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 КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

Построение модели машинного обучения для решения задачи классификации CPU.

Студент ____ИУ5-32м_____ _____ **Г.С.Седойкин**
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ **Ю.Е.Гапанюк**
(Подпись, дата) (И.О.Фамилия)

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

2021 г.

Введение

Задачей данной курсовой работы являетя построение классифицирующей модели машинного обучения. Модель должна решать задачу определения сегмента рынка CPU к которому относится исследуемый 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

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 так же 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.

```
"Product_Collection" (object) пропущенно 0%
"Vertical_Segment" (object) пропущенно 0%
"Processor_Number" (object) пропущенно 15%
"Status" (object) пропущенно 0%
"Launch_Date" (object) пропущенно 18%
"Lithography" (object) пропущенно 3%
"Recommended_Customer_Price" (object) пропущенно 43%
"nb_of_Cores" (int64) пропущенно 0%
"nb_of_Threads" (float64) пропущенно 37%
"Processor_Base_Frequency" (object) пропущенно 1%
"Max_Turbo_Frequency" (object) пропущенно 60%
"Cache" (object) пропущенно 1%
"Bus_Speed" (object) пропущенно 13%
"TDP" (object) пропущенно 3%
"Embedded_Options_Available" (object) пропущенно 0%
"Conflict_Free" (object) пропущенно 47%
"Max_Memory_Size" (object) пропущенно 39%
"Memory_Types" (object) пропущенно 39%
"Max_nb_of_Memory_Channels" (float64) пропущенно 38%
"Max_Memory_Bandwidth" (object) пропущенно 50%
"ECC_Memory_Supported" (object) пропущенно 34%
"Processor_Graphics_" (float64) пропущенно 100%
"Graphics_Base_Frequency" (object) пропущенно 63%
"Graphics_Max_Dynamic_Frequency" (object) пропущенно 68%
"Graphics_Video_Max_Memory" (object) пропущенно 82%
"Graphics_Output" (object) пропущенно 76%
"Support_4k" (float64) пропущенно 100%
"Max_Resolution_HDMI" (object) пропущенно 84%
"Max_Resolution_DP" (object) пропущенно 84%
"Max_Resolution_eDP_Integrated_Flat_Panel" (object) пропущенно 89%
"DirectX_Support" (object) пропущенно 83%
"OpenGL_Support" (float64) пропущенно 100%
"PCI_Express_Revision" (object) пропущенно 44%
"PCI_Express_Configurations_" (object) пропущенно 54%
"Max_nb_of_PCI_Express_Lanes" (float64) пропущенно 48%
"T" (object) пропущенно 11%
"Intel_Hyper_Threading_Technology_" (object) пропущенно 11%
"Intel_Virtualization_Technology_VTx_" (object) пропущенно 4%
"Intel_64_" (object) пропущенно 13%
"Instruction_Set" (object) пропущенно 6%
"Instruction_Set_Extensions" (object) пропущенно 46%
"Idle_States" (object) пропущенно 24%
"Thermal_Monitoring_Technologies" (object) пропущенно 39%
"Secure_Key" (object) пропущенно 66%
"Execute_Disable_Bit" (object) пропущенно 13%
Удалено 26 признаков:
['Recommended_Customer_Price', 'nb_of_Threads', 'Max_Turbo_Frequency', 'Conflict_Free', 'Max_Memory_Size', 'Memory_Types', 'Max_nb_of_Memory_Channels',
'Max_Memory_Bandwidth', 'ECC_Memory_Supported', 'Processor_Graphics_', 'Graphics_Base_Frequency', 'Graphics_Max_Dynamic_Frequency', 'Graphics_Video_Max_
Memory', 'Graphics_Output', 'Support_4k', 'Max_Resolution_HDMI', 'Max_Resolution_DP', 'Max_Resolution_eDP_Integrated_Flat_Panel', 'DirectX_Support', 'Op
enGL_Support', 'PCI_Express_Revision', 'PCI_Express_Configurations_', 'Max_nb_of_PCI_Express_Lanes', 'Instruction_Set_Extensions', 'Thermal_Monitoring_T
echnologies', 'Secure_Key']
```

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)))
```

```
"Product_Collection" (object) пропущенно 0%
"Vertical_Segment" (object) пропущенно 0%
"Processor_Number" (object) пропущенно 15%
"Status" (object) пропущенно 0%
"Launch_Date" (object) пропущенно 18%
"Lithography" (object) пропущенно 3%
"nb_of_Cores" (int64) пропущенно 0%
"Processor_Base_Frequency" (object) пропущенно 1%
"Cache" (object) пропущенно 1%
"Bus_Speed" (object) пропущенно 13%
"TDP" (object) пропущенно 3%
"Embedded_Options_Available" (object) пропущенно 0%
"T" (object) пропущенно 11%
"Intel_Hyper_Threading_Technology_" (object) пропущенно 11%
"Intel_Virtualization_Technology_VTx_" (object) пропущенно 4%
"Intel_64_" (object) пропущенно 13%
"Instruction_Set" (object) пропущенно 6%
"Idle_States" (object) пропущенно 24%
"Execute_Disable_Bit" (object) пропущенно 13%
Форма датасета: (2283, 19)
```

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 мнее 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))))
```

```
В следующих колонках будут удалены строки: ['Lithography', 'Processor_Base_Frequ
ency', 'Cache', 'TDP', 'Intel_Virtualization_Technology_VTx_', 'Instruction_Se
t']
Число удаленных строк: 225
Оставшиеся признаки с пропусками: ['Launch_Date', 'T', 'Intel_Hyper_Threading_Te
chnology_', 'Intel_64_']
```

Рисунок 3. Число пропусков после удаления строк.

1.2 Обработка не стандартных признаков

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

```
regex_is_valid = r'^[0-9\.°\sCC]*$'
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()
```

```
35.0     234
65.0     150
95.0     104
45.0      88
130.0     87
         ...
22.5       1
14.1       1
27.3       1
32.2       1
1.4        1
Name: TDP, Length: 221, dtype: int64
```

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()
        2.00 GHz    116
        2.40 GHz    108
        1.60 GHz    101
        2.80 GHz     98
        2.20 GHz     83
                    ...
        1.91 GHz      1
        930 MHz       1
        2.17 GHz      1
        333 MHz       1
        1.75 GHz      1
        Name: Processor_Base_Frequency, Length: 91, dtype: int64
```

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()
```

```
2000    116
2400    108
1600    101
2800     98
2200     83
        ...
433       1
1750      1
2170      1
333       1
1910      1
Name: Processor_Base_Frequency, Length: 91, dtype: int64
```

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)
```

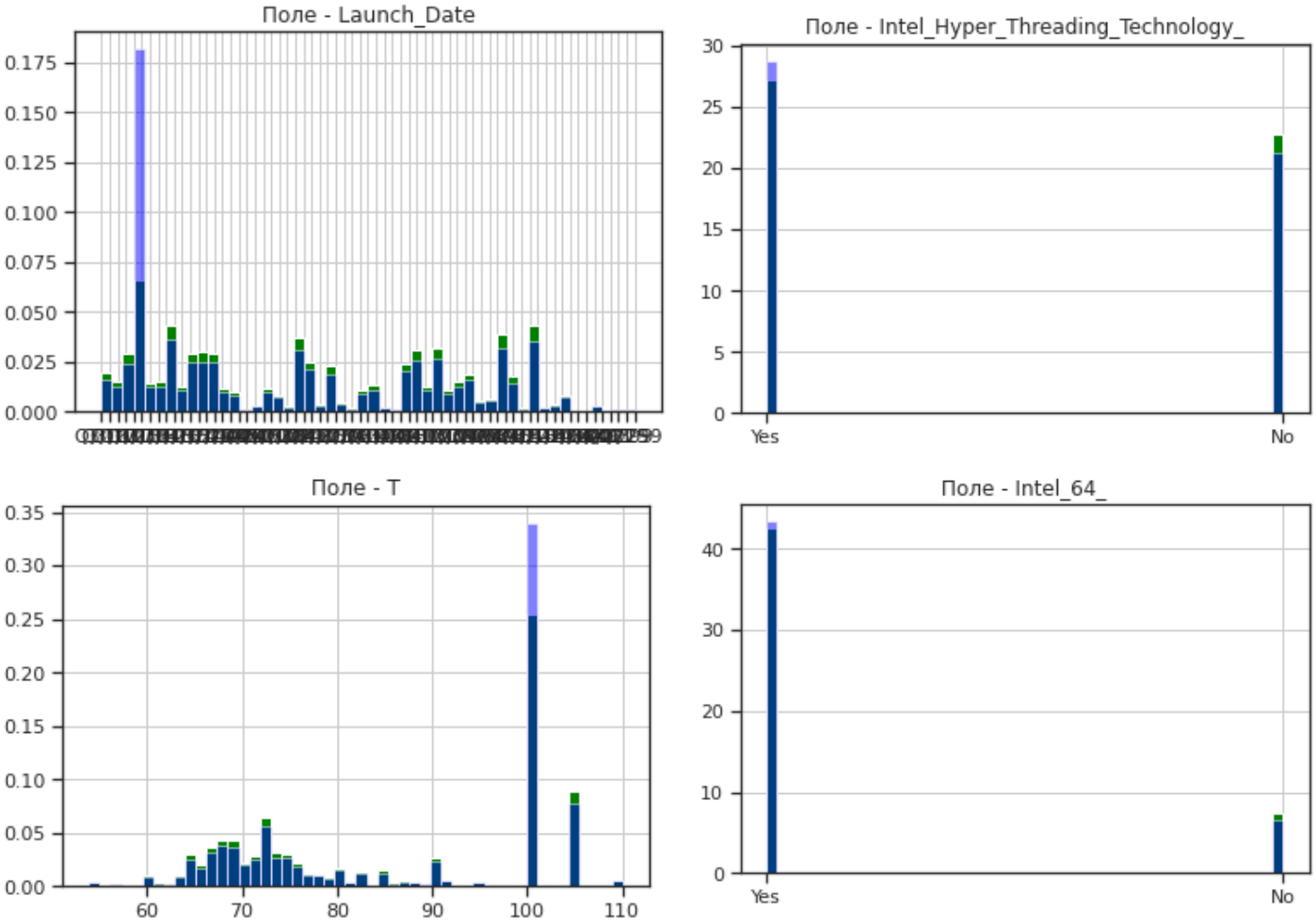


Рисунок 7. Разница между распределениями до и после устранения пропусков. Зеленый цвет – распределение до обработки. Светло синий – после обработки.

В оставшихся признаках датасета отсутствуют пропуски:

```
"Product_Collection" (object) пропущенно 0%
"Vertical_Segment" (object) пропущенно 0%
"Status" (object) пропущенно 0%
"Launch_Date" (object) пропущенно 0%
"Lithography" (object) пропущенно 0%
"nb_of_Cores" (int64) пропущенно 0%
"Processor_Base_Frequency" (int64) пропущенно 0%
"Cache" (object) пропущенно 0%
"TDP" (float64) пропущенно 0%
"T" (float64) пропущенно 0%
"Intel_Hyper_Threading_Technology_" (object) пропущенно 0%
"Intel_Virtualization_Technology_VTx_" (object) пропущенно 0%
"Intel_64_" (object) пропущенно 0%
"Instruction_Set" (object) пропущенно 0%
```

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"]]
```

	nb_of_Cores	Processor_Base_Frequency	T	T_bc	TDP
0	2	1300	100.0	4.605170	4.5
1	4	1600	100.0	4.605170	15.0
2	4	1800	100.0	4.605170	15.0
3	4	3600	66.8	4.201703	130.0
4	2	1200	100.0	4.605170	4.5
...	...	...	...	...	...
2053	2	1100	100.0	4.605170	4.5
2054	2	1100	100.0	4.605170	4.5
2055	2	1200	100.0	4.605170	4.5
2056	2	2000	105.0	4.653960	15.0
2057	2	3100	105.0	4.653960	28.0

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()
```

	nb_of_Cores	Processor_Base_Frequency	T	T_bc	TDP
count	2.058000e+03	2.058000e+03	2.058000e+03	2.058000e+03	2.058000e+03
mean	1.009127e-15	-2.717835e-16	-2.123342e-16	4.455997e-16	-4.857361e-16
std	1.000243e+00	1.000243e+00	1.000243e+00	1.000243e+00	1.000243e+00
min	-5.097813e-01	-2.557890e+00	-2.142569e+00	-2.503936e+00	-1.400169e+00
25%	-3.143762e-01	-7.507757e-01	-9.487338e-01	-9.123622e-01	-8.023500e-01
50%	-3.143762e-01	9.945685e-03	2.385053e-01	3.142552e-01	-2.622255e-01
75%	7.643397e-02	7.802964e-01	8.980825e-01	8.934207e-01	6.461655e-01
max	1.336398e+01	2.577781e+00	1.557660e+00	1.417340e+00	4.967161e+00

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

Исследуем рапределение значений после обработки:

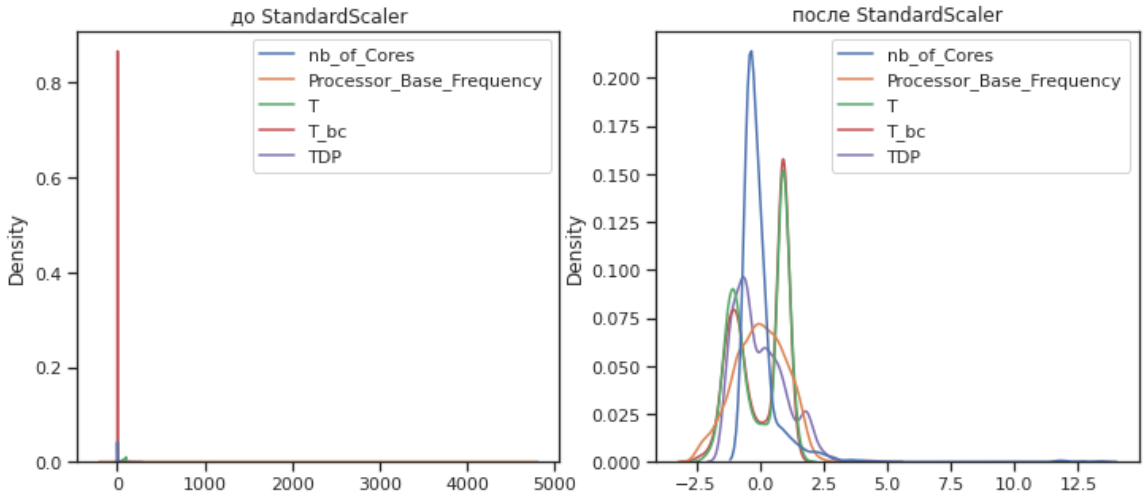


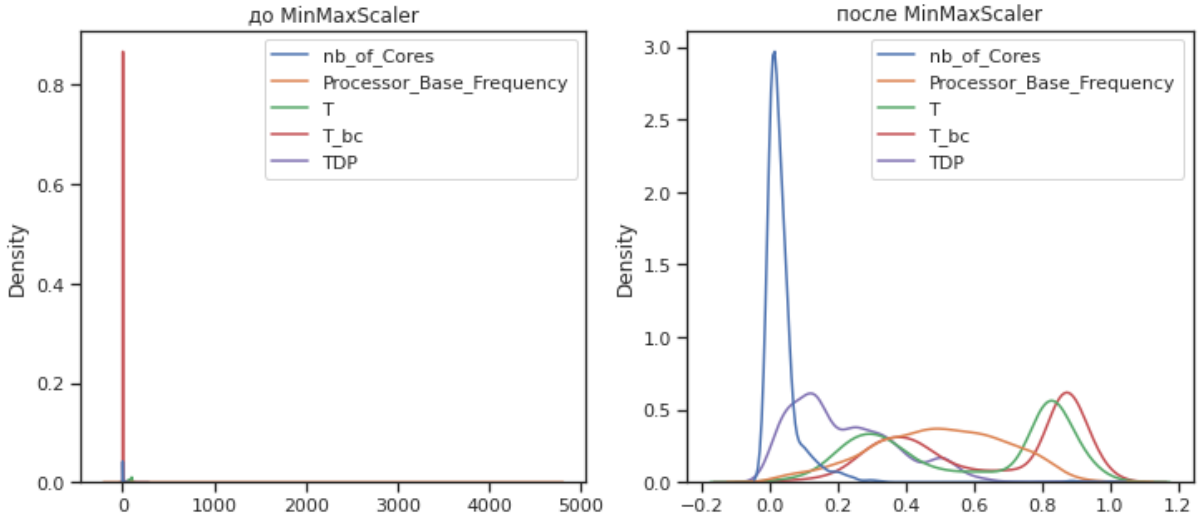
Рисунок 11. Рапределение значений до и после обработки.

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()
```

	nb_of_Cores	Processor_Base_Frequency	T	T_bc	TDP
count	2058.000000	2058.000000	2058.000000	2058.000000	2058.000000
mean	0.036744	0.498063	0.579037	0.638551	0.219899
std	0.072096	0.194764	0.270319	0.255081	0.157090
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.014085	0.351875	0.322638	0.405882	0.093889
50%	0.014085	0.500000	0.643494	0.718692	0.178716
75%	0.042254	0.650000	0.821747	0.866391	0.321380
max	1.000000	1.000000	1.000000	1.000000	1.000000

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()
```

	nb_of_Cores	Processor_Base_Frequency	T	T_bc	TDP
count	2058.000000	2058.000000	2058.000000	2058.000000	2058.000000
mean	0.804422	-0.006496	-0.129144	-0.174027	0.181031
std	2.559409	0.653296	0.541604	0.553911	0.690530
min	-0.500000	-1.677149	-1.289286	-1.560648	-0.785593
25%	0.000000	-0.496855	-0.642857	-0.679272	-0.372881
50%	0.000000	0.000000	0.000000	0.000000	0.000000
75%	1.000000	0.503145	0.357143	0.320728	0.627119
max	35.000000	1.677149	0.714286	0.610862	3.610169

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

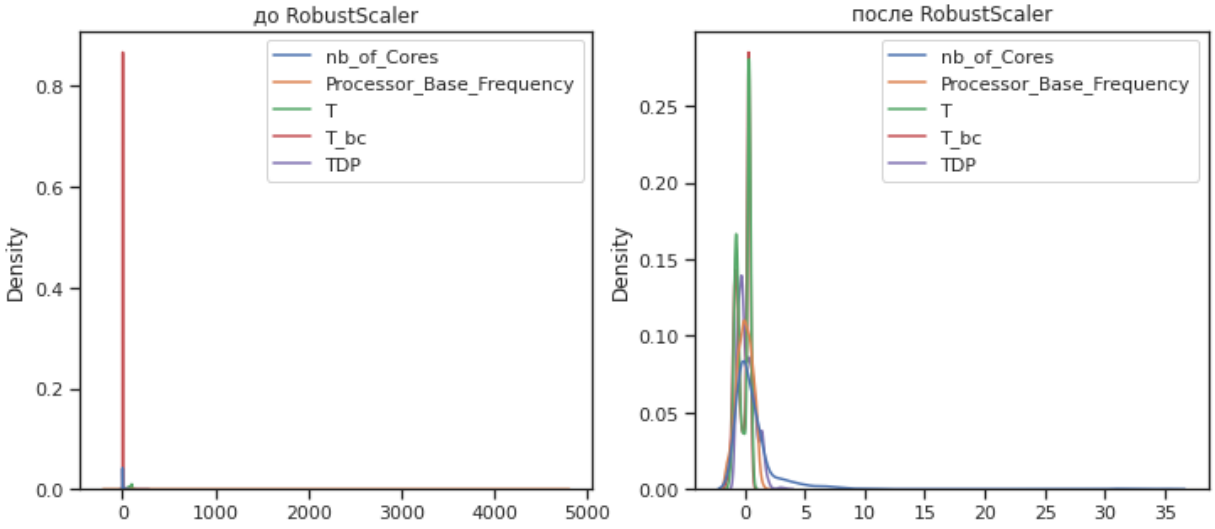


Рисунок 15. Рапределение значений до и после обработки

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

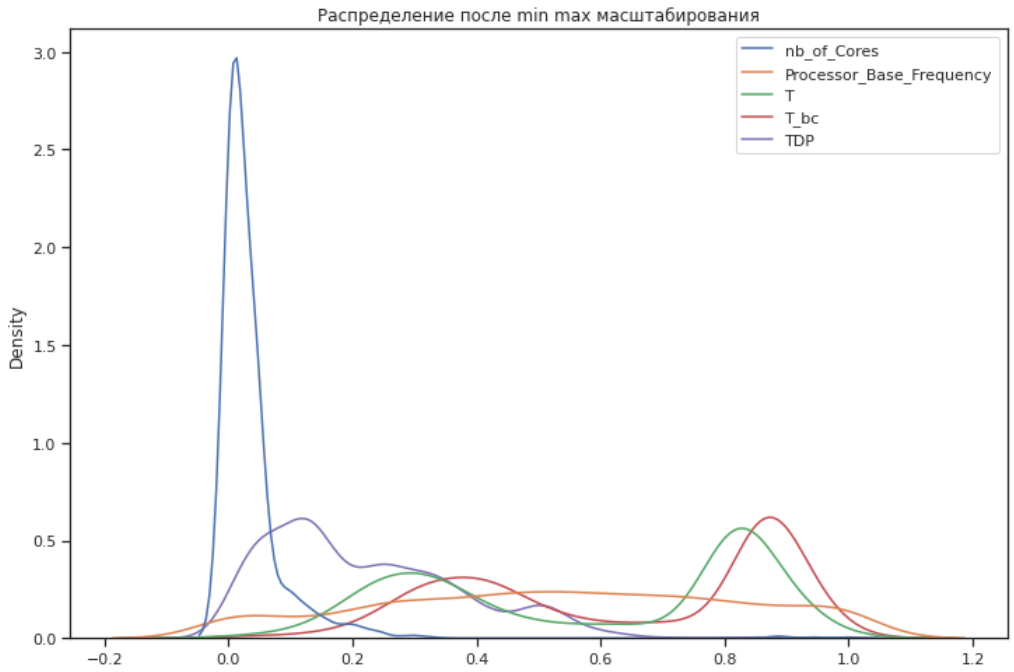


Рисунок 16. Рапределение значений после MinMax масштабирования.

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 Устранение 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:

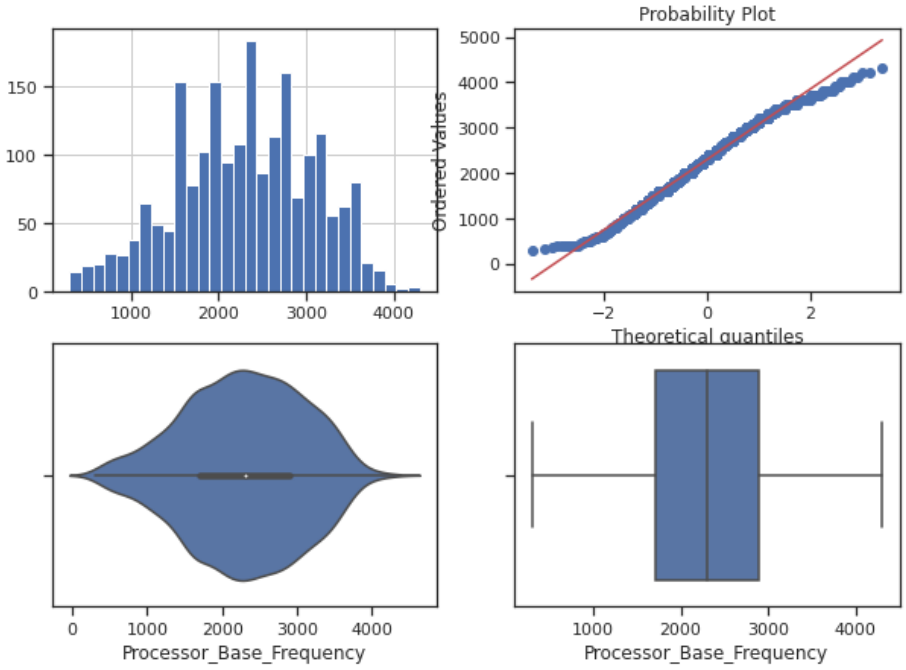


Рисунок 17. Показатели распределения для Processor_Base_Frequency.

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)
```

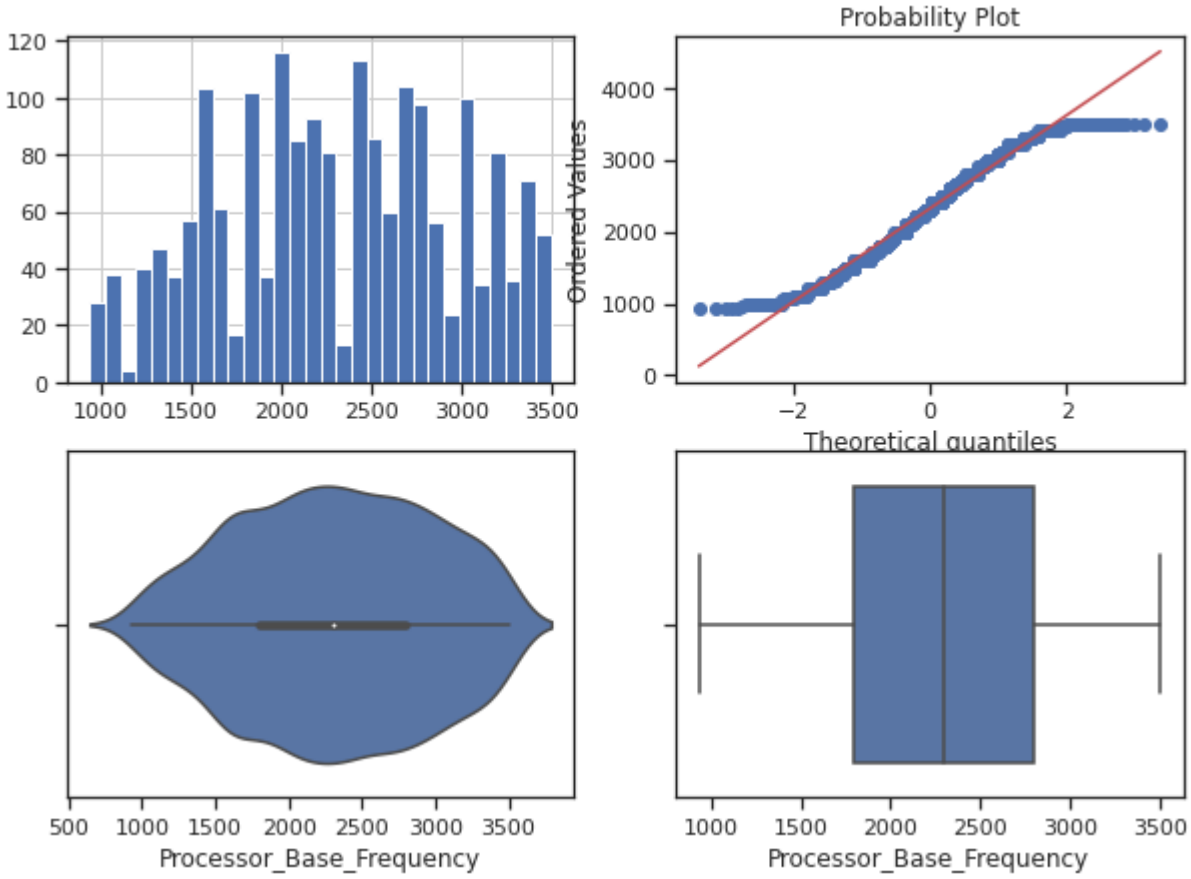


Рисунок 18. Показатели распределения для Processor_Base_Frequency после удаления выбросов.

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')
```

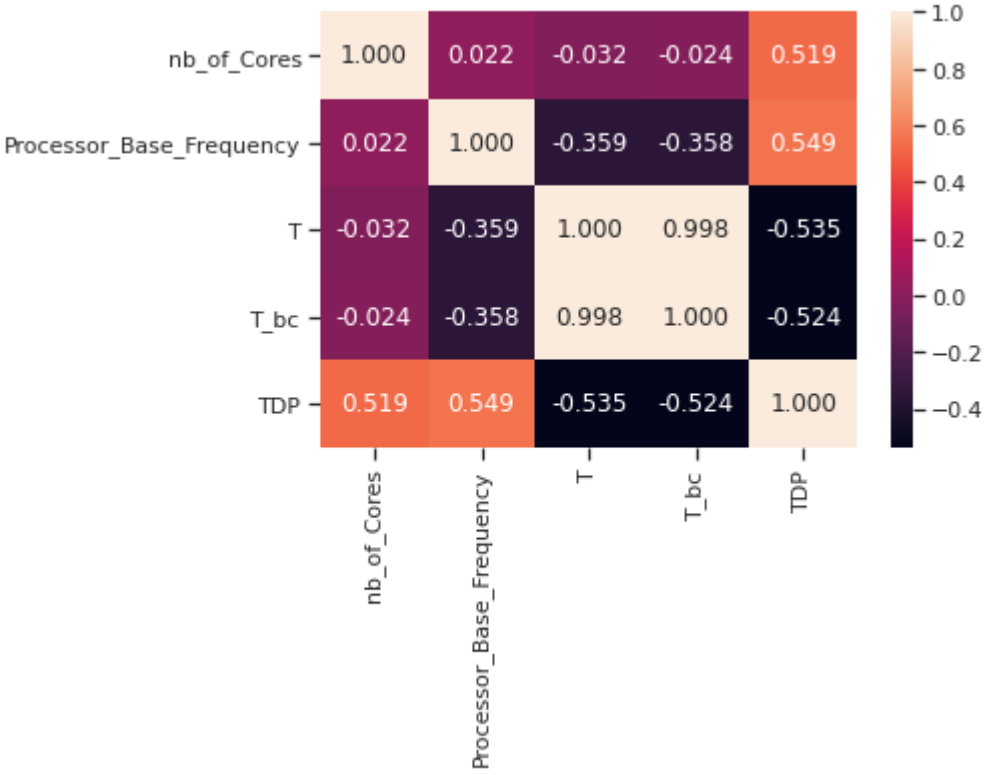


Рисунок 18. Матрица корреляций.

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()
```

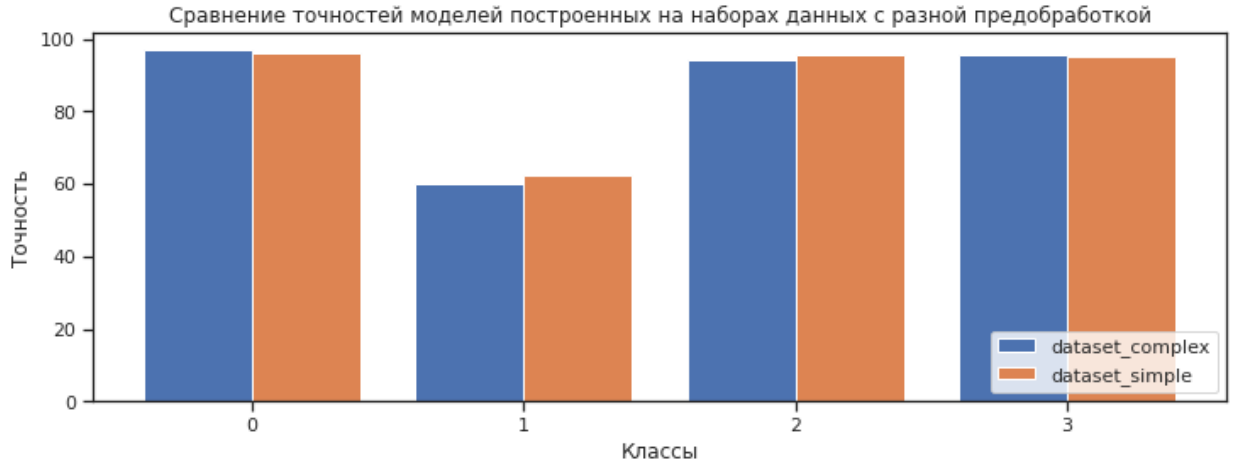


Рисунок 19. Результаты тестирования модели RandomForestClassifier .

Таким образом мы получили приемлимую модель для определения класса рынка к которому относится 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.