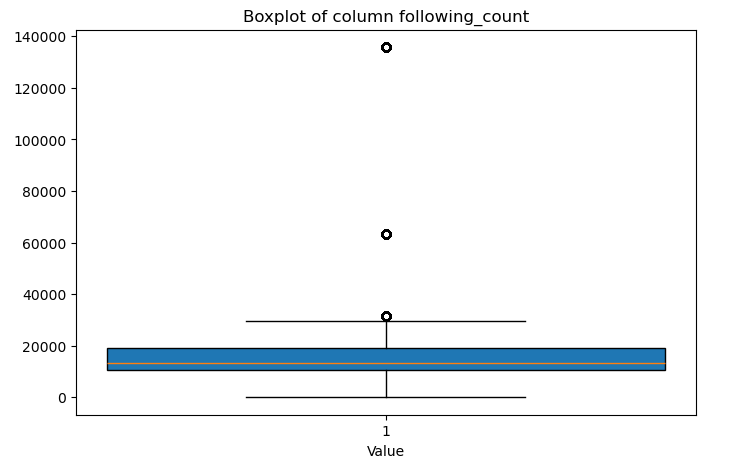


Рисунок 6 — Боксплот для колонки «following\_count» до удаления выбросов

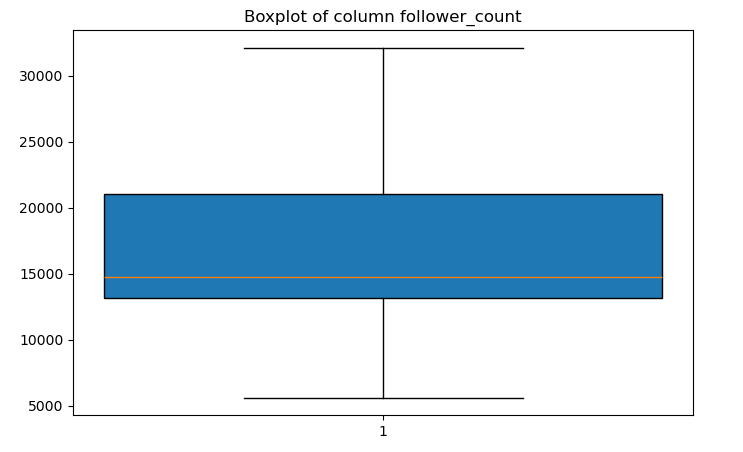


Рисунок 7 — Боксплот для колонки «follower\_count» после удаления выбросов

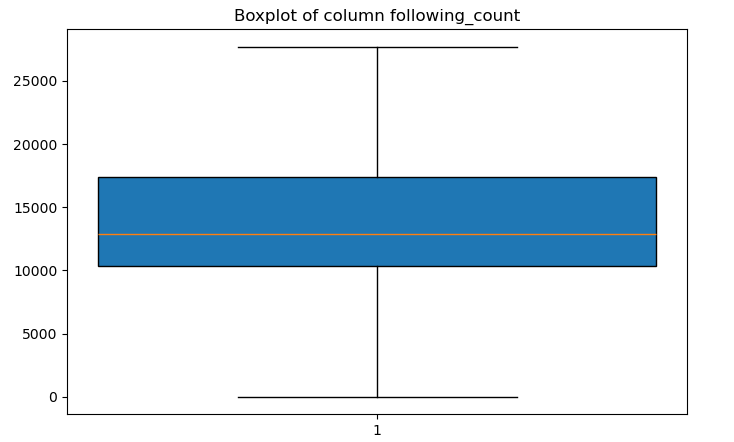


Рисунок 8 — Боксплот для колонки «following\_count» после удаления выбросов

Расчет статистических показателей признаков

Расчёт статистических показателей признаков заключается в подсчёте различной статистики для выбранных колонок из нашего датафрейма.

Для расчёта таких показателей были выбраны колонки с количественными признаками.

Статистические показатели, которые были выбраны:

1. Медиана – представляют собой средние значения в упорядоченном наборе данных;
2. Количество строк – количество значений в колонке;
3. Стандартное отклонение – мера разброса данных относительно среднего значения количественных показателей категориального признака;
4. Минимальное значение – самое минимальное значение для количественного признака;
5. Максимальное значение – самое максимальное значение для количественного признака.

for col\_name in numeric\_columns:

quantiles = df.approxQuantile(col\_name, [0.5], 0.05)

median\_score = quantiles[0]

print(f'Median {col\_name}: {median\_score}\n')

df.select(numeric\_columns).describe().show()

Расчёт всех этих показателей представлен на рисунках 9–10, где не стоит брать во внимание колонки «tweetid» и «userid», которые не представляют из себя чистую количественную информацию.

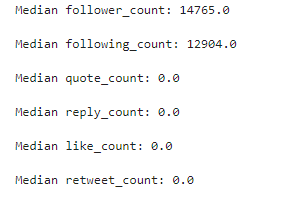


Рисунок 9 — Медианы для количественных признаков

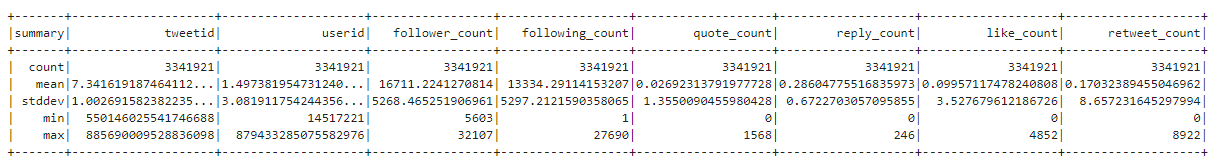


Рисунок 10 — Статистические показатели для количественных признаков

Визуализация распределения наиболее важных признаков

Для визуализации наиболее важных признаков, были выбраны круговые диаграммы, которые смогут отчётливо дать представление о бинарных признаках, а также диаграммы рассеивания, которые помогут нам при рассмотрении взаимосвязей различных количественных признаков.

Визуализация бинарных признаков проводилась с помощью подсчёта количества разных значений в колонке, а после их непосредственного вывода в виде круговой диаграммы. Пример реализации ниже.

count\_isretweet = df.groupBy("is\_retweet").count().collect()

values\_isretweet = [x['count'] for x in count\_isretweet]

categories\_isretweet = [str(x['is\_retweet']) for x in count\_isretweet]

plt.figure(figsize=(12, 12))

plt.subplot(221)

plt.pie(values\_isretweet, labels=categories\_isretweet, autopct='%1.1f%%', colors=['lightcoral', 'lightskyblue'])

plt.title('isRetweet Percentage')

plt.show()

На риснуке 11–12, мы сможем наблюдать сами диаграммы для каждого бинарного признака.

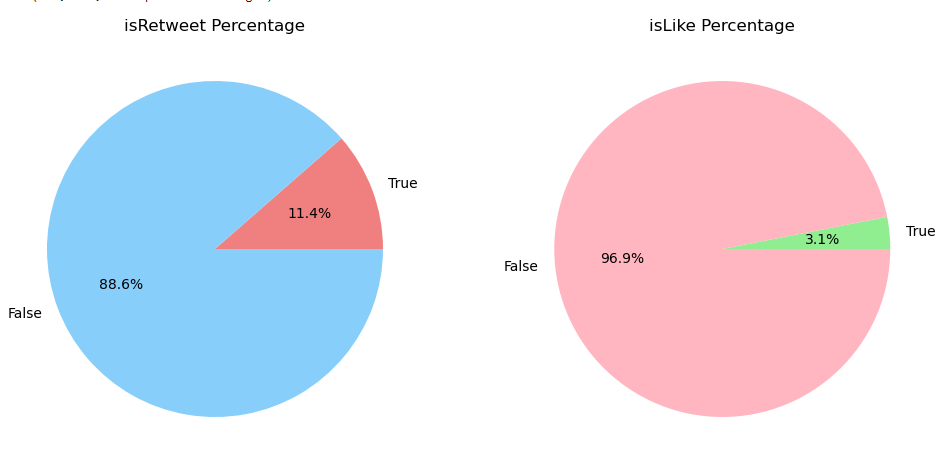


Рисунок 11 — Бинарные признаки, 1

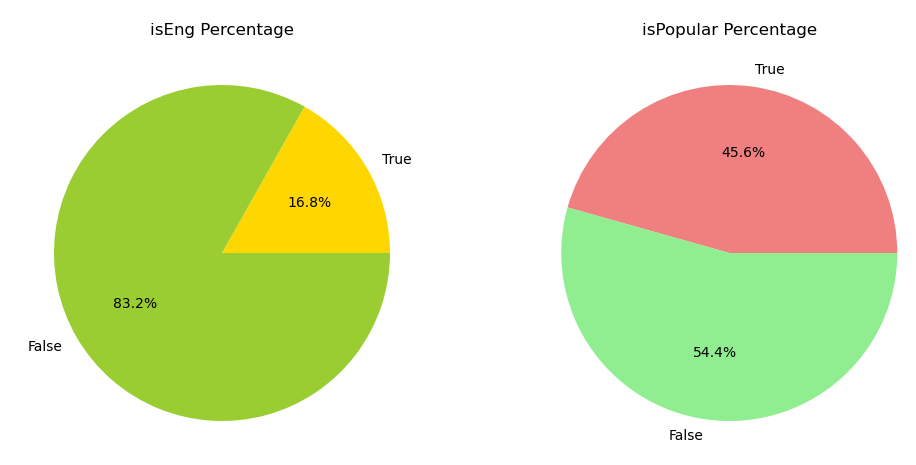


Рисунок 12 — Бинарные признаки, 2

Из рисунка 12, также можно заметить, что бинарное соотношение в колонке «is\_popular» является наиболее сбалансированным в данном датасете.

Визуализация количественных признаков была спроецирована через диаграммы рассеивания по нескольким ключевым соотношениям. Такая диаграмма поможет более детально взглянуть на взаимосвязь некоторых колонок между собой.

Пример реализации такой визуализации представлен ниже.

followers = sample.select('follower\_count').collect()

followings = sample.select('following\_count').collect()

plt.figure(figsize=(11, 7))

plt.scatter([x['follower\_count'] for x in followers], [x['following\_count'] for x in followings], alpha=0.6, color='lightgreen')

plt.title('Following and followers')

plt.xlabel('Count followers')

plt.ylabel('Count followings')

plt.show()

А на рисунках 13–15, собственно, можно более детально рассмотреть взаимосвязи различных колонок.

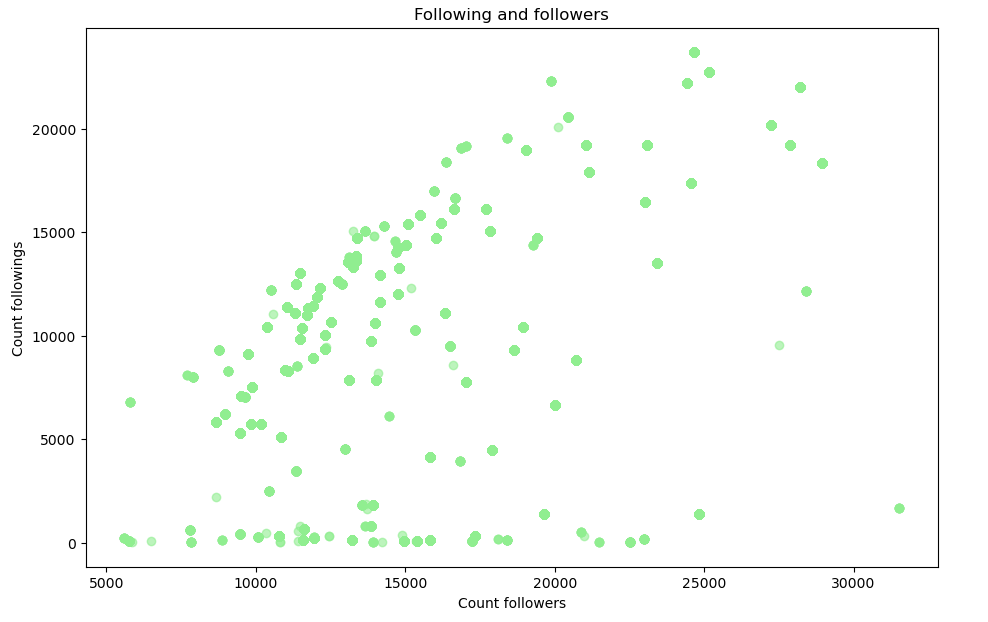


Рисунок 13 — Диаграмма рассеивания «following and followers»

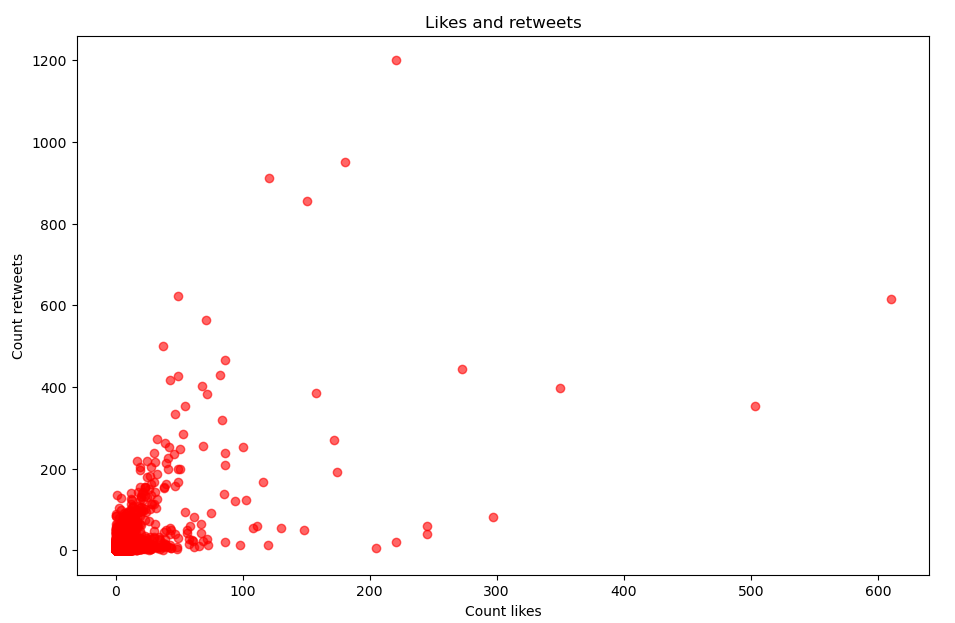


Рисунок 14 — Диаграмма рассеивания «likes and retweets»

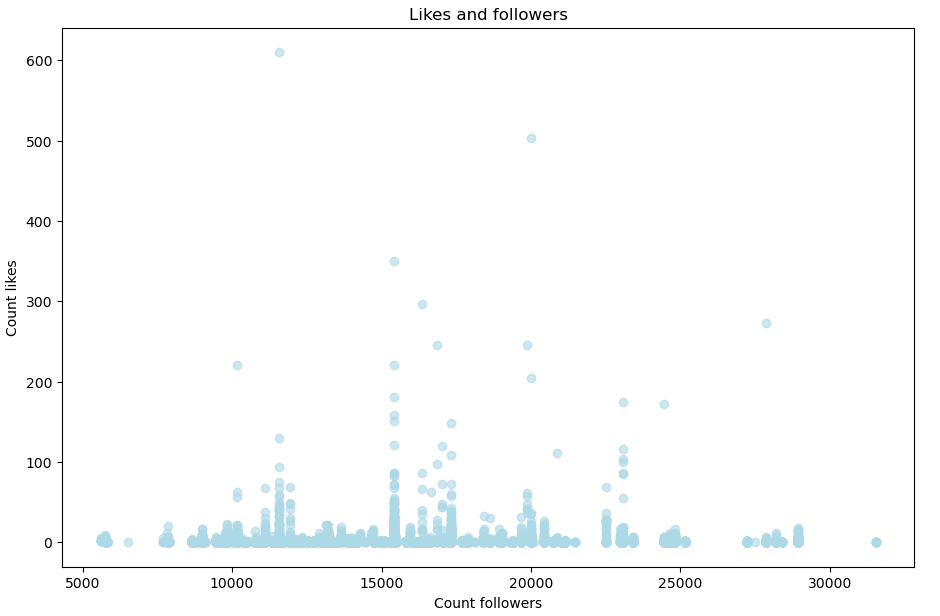


Рисунок 15 — Диаграмма рассеивания «likes and followers»

Корреляции между признаками

Чтобы выявить связи между признаками, нужно построить матрицу корреляций.

Корреляция будет считаться только по количественным признакам, которые имеются в нашем датасете.

Чтобы построить матрицу корреляции, нужно передать в неё нужные колонки. Затем полученный набор данных преобразовывается в список, из которого создаётся новый датафрейм.

Пример построения матрицы корреляции показан ниже.

assembler = VectorAssembler(inputCols=numeric\_columns, outputCol="features")

assembled\_data = assembler.transform(selected\_data)

correlation\_matrix = Correlation.corr(assembled\_data, "features").head()

corr\_matrix = correlation\_matrix[0].toArray()

correlation\_df = spark.createDataFrame(corr\_matrix, numeric\_columns)

Но сама матрица, в её изначальном виде, выглядит непрезентабельно, поэтому было решено построить тепловую карту для более наглядного обзора корреляционных зависимостей.

corr\_matrix\_pd = pd.DataFrame(corr\_matrix, columns=numeric\_columns, index=numeric\_columns)

plt.figure(figsize=(11, 8))

heatmap = sns.heatmap(corr\_matrix\_pd, annot=True, cmap="coolwarm", linewidths=.5)

plt.title('Correlation Heatmap')

plt.show()

На рисунке 16, мы можем наблюдать саму тепловую карту.

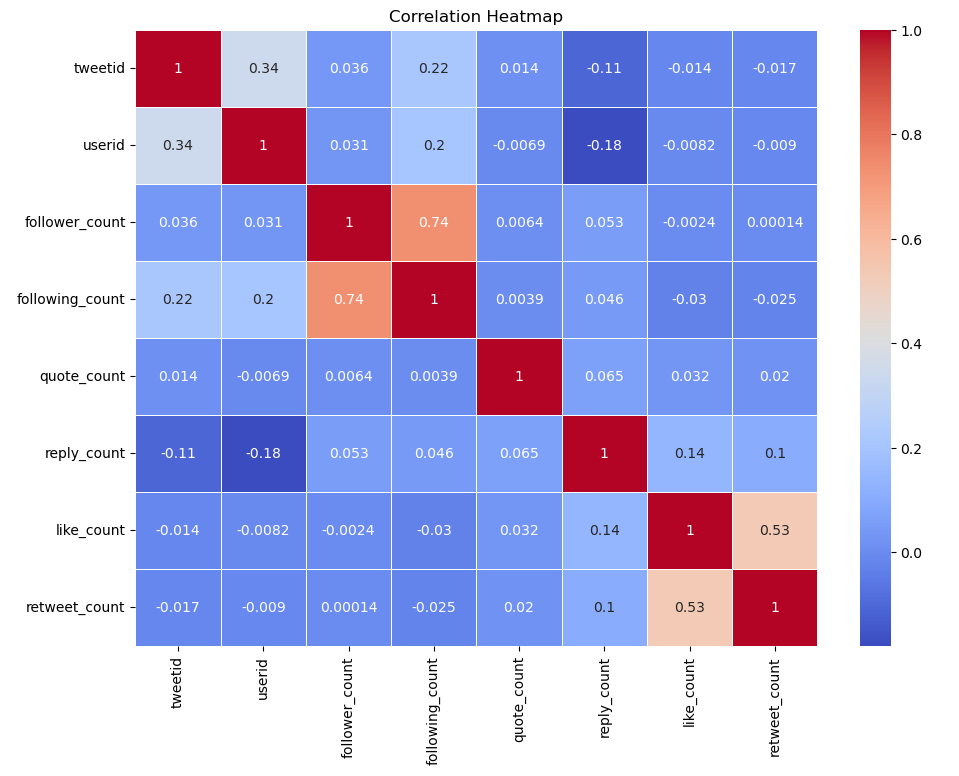


Рисунок 16 — Тепловая карта

Выводы

В процессе исследования был изучен функционал Apache PySpark, освоив его возможности благодаря Python API - PySpark. Основной задачей работы было проведение разведочного анализа датасета, в рамках которого были выполнены следующие шаги:

1. Анализ типов признаков: Определение включенных в датасет данных - числовых, категориальных и бинарных признаков;
2. Обработка пропущенных значений: Обнаружение и удаление строк с пропущенными значениями в данных;
3. Управление выбросами: Применение метода межквартильного интервала для обработки выбросов, с последующей проверкой отсутствия выбросов с использованием boxplot-ов;
4. Вычисление статистических показателей признаков: Расчет средних значений, медиан, квартилей, мод и стандартных отклонений для различных типов признаков с целью лучшего понимания их распределения;
5. Визуализация важных признаков: Построение визуализаций для наглядного анализа распределения ключевых признаков;
6. Изучение корреляций: Проведение анализа взаимосвязей между признаками для выявления степени их взаимосвязи.

Разведочный анализ помогает лучше понять характеристики датасета, готовит данные для построения моделей и выделяет ключевые закономерности, которые могут послужить основой для дальнейшего исследования.

2 Машинное обучение с помощью библиотеки SparkML

Постановка задач

Исходные данные представляют собой отфильтрованный набор постов, содержащий информацию о множестве числовых признаках, включая различные бинарные признаки.

Все эти признаки имеют потенциал использования для прогнозирования значений числовых показателей, а также для задач регрессии и классификации.

Целевая переменная представляет собой числовую характеристику, для которой будет проведено прогнозирование.

Для решения задачи регрессии будет использоваться критерий качества: RMSE (квадратный корень из среднеквадратичной ошибки), MSE (среднее значение квадратов разностей между предсказанными и фактическими значениями), R2 (коэффициент детерминации) и MAE (среднее абсолютное значение разностей между предсказанными и фактическими значениями).

Для задачи бинарной классификации будут использованы следующие критерии качества: Precision, Recall и F1.

По результатам разведочного анализа предпочтительно использовать признаки, проявившие наилучшую корреляцию. Для задачи классификации будет использован бинарный признак.

В итоге необходимо провести обучение и валидацию модели и рассчитать значения метрик классификации и регрессии.

Задача регрессии – Linear Regression

Для задачи регрессии стоит отобрать колонки, которые показали наилучшую корреляцию.

Сама задача регрессии заключается в предсказании значения количественного признака на основе значений других количественных признаков.

Колонки, которые были выбраны для задачи регрессии показаны на рисунке 17.

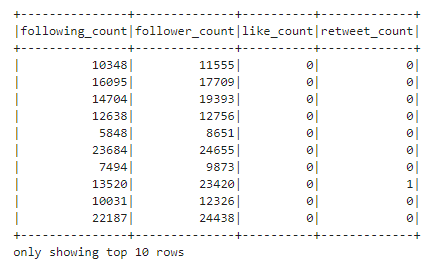


Рисунок 17 — Колонки для задачи регрессии

splits = df.randomSplit([0.8, 0.2])

train = splits[0]

test = splits[1]

train\_rows = train.count()

test\_rows = test.count()

В данном коде мы распределяем выборку на тренировку модели и её тестирование. 80% датафрейма уйдёт на обучение, а 20%, соответственно, на тестирование.

Конвейер представляет собой последовательность шагов преобразования и оценки данных, которые обычно используются для предварительной обработки данных перед моделированием, а затем для обучения прогностической модели.

numVect = VectorAssembler(inputCols = ["like\_count", "following\_count", "retweet\_count"], outputCol="features")

lr = LinearRegression(featuresCol="features", labelCol="follower\_count")

pipeline = Pipeline(stages=[numVect, lr])

model = pipeline.fit(train)

predictions = model.transform(test)

Было решено использовать колонку «follower\_count» для предсказания количества подписчиков, так как при многократном переборе всех доступных колонок, именно она показала наилучшие результаты среди выбранных ранее метрик.

rmse\_evaluator = RegressionEvaluator(labelCol="follower\_count", predictionCol="prediction", metricName="rmse")

mae\_evaluator = RegressionEvaluator(labelCol="follower\_count", predictionCol="prediction", metricName="mae")

mse\_evaluator = RegressionEvaluator(labelCol="follower\_count", predictionCol="prediction", metricName="mse")

r2\_evaluator = RegressionEvaluator(labelCol="follower\_count", predictionCol="prediction", metricName="r2")

rmse = rmse\_evaluator.evaluate(predictions)

mse = mse\_evaluator.evaluate(predictions)

mae = mae\_evaluator.evaluate(predictions)

r2 = r2\_evaluator. evaluate(predictions)

Код, показанный выше, рассчитывает описанные ранее метрики. На рисунке 18, мы можем увидеть значения этих метрик.

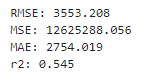


Рисунок 18 — Метрики до кросс-валидации

Результаты не впечатляющие, но как было сказано ранее, именно колонка «follower\_count» показала наилучшие результаты для каждой из метрик.

Полученные данные можно оптимизировать и улучшить показатели метрик с помощью применения параметрической сетки и кросс-валидации.

paramGrid = (ParamGridBuilder()

.addGrid(lr.maxIter, [100, 200, 300])

.addGrid(lr.regParam, [0.1, 0.01])

.build())

Для гиперпараметров выберем maxIter – максимальную итерацию и regParam – параметр регуляризации, который определяет, насколько сильно нужно игнорировать данные.

cv = CrossValidator(estimator=pipeline, \

estimatorParamMaps=paramGrid, \

evaluator=RegressionEvaluator(

predictionCol="prediction", \

labelCol="follower\_count", \

metricName="rmse"), \

numFolds=2)

cv\_model = cv.fit(train)

best\_model = cv\_model.bestModel

best\_predictions = best\_model.transform(test)

Затем передаем эти параметры в метод кросс-валидации в RegressionEvaluator с rmse.

К сожалению, как можно заметить на рисунке 19, методика кросс-валидации не помогла улучшить результаты метрик.

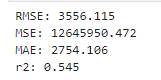


Рисунок 18 — Метрики после кросс-валидации

Задача бинарной классификации – RandomForest

Требуется решить задачу классификации. Суть задачи классификации заключается в предсказании значения бинарного признака на основе значений других количественных признаков. Для решения этой задачи будут использованы бинарные признаки «is\_popular» и «is\_retweet».

df = df\_full.select("following\_count", "follower\_count", "like\_count", "retweet\_count", "is\_popular")

df = df.withColumn("popular", when(col("is\_popular") == True, 1).otherwise(0))

df = df.drop('is\_popular')

Добавляем в наш датафрейм новую колонку popular, для которой 1 = True, а 0 = False.

true\_count = df.filter(df["popular"] == 1).count()

false\_count = df.filter(df["popular"] == 0).count()

true\_percentage = (true\_count / total\_count) \* 100

false\_percentage = (false\_count / total\_count) \* 100

В приведённом выше коде, мы выясняем какое процентное соотношение у нашего бинарного признака.

Процентное соотношение показано на рисунке 19. Из этого можно сделать вывод, что датасет сбалансирован, так как значения True и False не слишком рознятся в общем количестве.



Рисунок 19 — Процентное соотношение в бинарном признаке «is\_popular»

Далее, мы снова обучаем модель по тому же принципу, но уже по способу «случайного леса».

rf = RandomForestClassifier(labelCol="popular", featuresCol="features")

pipeline = Pipeline(stages=[numVect, rf])

model = pipeline.fit(train)

predictions = model.transform(test)

predicted = predictions.select("features", "prediction", "popular")

На рисунке 20, можно увидеть тестовые предсказания для нашей модели.

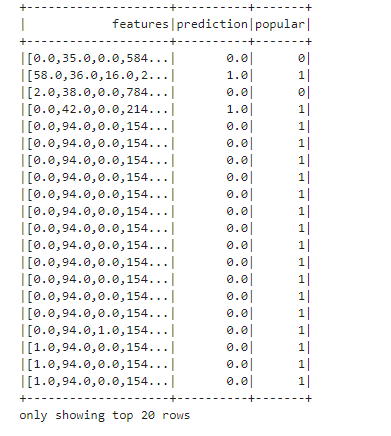


Рисунок 20 — Тестовые предсказания для колонки «popular»

Для оценки модели классификации используются следующие метрики:

1. True Positive (TP) - количество истинно положительных результатов;
2. False Positive (FP) - количество ложноположительных результатов;
3. True Negative (TN) - количество истинно отрицательных результатов;
4. False Negative (FN) - количество ложноотрицательных результатов;
5. Precision (точность) - отношение TP к сумме TP и FP. Оценивает, насколько много из предсказанных моделью положительных случаев действительно являются положительными;
6. Recall (полнота) - отношение TP к сумме TP и FN. Оценивает, насколько много истинных положительных случаев было правильно предсказано моделью;
7. F1-мера - среднее между точностью и полнотой.

tp = float(predicted.filter("prediction == 1.0 AND popular== 1").count())

fp = float(predicted.filter("prediction == 1.0 AND popular== 0").count())

tn = float(predicted.filter("prediction == 0.0 AND popular== 0").count())

fn = float(predicted.filter("prediction == 0.0 AND popular== 1").count())

pr = tp / (tp + fp)

re = tp / (tp + fn)

metrics = spark.createDataFrame([

("TP", tp),

("FP", fp),

("TN", tn),

("FN", fn),

("Precision", pr),

("Recall", re),

("F1", 2\*pr\*re/(re+pr))],["metric", "value"])

metrics.show()

На рисунке 21 можем заметить все метрики, представленные выше.

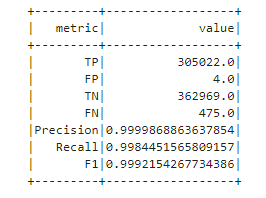


Рисунок 21 — Метрики задачи бинарной классификации

AUC-ROC (Area Under the Receiver Operating Characteristic Curve) — это популярная метрика для оценки производительности моделей классификации. ROC-кривая представляет собой график, отображающий отношение между долей истинно положительных результатов (True Positive Rate) и долей ложноположительных результатов (False Positive Rate) относительно изменяющегося порога классификации.

AUC-ROC напрямую измеряет площадь под этой ROC-кривой. Чем выше AUC-ROC, тем лучше модель способна различить между классами. Значение AUC-ROC находится между 0 и 1, где 1 означает идеальную модель, а 0.5 говорит о случайном предсказании.

Эта метрика особенно полезна при работе с несбалансированными классами, поскольку она позволяет оценить качество модели, учитывая полноту и точность.

evaluator = BinaryClassificationEvaluator(labelCol="popular", rawPredictionCol="rawPrediction", metricName="areaUnderROC")

aur = evaluator.evaluate(predictions)

На рисунке 22, мы можем увидеть значение метрики AUC-ROC.



Рисунок 22 — Значение метрики AUC-ROC задачи бинарной классификации

Модель обучилась с практическими идеальными значениями, что не может не настораживать. Для проверки на исправность и точность нашего кода, была обучена ещё одна модель задачи бинарной классификации. В этот раз мы возьмём колонку «is\_retweet» и проделаем с ней всё то же самое.

На рисунке 23, мы можем увидеть все нужные нам метрики для колонки «is\_retweet».

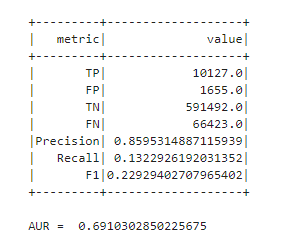


Рисунок 23 — Метрики задачи бинарной классификации для колонки «is\_retweet»

Как можно заметить, модель обучилась неплохо, но не так идеально, как наша предыдущая модель с колонкой «is\_popular», что свидетельствует о том, что код алгоритм работает исправно.

Из-за очень высоких значений метрик в модели с колонкой «is\_popular», было решено не проводить кросс-валидацию по сетке с гиперпараметрами, так как значения и так близки к идеальным. Это свидетельствует о том, что модель обучилась до неприличия хорошо.

Анализ результатов

В результате использования метрик в задаче регрессии были обнаружены параметры, которые недостаточно удовлетворительно оценили производительность модели. Попытки улучшить ситуацию с помощью кросс-валидации не принесли значительного улучшения результатов.

С другой стороны, в задаче классификации были получены весьма убедительные результаты, не требующие дополнительной коррекции через кросс-валидацию. Это свидетельствует о том, что модель классификации успешно справилась с поставленной задачей, и нам не требуется дополнительно корректировать ее на кросс-валидации для дальнейшего улучшения результатов.

Вывод

В процессе исследования функционала Apache PySpark я углубился в изучение библиотеки машинного обучения Apache Spark, освоив все основные моменты с использованием Python API - PySpark. Основной целью работы было проведение машинного обучения на больших объемах данных. Для достижения этой цели я применил два популярных алгоритма машинного обучения: Linear Regression для задачи регрессии и RandomForest для задачи бинарной классификации.

В ходе работы я осуществил обучение и валидацию моделей, вычислил значения метрик классификации и регрессии, а также провел подбор оптимальных гиперпараметров моделей методом перебора по сетке.

ЗАКЛЮЧЕНИЕ

В процессе выполнения курсовой работы мы провели разведочный анализ данных, использовав при этом PySpark, и также провели машинное обучение на больших объемах данных. В ходе исследования мы достигли следующих результатов:

* 1. Определили признаки датасета;
  2. Устранили пропущенные значения и выбросы;
  3. Определили статистические показатели и визуализированы с помощью диаграмм распределения признаки;
  4. Выявили корреляции признаков;
  5. Выполнили обучение и валидация моделей;
  6. Рассчитали значения метрик задач классификации и регрессии.

Поставленная цель работы достигнута, и задачи исследования полностью решены. Результаты работы могут быть приняты для пользования в образовательных целях и личного пользования.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. PySpark Overview – ApacheSpark [электронный ресурс]. – Режим доступа: https://spark.apache.org/docs/latest/api/python/index.html (дата обращения: 09.01.2024).
2. Exploratory Data Analysis (EDA) with PySpark on Databricks [Электронный ресурс]. – [2020]. – Режим доступа: <https://towardsdatascience.com/exploratory-data-analysis-eda-with-pyspark-on-databricks-e8d6529626b1> (дата обращения: 11.01.2024).
3. Advanced Pyspark for Exploratory Data Analysis [Электронный ресурс]. – [2022]. – Режим доступа: <https://www.kaggle.com/code/tientd95/advanced-pyspark-for-exploratory-data-analysis> (дата обращения: 13.01.2024).
4. Exploratory data analysis with pySpark [Электронный ресурс]. – [2020]. – Режим доступа: <https://github.com/roshankoirala/pySpark_tutorial/blob/master/Exploratory_data_analysis_with_pySpark.ipynb> (дата обращения: 14.01.2022).
5. Brahmane, A.V., Krishna, B.C. Big data classification using deep learning and apache spark architecture. Neural Comput & Applic 33, 15253–15266 (2021). https://doi.org/10.1007/s00521-021-06145-w (дата обращения: 14.01.2024).
6. Tinku Singh, Shivam Gupta, Satakshi, Manish Kumar, Performance Analysis and Deployment of Partitioning Strategies in Apache Spark. 2023, Procedia Computer Science, Volume 218, Pages 594-603. DOI: 10.1016/j.procs.2023.01.041. Ссылка: https://www.sciencedirect.com/science/article/pii/S1877050923000418 (дата обращения: 15.01.2024).
7. Salloum, S., Dautov, R., Chen, X. et al. Big data analytics on Apache Spark. Int J Data Sci Anal 1, 145–164 (2016). https://doi.org/10.1007/s41060-016-0027-9 (дата обращения: 15.01.2024).
8. Mostafaeipour, A., Jahangard Rafsanjani, A., Ahmadi, M. et al. Investigating the performance of Hadoop and Spark platforms on machine learning algorithms. J Supercomput 77, 1273–1300 (2021). https://doi.org/10.1007/s11227-020-03328-5 (дата обращения: 15.01.2024).
9. Exploratory Data Analysis (EDA) with PySpark on Databricks [Электронный ресурс]. – [2020]. – Режим доступа: <https://towardsdatascience.com/exploratory-data-analysis-eda-with-pyspark-on-databricks-e8d6529626b1> (дата обращения: 15.01.2024).
10. Изучаем Spark: молниеносный анализ данных / Х. Карау, Э. Конвински, П. Венделл, М.М. Захария // ДМК Пресс, 2015. — 304 с. — (дата обращения: 15.01.2024).

ПРИЛОЖЕНИЕ А — РАЗВЕДОЧНЫЙ АНАЛИЗ ДАННЫХ С ПОМОЩЬЮ API PYSPARK

# Import other modules not related to PySpark

import os

import sys

import pandas as pd

from pandas import DataFrame

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.ticker as mtick

import matplotlib

from mpl\_toolkits.mplot3d import Axes3D

import math

from IPython.core.interactiveshell import InteractiveShell

from datetime import \*

import statistics as stats

# This helps auto print out the items without explixitly using 'print'

InteractiveShell.ast\_node\_interactivity = "all"

%matplotlib inline

# Import PySpark related modules

import pyspark

from pyspark.rdd import RDD

from pyspark.sql import Row

from pyspark.sql import DataFrame

from pyspark.sql import SparkSession

from pyspark.sql import SQLContext

from pyspark.sql.functions import lit, desc, col, size, array\_contains\

, isnan, udf, hour, array\_min, array\_max, countDistinct

from pyspark.sql.types import \*

from functools import reduce

from pyspark.sql import functions

from pyspark.sql.types import \*

from pyspark.sql.functions import length, mean, split, udf, col

from pyspark.sql import functions as F

import pyarrow.parquet as pq

import seaborn as sns

MAX\_MEMORY = '8G'

# Initialize a spark session.

conf = pyspark.SparkConf().setMaster("local[\*]") \

.set('spark.executor.heartbeatInterval', 10000) \

.set('spark.network.timeout', 10000) \

.set("spark.core.connection.ack.wait.timeout", "3600") \

.set("spark.executor.memory", MAX\_MEMORY) \

.set("spark.driver.memory", MAX\_MEMORY)

def init\_spark():

spark = SparkSession \

.builder \

.appName("Lab1") \

.config(conf=conf) \

.getOrCreate()

return spark

spark = init\_spark()

filename\_data = 'data/china\_082019\_3\_tweets\_csv\_hashed\_part2.csv'

df = spark.read.csv(filename\_data,header=True, mode="DROPMALFORMED", inferSchema=True)

print('Data frame type: ' + str(type(df)))

print('Data overview')

df.printSchema()

print('Columns overview')

pd.DataFrame(df.dtypes, columns = ['Column Name','Data type'])

df = df.drop("user\_profile\_description", "account\_creation\_date", "user\_reported\_location", "account\_language", "tweet\_text", "tweet\_time", "tweet\_client\_name", "in\_reply\_to\_userid", "in\_reply\_to\_tweetid", "quoted\_tweet\_tweetid", "retweet\_userid", "retweet\_tweetid", "latitude", "longitude", "user\_mentions", "poll\_choices", "hashtags", "urls", "user\_screen\_name", "user\_display\_name", "user\_profile\_url")

df = df.withColumn("tweetid", col("tweetid").cast(LongType()))

df = df.withColumn("userid", col("userid").cast(LongType()))

df = df.withColumn("follower\_count", col("follower\_count").cast(IntegerType()))

df = df.withColumn("following\_count", col("following\_count").cast(IntegerType()))

df = df.withColumn("tweet\_language", col("tweet\_language").cast(StringType()))

df = df.withColumn("quote\_count", col("quote\_count").cast(IntegerType()))

df = df.withColumn("reply\_count", col("reply\_count").cast(IntegerType()))

df = df.withColumn("like\_count", col("like\_count").cast(IntegerType()))

df = df.withColumn("retweet\_count", col("retweet\_count").cast(IntegerType()))

df = df.withColumn("is\_retweet", col("is\_retweet").cast("boolean"))

df = df.withColumn("is\_like", F.when(df["like\_count"] > 0, True).otherwise(False).cast("boolean"))

df = df.withColumn("is\_en\_language", F.when(df["tweet\_language"] == "en", True).otherwise(False).cast("boolean"))

df = df.withColumn("is\_popular", F.when(df["follower\_count"] > 15000, True).otherwise(False).cast("boolean"))

df.show(10)

df.printSchema()

string\_columns = ['tweet\_language', 'is\_like', 'is\_en\_language', 'is\_popular']

numeric\_columns = ['tweetid','userid','follower\_count','following\_count','quote\_count','reply\_count','like\_count', 'retweet\_count']

binary\_columns = ['is\_retweet']

def view\_missing\_values(string\_columns,numeric\_columns,binary\_columns):

missing\_values = {}

for index, column in enumerate(df.columns):

if column in string\_columns: # check string columns with None and Null values

missing\_count = df.filter(col(column).eqNullSafe(None) | col(column).isNull()).count()

missing\_values.update({column:missing\_count})

if column in numeric\_columns: # check zeroes, None, NaN

missing\_count = df.where(col(column).isin([None,np.nan])).count()

missing\_values.update({column:missing\_count})

if column in binary\_columns: # check zeroes, None, NaN

missing\_count = df.where(col(column).isNull()).count()

missing\_values.update({column:missing\_count})

return pd.DataFrame.from\_dict([missing\_values])

missing\_df = view\_missing\_values(string\_columns,numeric\_columns,binary\_columns)

missing\_df

df.printSchema()

df = df.dropna()

num\_col = ["follower\_count", "following\_count"]

for column in num\_col:

boxplot\_data = df.select(column).collect()

values = [row[column] for row in boxplot\_data]

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

plt.boxplot(values, widths=0.9, patch\_artist=True, showmeans=False, autorange=True)

plt.title(f'Boxplot of column {column}')

plt.xlabel('Value')

plt.show()

for column in num\_col:

quantiles = df.approxQuantile(column, [0.25, 0.75], 0) # 25-й и 75-й процентили

IQR = quantiles[1] - quantiles[0]

lower\_bound = quantiles[0] - 1.5 \* IQR

upper\_bound = quantiles[1] + 1.5 \* IQR

df = df.filter((col(column) >= lower\_bound) & (col(column) <= upper\_bound))

boxplot\_data = df.select(column).collect()

values = [row[column] for row in boxplot\_data]

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

plt.boxplot(values, widths=0.9, patch\_artist=True, showmeans=False, autorange=True)

plt.title(f'Boxplot of column {column}')

plt.show()

for col\_name in numeric\_columns:

quantiles = df.approxQuantile(col\_name, [0.5], 0.05)

median\_score = quantiles[0]

print(f'Median {col\_name}: {median\_score}\n')

df.select(numeric\_columns).describe().show()

count\_isretweet = df.groupBy("is\_retweet").count().collect()

count\_islike = df.groupBy("is\_like").count().collect()

count\_isenlang = df.groupBy("is\_en\_language").count().collect()

count\_ispopular = df.groupBy("is\_popular").count().collect()

values\_isretweet = [x['count'] for x in count\_isretweet]

categories\_isretweet = [str(x['is\_retweet']) for x in count\_isretweet]

values\_islike = [x['count'] for x in count\_islike]

categories\_islike = [str(x['is\_like']) for x in count\_islike]

values\_isenlang = [x['count'] for x in count\_isenlang]

categories\_isenlang = [str(x['is\_en\_language']) for x in count\_isenlang]

values\_ispopular = [x['count'] for x in count\_ispopular]

categories\_ispopular = [str(x['is\_popular']) for x in count\_ispopular]

plt.figure(figsize=(12, 12))

plt.subplot(221)

plt.pie(values\_isretweet, labels=categories\_isretweet, autopct='%1.1f%%', colors=['lightcoral', 'lightskyblue'])

plt.title('isRetweet Percentage')

plt.subplot(222)

plt.pie(values\_islike, labels=categories\_islike, autopct='%1.1f%%', colors=['lightgreen', 'lightpink'])

plt.title('isLike Percentage')

plt.subplot(223)

plt.pie(values\_isenlang, labels=categories\_isenlang, autopct='%1.1f%%', colors=['gold', 'yellowgreen'])

plt.title('isEng Percentage')

plt.subplot(224)

plt.pie(values\_ispopular, labels=categories\_ispopular, autopct='%1.1f%%', colors=['lightcoral', 'lightgreen'])

plt.title('isPopular Percentage')

plt.show()

from pyspark.ml.feature import VectorAssembler

from pyspark.ml.stat import Correlation

selected\_data = df

assembler = VectorAssembler(inputCols=numeric\_columns, outputCol="features")

assembled\_data = assembler.transform(selected\_data)

correlation\_matrix = Correlation.corr(assembled\_data, "features").head()

corr\_matrix = correlation\_matrix[0].toArray()

correlation\_df = spark.createDataFrame(corr\_matrix, numeric\_columns)

correlation\_df.show()

corr\_matrix\_pd = pd.DataFrame(corr\_matrix, columns=numeric\_columns, index=numeric\_columns)

# Построение тепловой карты

plt.figure(figsize=(11, 8))

heatmap = sns.heatmap(corr\_matrix\_pd, annot=True, cmap="coolwarm", linewidths=.5)

plt.title('Correlation Heatmap')

plt.show()

sample = df.sample(False, 0.1, seed=69)

followers = sample.select('follower\_count').collect()

followings = sample.select('following\_count').collect()

likes = sample.select('like\_count').collect()

retweets = sample.select('retweet\_count').collect()

plt.figure(figsize=(11, 7))

plt.scatter([x['follower\_count'] for x in followers], [x['following\_count'] for x in followings], alpha=0.6, color='lightgreen')

plt.title('Following and followers')

plt.xlabel('Count followers')

plt.ylabel('Count followings')

plt.show()

plt.figure(figsize=(11, 7))

plt.scatter([x['like\_count'] for x in likes], [x['retweet\_count'] for x in retweets], alpha=0.6, color='red')

plt.title('Likes and retweets')

plt.xlabel('Count likes')

plt.ylabel('Count retweets')

plt.show()

plt.figure(figsize=(11, 7))

plt.scatter([x['follower\_count'] for x in followers], [x['like\_count'] for x in likes], alpha=0.6, color='lightblue')

plt.title('Likes and followers')

plt.xlabel('Count followers')

plt.ylabel('Count likes')

plt.show()

df.count()

df.write.mode('overwrite').parquet("data/df.parquet")

ПРИЛОЖЕНИЕ Б – МАШИННОЕ ОБУЧЕНИЕ С ПОМОЩЬЮ БИБЛИОТЕКИ SPARKML

# The Jupyter Notebook and dataset were downloaded from https://www.kaggle.com/code/tylerx/machine-learning-with-spark

# This Python 3 environment comes with many helpful analytics libraries installed

# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python

# For example, here's several helpful packages to load in

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

# Input data files are available in the "../input/" directory.

# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

import os

# Any results you write to the current directory are saved as output.

from pyspark.sql.types import \*

from pyspark.sql.functions import \*

from pyspark.sql import SparkSession

from pyspark.ml import Pipeline

from pyspark.ml.feature import VectorAssembler, StringIndexer, VectorIndexer, MinMaxScaler

from pyspark.ml.classification import LogisticRegression

from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

from pyspark.ml.evaluation import BinaryClassificationEvaluator

from pyspark.ml.regression import LinearRegression

from pyspark.ml.evaluation import RegressionEvaluator

from pyspark.ml.regression import RandomForestRegressor

from pyspark.ml.classification import RandomForestClassifier

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

spark = SparkSession.builder.master("local[\*]").getOrCreate()

df\_full = spark.read.parquet("data/df.parquet")

df\_full.show(10)

df = df\_full.select("following\_count", "follower\_count", "like\_count", "retweet\_count")

df.show(10)

# Распределение на обучение

splits = df.randomSplit([0.8, 0.2])

train = splits[0]

test = splits[1]

train\_rows = train.count()

test\_rows = test.count()

print("Training Rows:", train\_rows, " Testing Rows:", test\_rows)

numVect = VectorAssembler(inputCols = ["like\_count", "following\_count", "retweet\_count"], outputCol="features")

lr = LinearRegression(featuresCol="features", labelCol="follower\_count")

pipeline = Pipeline(stages=[numVect, lr])

model = pipeline.fit(train)

predictions = model.transform(test)

rmse\_evaluator = RegressionEvaluator(labelCol="follower\_count", predictionCol="prediction", metricName="rmse")

mae\_evaluator = RegressionEvaluator(labelCol="follower\_count", predictionCol="prediction", metricName="mae")

mse\_evaluator = RegressionEvaluator(labelCol="follower\_count", predictionCol="prediction", metricName="mse")

r2\_evaluator = RegressionEvaluator(labelCol="follower\_count", predictionCol="prediction", metricName="r2")

rmse = rmse\_evaluator.evaluate(predictions)

mse = mse\_evaluator.evaluate(predictions)

mae = mae\_evaluator.evaluate(predictions)

r2 = r2\_evaluator.evaluate(predictions)

print(f"RMSE: {rmse:0.3f}" )

print(f"MSE: {mse:0.3f}" )

print(f"MAE: {mae:0.3f}" )

print(f"r2: {r2:0.3f}" )

predicted = predictions.select("features", "prediction", "follower\_count")

predicted.show(20)

paramGrid = (ParamGridBuilder()

.addGrid(lr.maxIter, [100, 200, 300])

.addGrid(lr.regParam, [0.1, 0.01])

.build())

v = CrossValidator(estimator=pipeline, \

estimatorParamMaps=paramGrid, \

evaluator=RegressionEvaluator(

predictionCol="prediction", \

labelCol="follower\_count", \

metricName="rmse"), \

numFolds=2)

cv\_model = cv.fit(train)

best\_model = cv\_model.bestModel

best\_predictions = best\_model.transform(test)

rmse\_evaluator = RegressionEvaluator(labelCol="follower\_count", predictionCol="prediction", metricName="rmse")

mae\_evaluator = RegressionEvaluator(labelCol="follower\_count", predictionCol="prediction", metricName="mae")

mse\_evaluator = RegressionEvaluator(labelCol="follower\_count", predictionCol="prediction", metricName="mse")

r2\_evaluator = RegressionEvaluator(labelCol="follower\_count", predictionCol="prediction", metricName="r2")

best\_rmse = rmse\_evaluator.evaluate(best\_predictions)

best\_mse = mse\_evaluator.evaluate(best\_predictions)

best\_mae = mae\_evaluator.evaluate(best\_predictions)

best\_r2 = r2\_evaluator.evaluate(best\_predictions)

print(f"RMSE: {best\_rmse:0.3f}" )

print(f"MSE: {best\_mse:0.3f}" )

print(f"MAE: {best\_mae:0.3f}" )

print(f"r2: {best\_r2:0.3f}" )

df = df\_full.select("following\_count", "follower\_count", "like\_count", "retweet\_count", "is\_popular")

df = df.withColumn("popular", when(col("is\_popular") == True, 1).otherwise(0))

df = df.drop('is\_popular')

df.show(10)

df.printSchema()

total\_count = df.count()

true\_count = df.filter(df["popular"] == 1).count()

false\_count = df.filter(df["popular"] == 0).count()

print("True:", true\_count)

print("False:", false\_count)

true\_percentage = (true\_count / total\_count) \* 100

false\_percentage = (false\_count / total\_count) \* 100

print(f"True: {true\_percentage:0.3f}%")

print(f"False: {false\_percentage:0.3f}%")

splits = df.randomSplit([0.8, 0.2])

train = splits[0]

test = splits[1]

train\_rows = train.count()

test\_rows = test.count()

print("Training Rows:", train\_rows, " Testing Rows:", test\_rows)

numVect = VectorAssembler(inputCols = ["like\_count", "following\_count", "retweet\_count", "follower\_count"], outputCol="features")

rf = RandomForestClassifier(labelCol="popular", featuresCol="features")

pipeline = Pipeline(stages=[numVect, rf])

model = pipeline.fit(train)

predictions = model.transform(test)

predicted = predictions.select("features", "prediction", "popular")

predicted.show(20)

evaluator = MulticlassClassificationEvaluator(labelCol="popular", predictionCol="prediction", metricName="accuracy")

tp = float(predicted.filter("prediction == 1.0 AND popular == 1").count())

fp = float(predicted.filter("prediction == 1.0 AND popular == 0").count())

tn = float(predicted.filter("prediction == 0.0 AND popular == 0").count())

fn = float(predicted.filter("prediction == 0.0 AND popular == 1").count())

pr = tp / (tp + fp)

re = tp / (tp + fn)

metrics = spark.createDataFrame([

("TP", tp),

("FP", fp),

("TN", tn),

("FN", fn),

("Precision", pr),

("Recall", re),

("F1", 2\*pr\*re/(re+pr))],["metric", "value"])

metrics.show()

evaluator = BinaryClassificationEvaluator(labelCol="popular", rawPredictionCol="rawPrediction", metricName="areaUnderROC")

aur = evaluator.evaluate(predictions)

print ("AUR = ", aur)

df = df\_full.select("following\_count", "follower\_count", "like\_count", "retweet\_count", "is\_retweet")

df = df.withColumn("retweet", when(col("is\_retweet") == True, 1).otherwise(0))

df = df.drop('is\_retweet')

df.show(10)

df.printSchema()

total\_count = df.count()

true\_count = df.filter(df["retweet"] == 1).count()

false\_count = df.filter(df["retweet"] == 0).count()

print("True:", true\_count)

print("False:", false\_count)

true\_percentage = (true\_count / total\_count) \* 100

false\_percentage = (false\_count / total\_count) \* 100

print(f"True: {true\_percentage:0.3f}%")

print(f"False: {false\_percentage:0.3f}%")

splits = df.randomSplit([0.8, 0.2])

train = splits[0]

test = splits[1]

train\_rows = train.count()

test\_rows = test.count()

print("Training Rows:", train\_rows, " Testing Rows:", test\_rows)

numVect = VectorAssembler(inputCols = ["like\_count", "following\_count", "retweet\_count", "follower\_count"], outputCol="features")

rf = RandomForestClassifier(labelCol="retweet", featuresCol="features")

pipeline = Pipeline(stages=[numVect, rf])

model = pipeline.fit(train)

predictions = model.transform(test)

predicted = predictions.select("features", "prediction", "retweet")

predicted.show(20)

evaluator = MulticlassClassificationEvaluator(labelCol="retweet", predictionCol="prediction", metricName="accuracy")

tp = float(predicted.filter("prediction == 1.0 AND retweet == 1").count())

fp = float(predicted.filter("prediction == 1.0 AND retweet == 0").count())

tn = float(predicted.filter("prediction == 0.0 AND retweet == 0").count())

fn = float(predicted.filter("prediction == 0.0 AND retweet == 1").count())

pr = tp / (tp + fp)

re = tp / (tp + fn)

metrics = spark.createDataFrame([

("TP", tp),

("FP", fp),

("TN", tn),

("FN", fn),

("Precision", pr),

("Recall", re),

("F1", 2\*pr\*re/(re+pr))],["metric", "value"])

metrics.show()

evaluator = BinaryClassificationEvaluator(labelCol="retweet", rawPredictionCol="rawPrediction", metricName="areaUnderROC")

aur = evaluator.evaluate(predictions)

print ("AUR = ", aur)