|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_Информатика и системы управления\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_Системы обработки информации и управления\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ***

***НА ТЕМУ:***

***Построение модели машинного обучения для\_\_\_\_\_\_\_ решения задачи классификации CPU\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

Студент \_\_\_\_ИУ5-34М\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Д.А. Лычагин\_\_\_\_**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_Ю.Е.Гапанюк\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

*2022 г.*

**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

УТВЕРЖДАЮ

Заведующий кафедрой \_\_\_ИУ5\_\_

(Индекс)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **\_В.И. Терехов\_**

(И.О.Фамилия)

« \_\_\_\_\_ » \_\_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_\_ г.

**ЗАДАНИЕ**

**на выполнение научно-исследовательской работы**

по теме \_ Построение модели машинного обучения для решения задачи классификации CPU \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент группы \_\_\_\_ ИУ5-34М\_\_\_\_

\_\_\_Лычагин Дмитрий Андреевич\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)

\_\_\_учебная\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Источник тематики (кафедра, предприятие, НИР) \_\_кафедра\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

График выполнения НИР: 25% к \_4\_ нед., 50% к \_8\_ нед., 75% к \_12\_ нед., 100% к \_17 нед.

***Техническое задание:*** *построить модели машинного обучения для решения задачи классификации CPU \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

***Оформление научно-исследовательской работы:***

Расчетно-пояснительная записка на 22 листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

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

Дата выдачи задания « \_1\_ » \_\_\_октября\_\_\_ 2022 г.

**Руководитель НИР**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **\_Ю.Е. Гапанюк \_\_**

(Подпись, дата) (И.О.Фамилия)

**Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.А. Лычагин\_**

(Подпись, дата) (И.О.Фамилия)

Введение

Задачей научно-исследовательской работы работы являетя построение классифицирующей модели машинного обучения. Модель должна решать задачу определения сегмента рынка CPU к которому относится ислледуемый образец CPU.

Задание буду выполнять на датасете "Computer Parts (CPUs and GPUs)". Датасет содержит значения различных характеристик по CPU GPU. В датасете есть как числовые и категориальные признаки, так и нестандартные признаки. Есть колонки с пропусками, ненормированными значениями. Датасет состоит из двух наборов данных по CPU и по GPU. Входе построения модели будет использоваться набор данных по CPU.

Оглавление

[Введение 3](#_Toc91068752)

[1.Предобработка данных 5](#_Toc91068753)

[1.1 Устанение пропусков 5](#_Toc91068754)

[1.2 Обработка не стандартных признаков 8](#_Toc91068755)

[1.3 Масштабирование признаков 12](#_Toc91068756)

[1.4 Кодирование категориальных значений 16](#_Toc91068757)

[1.5 Устраниение выбросов в данных 17](#_Toc91068758)

[2. Построение модели машинного обучения. 19](#_Toc91068759)

[2.1 Отбор признаков 19](#_Toc91068760)

[2.2 Обучение и тестирование модели 20](#_Toc91068761)

[3. Список литературы 22](#_Toc91068762)

1.Предобработка данных

Предварительная обработка и очистка данных должны проводиться до того, как набор данных будет использоваться для обучения модели. Необработанные данные зачастую искажены и ненадежны, и в них могут быть пропущены значения. Использование таких данных при моделировании может приводить к неверным результатам. Исследуем датасет на наличие не нормированных значений, пропусков, категориальных признаковю

1.1 Устанение пропусков

Часть признаков данном датасете содержит пропуски. Наличие пропусков в данных недпустимо при обучении большинства моделей машинного обучения, а также построении корреляционной матрицы. Одна из стратегий устранения пропусков, предполагает удаление признаков целиком если число пропусков велико – например более 30%. Проверим датасет на наличие пропусков и удалим слишком разряженные признаки:

**def** get\_missing\_columns(dataset, percent\_min **=** 0, percent\_max **=** 100, is\_print **=** **True**):

columns\_with\_omissions **=** []

row\_count **=** dataset**.**shape[0]

**for** col **in** dataset**.**columns:

percent **=** round((dataset[col]**.**isnull()**.**sum() **/** row\_count) **\*** 100)

**if** is\_print:

print("\"{0}\" ({1}) пропущенно {2}% "**.**format(col, dataset[col]**.**dtype, percent))

**if** percent **>** percent\_min **and** percent **<=** percent\_max:

columns\_with\_omissions**.**append(col)

**return** columns\_with\_omissions

del\_cols\_names **=** get\_missing\_columns(dataset, percent\_min**=**30)

**for** col **in** del\_cols\_names:

dataset **=** dataset**.**drop(col, axis **=** 1)

print('Удалено {} признаков: '**.**format(len(del\_cols\_names)))

print(del\_cols\_names)

Функция для фильтрации признаков.

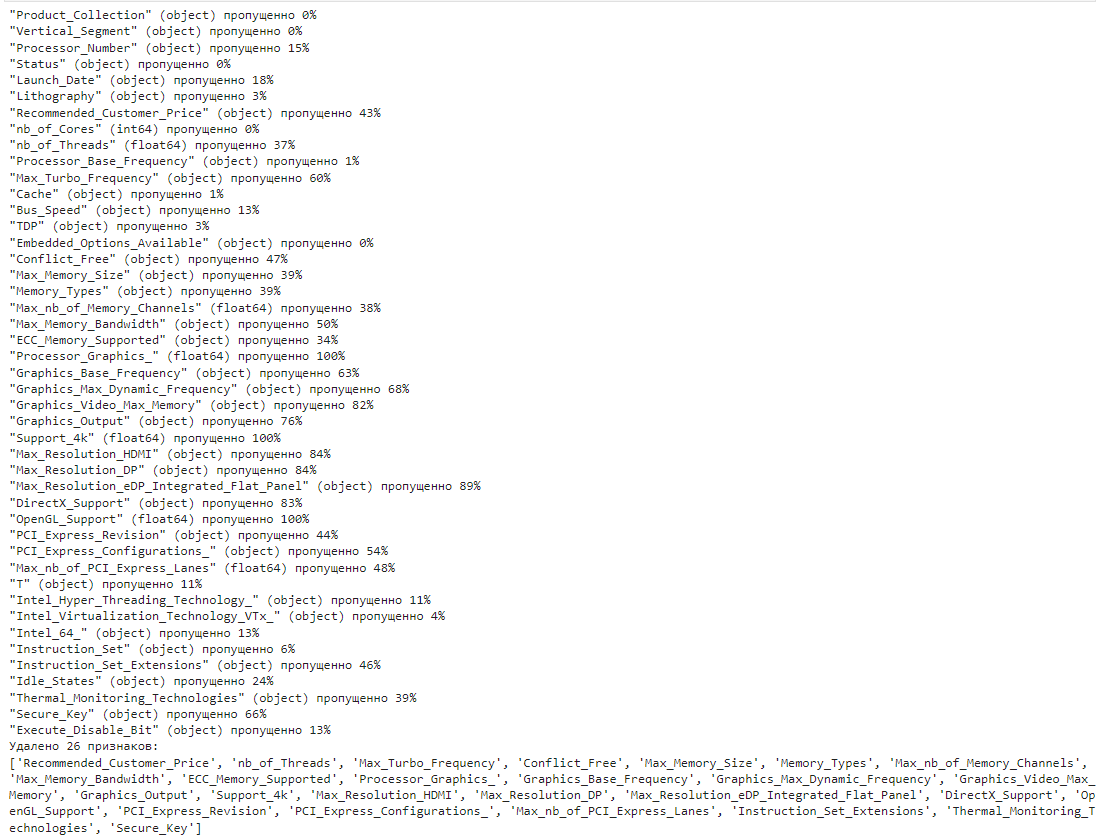


Рисунок 1. Предобработка датасета.

В ходе работы функции было удалено 26 признаков из 45 не целевых. В удаленных признаках отсутствовало более 30% значений.

*# Статистика числа пропусков для каждого оставшегося признака:*

get\_missing\_columns(dataset)

print("Форма датасета: {}"**.**format(str(dataset**.**shape)))

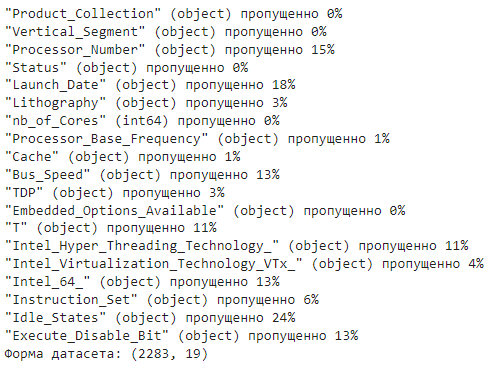


Рисунок 2. Число пропусков после обработки.

Следующая стратегия устанения пропусков – удаление строк. Ее следует применять, когда число пропусков не велико и удалее строк не приведет к существенному сокращению датасета. Удали строки где число пропущенных значений менее 7%.

row\_before\_dpop **=** dataset**.**shape[0]

cols\_with\_nulls\_rows **=** get\_missing\_columns(dataset, percent\_max **=** 7, is\_print **=** **False**)

print("В следующих колонках будут удалены строки: {}"**.**format(str(cols\_with\_nulls\_rows)))

dataset **=** dataset**.**dropna(axis **=** 0, subset **=** cols\_with\_nulls\_rows)

row\_after\_dpop **=** dataset**.**shape[0]

print("Число удаленных строк: {}"**.**format(row\_before\_dpop **-** row\_after\_dpop))

print("Оставшиеся признаки с пропусками: {}"**.**format(str(get\_missing\_columns(dataset, is\_print **=** **False**))))

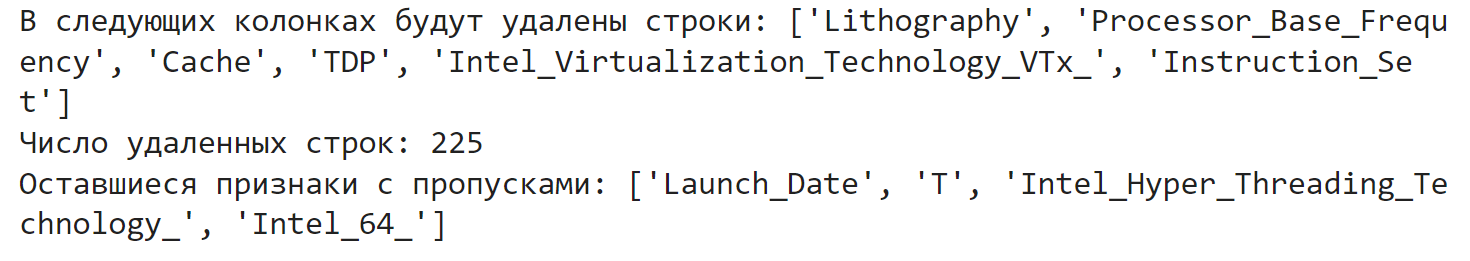


Рисунок 3. Число пропусков после удаления строк.

1.2 Обработка не стандартных признаков

Таким образом, остались лишь категориальные признаки с пропусками. Однако признак Т - температура, по существу, числовое значение. Необходимо преобразовать его к float типу. Часть значений в колонке Т сложно однозначно итерпретировать (например: "C1+D1=75°C; M0=72°C"), заменим их пустыми значениями:

regex\_is\_valid **=** r'^[0-9\.°\sCС]\*$'

not\_valid **=** []

**for** val **in** dataset['T']:

**if** **not** re**.**match(regex\_is\_valid, str(val)):

not\_valid**.**append(val)

dataset['T'] **=** dataset['T']**.**replace(list(set(not\_valid)), np**.**nan)

Далее извлечем строк float значения:

regex\_val **=** r'[0-9\.]{1,7}'

col\_name **=** "T"

**for** val **in** dataset[col\_name]:

**if** type(val) **is** **not** str:

**continue**

match **=** re**.**search(regex\_val, val)

**if** **not** match:

**raise** BaseException("Не удалось распарсить {}"**.**format(val))

**else**:

dataset[col\_name] **=** dataset[col\_name]**.**replace([val], match**.**group())

dataset[col\_name] **=** dataset[col\_name]**.**astype(float)

После чего обработаем признак TDP, также являющийся по существу числовым:

col\_name **=** "TDP"

**for** val **in** dataset[col\_name]:

**if** type(val) **is** **not** str:

**continue**

num **=** float(str(val)**.**split(" ")[0])

dataset[col\_name] **=** dataset[col\_name]**.**replace([val], num)

dataset[col\_name] **=** dataset[col\_name]**.**astype(float)

dataset[col\_name]**.**value\_counts()

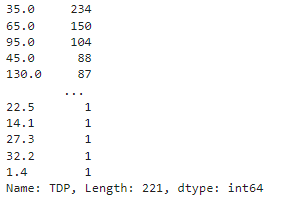


Рисунок 4. Извлеченные числовые значения TDP.

Также нестандартным является признак Processor\_Base\_Frequency. В датасете это не числовое значение (type object), но на деле представляет собой float значение. Причем частоты представлены в MHz и GHz, что также требует конвертации.

Попробуем извлечь числовые значения и привести их к одинаковой размерности - MHz.

dataset["Processor\_Base\_Frequency"]**.**value\_counts()

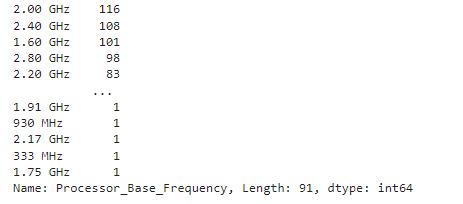


Рисунок 5. Значения признака Processor\_Base\_Frequency до обработки.

Код для обработки признака:

regex\_ghz **=** r"[0-9\.]\*(?= GHz)"

regex\_mhz **=** r"[0-9\.]\*(?= MHz)"

col\_name **=** "Processor\_Base\_Frequency"

**for** val **in** dataset[col\_name]:

**if** type(val) **is** **not** str:

**continue**

match **=** re**.**search(regex\_ghz, str(val))

**if** match:

dataset[col\_name] **=** dataset[col\_name]**.**replace([val], float(match**.**group()) **\*** 1000)

**else**:

match **=** re**.**search(regex\_mhz, str(val))

**if** **not** match:

**raise** BaseException("Не удалось распарсить {}"**.**format(val))

**else**:

dataset[col\_name] **=** dataset[col\_name]**.**replace([val], float(match**.**group()))

dataset[col\_name] **=** dataset[col\_name]**.**astype(int)

dataset["Processor\_Base\_Frequency"]**.**value\_counts()

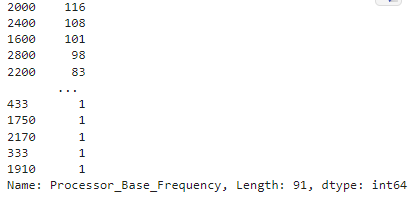


Рисунок 6. Значения признака Processor\_Base\_Frequency после обработки.

Оставшиеся пропуски в данных заполним наиболее частыми значениями соответствующих признаков для чего используем SimpleImputer со статегией most\_frequent:

before\_imputing **=** dataset**.**copy();

imputing\_cols **=** get\_missing\_columns(dataset, is\_print **=** **False**)

imputer **=** SimpleImputer(strategy **=** "most\_frequent")

**for** col **in** imputing\_cols:

dataset[col] **=** imputer**.**fit\_transform(dataset[[col]])

Исследуем разницу распределений значений до и после устранения пропусков:

**def** plot\_hist\_diff(old\_ds, new\_ds, cols):

"""

Разница между распределениями до и после устранения пропусков

"""

**for** c **in** cols:

fig **=** plt**.**figure()

ax **=** fig**.**add\_subplot(111)

ax**.**title**.**set\_text('Поле - ' **+** str(c))

old\_ds[c]**.**hist(bins**=**50, ax**=**ax, density**=True**, color**=**'green')

new\_ds[c]**.**hist(bins**=**50, ax**=**ax, density**=True**, color**=**'blue', alpha**=**0.5)

plt**.**show()

plot\_hist\_diff(before\_imputing, dataset, imputing\_cols);

get\_missing\_columns(dataset)

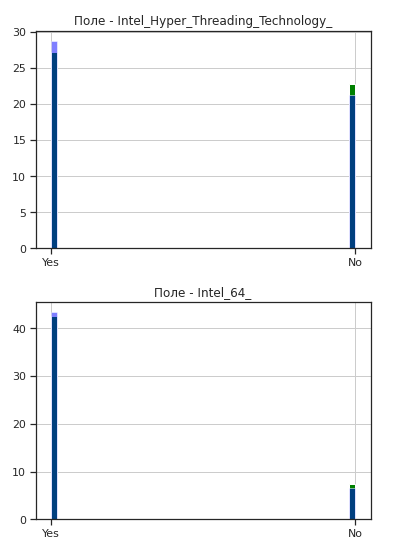
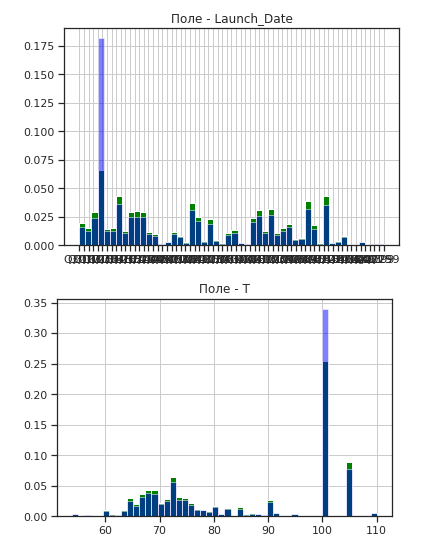


Рисунок 7. Разница между распределениями до и после устранения пропусков. Зеленый цвет – распределение до обработки. Светло синий – после обработки.

В оставшихся признаках датасета отсутствуют пропуски:

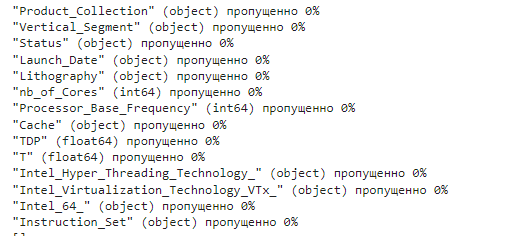


Рисунок 8. Анализ пропусков в признаках после их устранения.

1.3 Масштабирование признаков

Большинство алгоритмов машинного обучения пказывают более высукую точность на масштабированных признаках. Масштабированием называется общий процесс изменения диапазона признака. Это необходимый шаг, потому что признаки измеряются в разных единицах, а значит покрывают разные диапазоны. Это сильно искажает результаты таких алгоритмов, как метод опорных векторов и метод k-ближайших соседей, которые учитывают расстояния между измерениями.

Выделим признаки, которые необходимо масштабировать

dataset\_num **=** dataset[["nb\_of\_Cores", "Processor\_Base\_Frequency", "T", "T\_bc", "TDP"]]

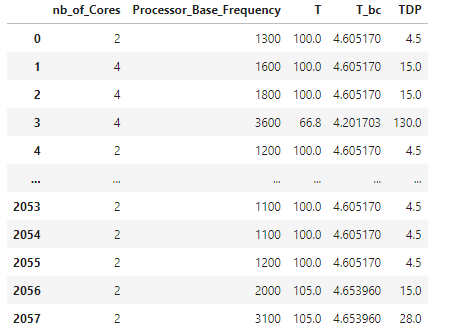


Рисунок 9. Состав признаков для масштабирования.

Выполним масштабирование указанных выше признаков на осноании z-оценки. Z оценка - это мера относительного разброса наблюдаемого или измеренного значения, которая показывает, сколько стандартных отклонений составляет его разброс относительного среднего значения. Это безразмерный статистический показатель, используемый для сравнения значений разной размерности или шкалой измерений.

standard\_scaler **=** StandardScaler()

dataset\_num\_ss **=** pd**.**DataFrame(standard\_scaler**.**fit\_transform(dataset\_num), columns**=**dataset\_num**.**columns)

dataset\_num\_ss**.**describe()

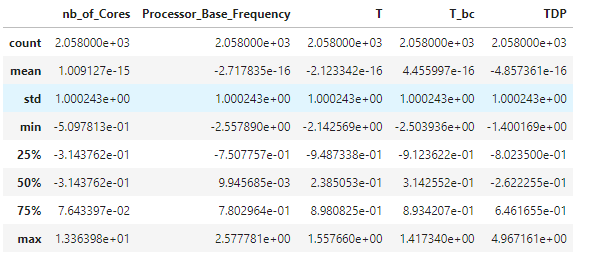


Рисунок 10. Статистические показатели для признаков, прошедших обработку StandardScaler.

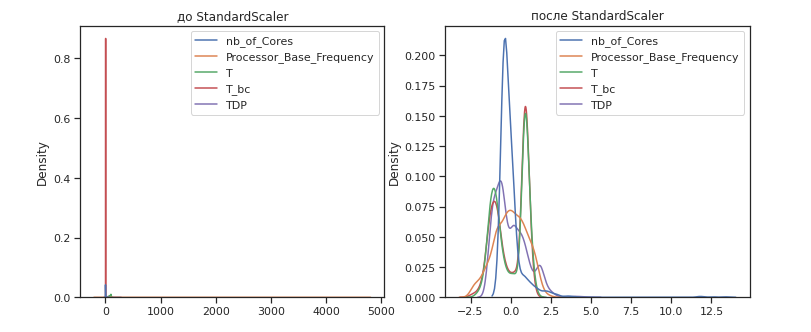
Исследуем рапределение значений после обработки:

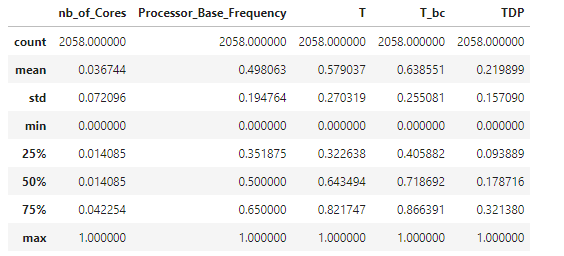
Рисунок 11. Рапределение значений до и после обработки.

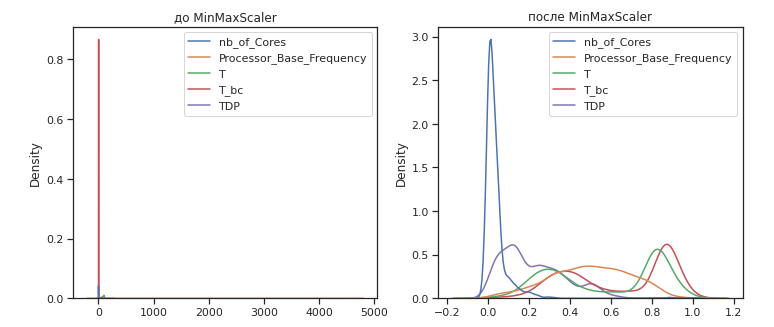
Изучим показатели распределения для иных алгоритмов масштабирования. Воспользуемся MinMaxScaler. Этот оценщик масштабирует и преобразует каждый признак индивидуально таким образом, чтобы он находилась в заданном диапазоне в обучающем наборе, например, между нулем и единицей.

min\_max\_scaler **=** MinMaxScaler()

dataset\_num\_mm **=** pd**.**DataFrame(min\_max\_scaler**.**fit\_transform(dataset\_num), columns**=**dataset\_num**.**columns)

dataset\_num\_mm**.**describe()



Рисунок 12. Статистические показатели для признаков, прошедших обработку MinMaxScaler.

Также выполним аналогичные операции для алгоритма масштабирования. Этот алгоритм удаляет медиану и масштабирует данные в соответствии с диапазоном квантилей (по умолчанию используется значение IQR: Межквартильный диапазон). IQR - это диапазон между 1-м квартилем (25-й квантиль) и 3-м квартилем (75-й квантиль).

robust\_scaler **=** RobustScaler()

dataset\_num\_r **=** pd**.**DataFrame(robust\_scaler**.**fit\_transform(dataset\_num), columns**=**dataset\_num**.**columns)

dataset\_num\_r**.**describe()

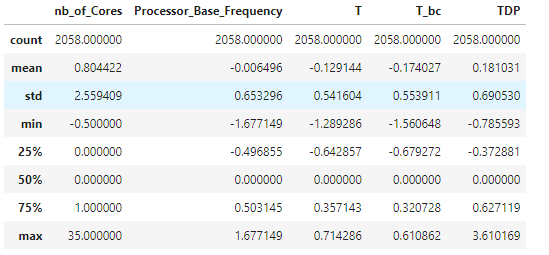


Рисунок 14. Статистические показатели для признаков, прошедших обработку RobustScaler.

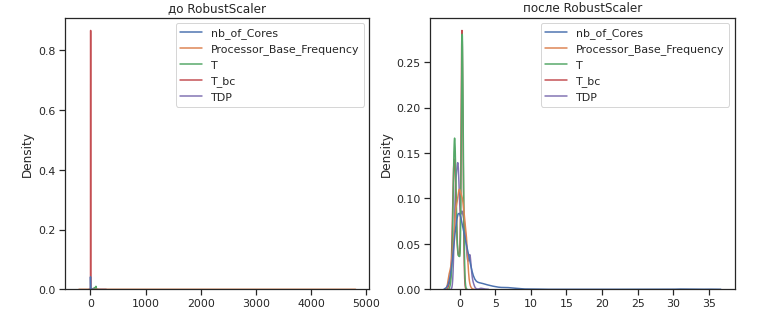


Рисунок 15. Рапределение значений до и после обработки

Как следует из графиков распределений, исходная форма распределения наиболее точно сохраняется при использовании MinMaxScaler. Далее при обучении модели воспользуемся результатом работы именно этого алгоритма.

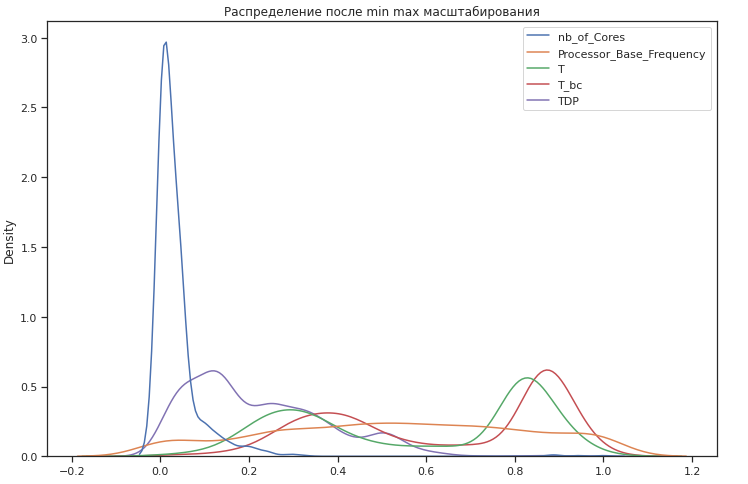


Рисунок 16. Рапределение значений после MinMax масштабирования.

1.4 Кодирование категориальных значений

Большинство алгоритмов машинного обучения не могут обрабатывать категориальные переменные, если они не преобразованы в числовые значения, а производительность многих алгоритмов зависит от того, как закодированы категориальные переменные.

Выполним кодирование части признаков с помощью LabelEncoder, а часть категориальных признаков, в которых мало уникальных категорий (например значения yes/no) или категории "не упорядоченные" с помощью OneHotEncoder, чтобы не увеличивать сильно признаковое пространство.

В случае LabelEncoder, каждой категории присваивается значение от 1 до N (здесь N - номер категории для признака). Одна из основных проблем этого подхода заключается в том, что между этими классами нет отношения или порядка, но алгоритм может рассматривать их как своего рода порядок или есть какая-то связь. Поэтому исползование OneHotEncoder также частичто позволит решить эту проблему оставаясь в компрмисе с размером признакового простанства.

В случае OneHotEncoder, сопоставялется каждая категория с вектором, который содержит 1 и 0, обозначая наличие или отсутствие признака. Количество векторов зависит от количества категорий для объекта.

dataset\_complex **=** dataset**.**copy()

categories\_le **=** []

categories\_oh **=** []

**for** col **in** categories\_all:

**if** (len(dataset[col]**.**value\_counts()) **<** 4):

categories\_oh**.**append(col)

**else**:

categories\_le**.**append(col)

**for** cat **in** categories\_le:

dataset\_complex[cat] **=** LabelEncoder()**.**fit\_transform(dataset\_complex[cat])

dataset\_complex[cat] **=** dataset\_complex[cat]**.**astype(int)

**for** cat **in** categories\_oh:

dataset\_complex **=** pd**.**concat([dataset\_complex, pd**.**get\_dummies(dataset\_complex[cat],

prefix **=** cat, drop\_first **=** **True**)],axis **=** 1)

dataset\_complex **=** dataset\_complex**.**drop(cat, axis **=** 1)

Таким образом, после обработки в датасете 16 признаков.

1.5 Устраниение выбросов в данных

В статистике выброс это экземпляр данных, который значительно отличается от других наблюдений. Отклонение может быть вызвано изменчивостью результатов измерений или может указывать на ошибку эксперимента. Выброс может вызвать серьезные проблемы при статистическом анализе и при построении модели машинного обучения.

Исследуем распределение значений признака Processor\_Base\_Frequency:

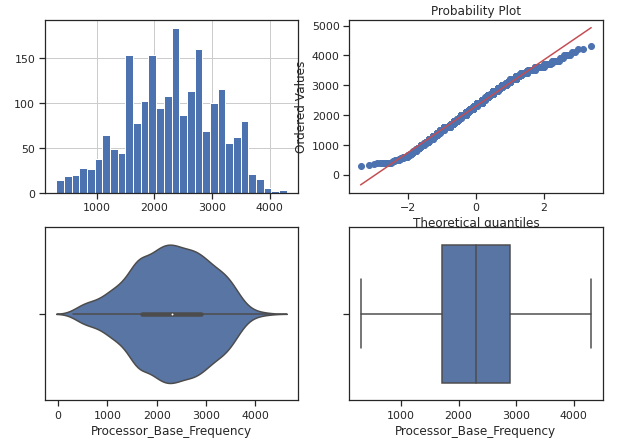


Рисунок 17. Показатели распределения для Processor\_Base\_Frequency.

Как видим распределение признака Processor\_Base\_Frequency нормальное, таким образом выборсы можно определять по правилу трех сигм или 5% и 95% квантилей. Воспользуемся последним правилом. Устраним выбросы с помощью удаления выбросов:

lower\_boundary **=** dataset[col\_with\_outlier]**.**quantile(0.05)

upper\_boundary **=** dataset[col\_with\_outlier]**.**quantile(0.95)

outliers\_temp **=** np**.**where(dataset[col\_with\_outlier] **>** upper\_boundary, **True**,

np**.**where(dataset[col\_with\_outlier] **<** lower\_boundary, **True**, **False**))

*# Удаление данных на основе флага*

dataset\_trimmed **=** dataset**.**loc[**~**(outliers\_temp), ]

title **=** "Processor\_Base\_Frequency после удаления выбросов, удалено строк: {}"**.**format(dataset**.**shape[0] **-** dataset\_trimmed**.**shape[0])

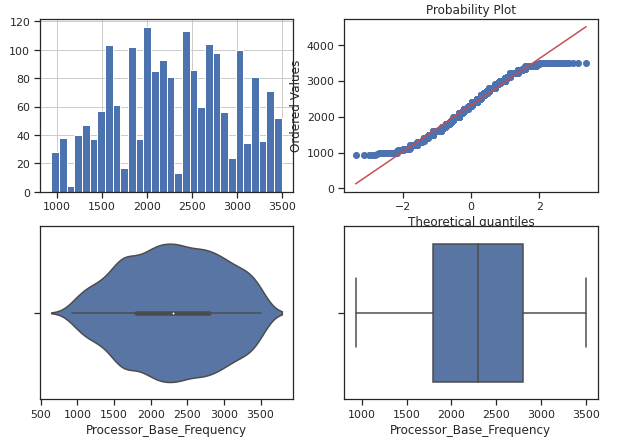
diagnostic\_plots(dataset\_trimmed, col\_with\_outlier, title)

Рисунок 18. Показатели распределения для Processor\_Base\_Frequency после удаления выбросов.

2. Построение модели машинного обучения.

2.1 Отбор признаков

Этап отбора признаков предполагает, что некоторе призенаки в датасетее либо излишни, либо незначимы, а потому могут быть удалены без существенной потери информации [1]. "Излишний" и "незначимый" являются двумя различными понятиями, поскольку один значимый признак может быть излишним при присутствии другого существенного признака, с которым он сильно коррелирует. Таким образом, задача отбора признаков - выбрать признаки, наиболее полезные для дальнейшего построения модели.

Воспользуемся методом отбора признаков основанном на анализе корреляции. Для чего необходимо построить матрицу корреляции и отобрать признаки, хорошо коррелирующие с целевым признаков и слабо коррелирующие друг с другом. Построим матрицу корреляции:

sns**.**heatmap(dataset[x\_names]**.**corr(), annot**=True**, fmt**=**'.3f')

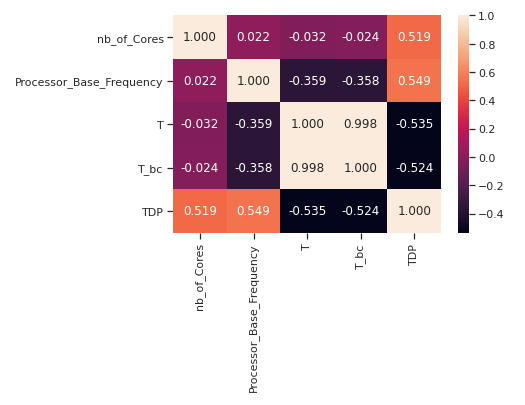


Рисунок 18. Матрица корреляций.

Видно что признаки T\_bc и T коррелируют друг сдругом и ни один из них не является целевым. Стоит исключить один из них.

2.2 Обучение и тестирование модели

Обучим модель RandomForestClassifier. Алгоритм «RandomForestClassifier» - это метаоценщик, он использует несколько decision tree моделей, обученных на разных выборках и использует усреднение их предсказаний для повышения точности прогнозирования и контроля над подгонкой:

**from** sklearn.ensemble **import** RandomForestClassifier

**from** sklearn.tree **import** DecisionTreeClassifier

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

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

**from** typing **import** Dict

**def** accuracy\_score\_for\_classes(

y\_true: np**.**ndarray,

y\_pred: np**.**ndarray) **->** Dict[int, float]:

"""

Вычисление метрики accuracy для каждого класса

y\_true - истинные значения классов

y\_pred - предсказанные значения классов

Возвращает словарь: ключ - метка класса,

значение - Accuracy для данного класса

"""

*# Для удобства фильтрации сформируем Pandas DataFrame*

d **=** {'t': y\_true, 'p': y\_pred}

df **=** pd**.**DataFrame(data**=**d)

*# Метки классов*

classes **=** np**.**unique(y\_true)

*# Результирующий словарь*

res **=** dict()

*# Перебор меток классов*

**for** c **in** classes:

*# отфильтруем данные, которые соответствуют*

*# текущей метке класса в истинных значениях*

temp\_data\_flt **=** df[df['t']**==**c]

*# расчет accuracy для заданной метки класса*

temp\_acc **=** round(accuracy\_score(temp\_data\_flt['t']**.**values,temp\_data\_flt['p']**.**values)**\***100,2)

*# сохранение результата в словарь*

res[c] **=** temp\_acc

**return** res

y\_name **=** "Vertical\_Segment"

x\_complex **=** list(dataset\_complex**.**columns)

x\_complex**.**remove(y\_name)

x\_simple **=** list(dataset\_simple**.**columns)

x\_simple**.**remove(y\_name)

x\_data **=** {"dataset\_complex" : dataset\_complex[x\_complex],

"dataset\_simple" : dataset\_simple[x\_simple]}

dataset\_complex[y\_name] **=** dataset\_complex[y\_name]**.**astype(int)

model **=** RandomForestClassifier(n\_estimators**=**50, random\_state**=**1)

results **=** {}

**for** data\_name, x\_data **in** x\_data**.**items():

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(x\_data, dataset\_complex[y\_name],

test\_size**=**0.3, random\_state**=**1)

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

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

scores **=** accuracy\_score\_for\_classes(y\_test, y\_pred)

results[data\_name] **=** scores

Отобразим диаграммы точности для кажого класса, для датасетов разной степени предобработки:

index **=** np**.**arange(len(dataset[y\_name]**.**value\_counts()))

bar\_width **=** 0.4

fig, ax **=** plt**.**subplots(figsize **=** (12,4))

ax**.**bar(index, results["dataset\_complex"]**.**values(), bar\_width, label**=**"dataset\_complex")

ax**.**bar(index **+** bar\_width, results["dataset\_simple"]**.**values(), bar\_width, label**=**"dataset\_simple")

ax**.**set\_ylabel('Точность')

ax**.**set\_xlabel('Классы')

ax**.**set\_title("Сравнение точностей моделей построенных на наборах данных с разной предобработкой")

ax**.**set\_xticks(index **+** bar\_width **/** 2)

ax**.**set\_xticklabels(index)

ax**.**legend(loc **=** "lower right")

plt**.**show()

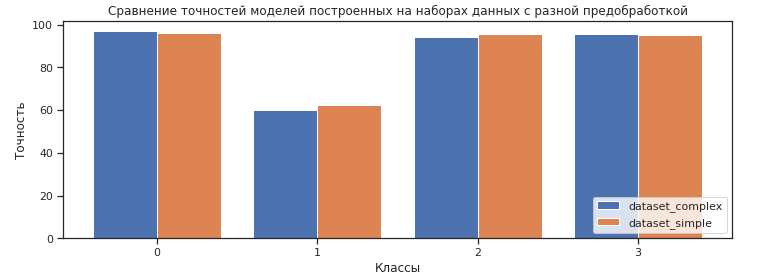


Рисунок 19. Результаты тестирования модели RandomForestClassifier .

Таким образом, мы получили приемлимую модель для определения класса рынка, к которому отностится исследуемый CPU.

3. Список литературы

1. К. Элбон. Машинное обучение с использованием Python. Сборник рецептов— Санкт-Петербург, Вильямс, 2019 – 200 с.

2. Ю.Е. Гапанюк. Обработка пропусков в данных, кодирование категориальных признаков, масштабирование данных. — [Электронный ресурс] — Режим доступа. — URL: https://nbviewer.org/github/ugapanyuk/ml\_course\_2020/blob/master/common/notebooks/missing/handling\_missing\_norm.ipynb (Дата обращения: 15.12.2021).

3. А. Мюллер, С. Гвидо. Введение в машинное обучение с помощью Python. Руководство для специалистов по работе с данными. — Санкт-Петербург, Вильямс, 2020 – 480 с.