Министерство образования и науки 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

Классификация компьютерных процессоров

Студент ___ИУ5-33М___ **Шутов Максим Романович** _____
(Группа) (Подпись, дата) (Фамилия.И.О)

Руководитель **Юрий Евгеньевич Гапанюк**
(Подпись,дата) (фамилия.И.О)

г. Моква, 2021 г

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана» (МГТУ им. Н.Э.Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой ИУ-5

(Индекс)

В.М.Черненьки

(И.О.Фамилия) «

__ » ____ 2021г.

З А Д А Н И Е

на выполнение научно-исследовательской работы по теме«обработке и анализу данных»

Студент группы ИУ5-31М

Шутов Максим Романович

(фамилия, имя, отчество)

Направленость НИР (учебная, исследовательская, практическая производственная, и др.) исследовательская

Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения НИР: 25% к_нед, 50% к___нед, 75% к__нед, 100%к____нед.

Техническое задание: Классификация компьютерных процессоров

Оформление научно-исследовательской работы:

Расчетно-пояснительная запимка на _____листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плпкаты, слайды и.т.д.)

Дата выдачи « ___» _____2021г.

Руководитель НИР ____ **Юрий Евгеньевич Гапанюк** _____

(Подпись,дата) (*И.О.Фамилия)*

Студент Шутов Максим Романович

(Подпись,дата) (*И.О.Фамилия)*

Обучение с учителем.

Задание.

Задание по дисциплине направлено на решение комплексной задачи машинного обучения с учителем. Задание включает выполнение следующих шагов:

1. Поиск и выбор набора данных для построения модели машинного обучения. На основе выбранного набора данных строится модель машинного обучения для решения или задачи классификации, или задачи регрессии.

2. Для выбранного датасета (датасетов) на основе материалов лекций, в целях улучшения выборки, решить следующие задачи (если это необходимо в данном датасете):

    - устранение пропусков в данных;
    - кодирование категориальных признаков;
    - нормализацию числовых признаков;
    - масштабирование признаков;
    - обработку выбросов для числовых признаков;
    - обработку нестандартных признаков (которые не является числовым или категориальным);
    - отбор признаков, наиболее подходящих для построения модели;
    - устранение дисбаланса классов в случае решения задачи классификации на дисбалансированной выборке.
3. Обучить модель и оценить метрики качества для двух выборок : исходная выборка, которая содержит только минимальную предобработку данных, необходимую для построения модели (например, кодирование категориальных признаков). улучшенная выборка, полученная в результате полной предобработки данных в пункте 2.

    A. Построить модель с использованием произвольной библиотеки AutoML.
4. Сравнить метрики для трех полученных моделей.

Выполнение

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. Анализировать будем CPU файл. В ходе выполненния ДЗ будет решаться задача классификации.

In [1]:

```
import re
import numpy as np
import pandas as pd
pd.options.mode.chained_assignment = None
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
import scipy.stats as stats
%matplotlib inline
sns.set(style="ticks")


dataset = pd.read_csv('../test/Intel_CPUs.csv', sep=",")
dataset
```

Out[1]:

	Product_Collection	Vertical_Segment	Processor_Number	Status	Launch_Date	Lithograp
0	7th Generation Intel® Core™ i7 Processors	Mobile	i7-7Y75	Launched	Q3'16	14 r
1	8th Generation Intel® Core™ i5 Processors	Mobile	i5-8250U	Launched	Q3'17	14 r
2	8th Generation Intel® Core™ i7 Processors	Mobile	i7-8550U	Launched	Q3'17	14 r
3	Intel® Core™ X-series Processors	Desktop	i7-3820	End of Life	Q1'12	32 r
4	7th Generation Intel® Core™ i5 Processors	Mobile	i5-7Y57	Launched	Q1'17	14 r
...	...	...	...	...	...	
2278	6th Generation Intel® Core™ m Processors	Mobile	M5-6Y54	Launched	Q3'15	14 r
2279	6th Generation Intel® Core™ m Processors	Mobile	M5-6Y57	Launched	Q3'15	14 r
2280	6th Generation Intel® Core™ m Processors	Mobile	M7-6Y75	Launched	Q3'15	14 r
2281	5th Generation Intel® Core™ i7 Processors	Mobile	i7-5550U	Launched	Q1'15	14 r
2282	5th Generation Intel® Core™ i7 Processors	Mobile	i7-5557U	Launched	Q1'15	14 r

2283 rows × 45 columns

Устранение пропусков в данных:

Удалим признаки, число пропущенных значений в который больше 30%

In [2]:

```
def get_missing_columns(dataset, percent_min = 0, percent_max = 100, is_pri
    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]
        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)
```

```
"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', 'Con
flict_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_Suppor
t', 'OpenGL_Support', 'PCI_Express_Revision', 'PCI_Express_Configurations_
', 'Max_nb_of_PCI_Express_Lanes', 'Instruction_Set_Extensions', 'Thermal_Mo
```

In [3]:

```
# 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)
```

In [4]:

```
#Стоит исключить ряд признаков, которые не будут полезны при построении мод
dataset = dataset.drop(['Bus_Speed', 'Idle_States', 'Execute_Disable_Bit',
```

Если число пропущенных значений мало (менее 7%), то удалим строки

In [5]:

```
row_before_dpop = dataset.shape[0]
cols_with_nulls_rows = get_missing_columns(dataset, percent_max = 7, is_pri
print("В следующих колонках будут удалены строки: {}".format(str(cols_with_
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
```

```
В следующих колонках будут удалены строки: ['Lithography', 'Processor_Base_
Frequency', 'Cache', 'TDP', 'Intel_Virtualization_Technology_VTx_', 'Instru
ction_Set']
```

```
Число удаленных строк: 225
Оставшиеся признаки с пропусками: ['Launch_Date', 'T', 'Intel_Hyper_Threadi
```

Преобразование не стандартных признаков

Признак T - температура, вообще говоря исловое 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:

In [6]:

```
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)
```

In [7]:

```
#Теперь все красиво, можем начать преобразовывать в числовые данные:
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)
```

In [8]:

```
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()
```

Out[8]:

```
35.0     234
65.0     150
95.0     104
45.0      88
130.0     87
        ...
29.1       1
21.8       1
16.8       1
57.8       1
17.8       1
Name: TDP, Length: 221, dtype: int64
```

Признак Processor_Base_Frequency также требует предобработки. 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

распрарсить числовые значения и привести их к одинаковой размерности - MHz.

In [9]:
```
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.gr
    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(matc
dataset[col_name] = dataset[col_name].astype(float)
```

Заполним пропуски

Заполним пропуски в категориальных значениях с помощью SimpleImputer со стратегией most_frequent:

In [10]:
```
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.
        plt.show()

before_imputing = dataset.copy();
imputing_cols = get_missing_columns(dataset, is_print = False)

imputer = SimpleImputer(strategy = "most_frequent")

dataset.shape
for col in imputing_cols:
    dataset[col] = imputer.fit_transform(dataset[[col]])

plot_hist_diff(before_imputing, dataset, imputing_cols);
get_missing_columns(dataset)
```

Поле - Launch_Date
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000

Поле - T
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
60
70
80
90
100
110

Поле - Intel_Hyper_Threading_Technology_
30
25
20
15
10
5
0
Yes
No

```
"Product_Collection" (object) пропущенно 0%
"Vertical_Segment" (object) пропущенно 0%
```

Поиск дисбалансных признаков

Перед тем, как перейти к кодирования категориавльных признаков, изучи наличие дисбалансных классов

In [11]:

```
categories_all = ["Product_Collection", "Status", "Vertical_Segment", "Laur
                  "Instruction_Set", "Intel_Hyper_Threading_Technology_", "Inte
                  "Intel_64_"]


for cat in categories_all:
    classes = dataset[cat].value_counts()
    max_cls_count = classes[0]
    min_cls_count = classes[len(classes) - 1]
    print("{} imbalance ratio = {}".format(cat, round(max_cls_count / min_c
```

```
Product_Collection imbalance ratio = 187
Status imbalance ratio = 132
Vertical_Segment imbalance ratio = 4
Launch_Date imbalance ratio = 455
Lithography imbalance ratio = 27
Cache imbalance ratio = 208
Instruction_Set imbalance ratio = 56
Intel_Hyper_Threading_Technology_ imbalance ratio = 1
Intel_Virtualization_Technology_VTx_ imbalance ratio = 250
Intel_64_ imbalance ratio = 7
```

Кодирование категориальных значений

В случае минимальной переработки датасета выполним кодирование всех признаков с помощью LabelEncoder:

In [12]:

```
dataset_simple = dataset.copy()
for cat in categories_all:
    dataset_simple[cat] = LabelEncoder().fit_transform(dataset_simple[cat])
    dataset_simple[cat] = dataset_simple[cat].astype(int)
```

Для более глубокой предобработки 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.

In [13]:

```
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_co
                                                                 prefix = c
    dataset_complex = dataset_complex.drop(cat, axis = 1)
dataset_complex
```

Out[13]:

	Product_Collection	Vertical_Segment	Status	Launch_Date	Lithography	nb_of_Cores	Proce
0	13	2	3	49	1	2	
1	15	2	3	50	1	4	
2	16	2	3	50	1	4	
3	29	0	2	12	5	4	
4	12	2	3	17	1	2	
...	...	...	...	...	...	...	
2278	10	2	3	48	1	2	
2279	10	2	3	48	1	2	
2280	10	2	3	48	1	2	
2281	6	2	3	15	1	2	
2282	6	2	3	15	1	2	

2058 rows × 16 columns

Нормализация значений

Посмотрим как распределены числовые признаки, и нормализуем их с помощью преобразовани логарифмического преобразования.

In [14]:

```
def diagnostic_plots(df, variable):
    plt.figure(figsize=(15,6))
    # гистограмма
    plt.subplot(1, 2, 1)
    df[variable].hist(bins=30)
    ## Q-Q plot
    plt.subplot(1, 2, 2)
    stats.probplot(df[variable], dist="norm", plot=plt)
    plt.show()
num_cols = ["nb_of_Cores", "Processor_Base_Frequency", "T", "TDP"]
dataset_complex[num_cols].hist(figsize=(10, 10))
plt.show()
```

In [15]:

```
# Необходимо преобразовать данные к действительному типу
dataset_complex['TDP'] = dataset_complex['TDP'].astype('float')
dataset_complex['TDP'], param = stats.yeojohnson(dataset_complex['TDP'])
print('Оптимальное значение λ = {}'.format(param))
diagnostic_plots(dataset_complex, 'TDP')
```

```
Оптимальное значение λ = 0.37242146081505817
```

Обработка выбросов

Изучим числовые признаки на наличие выбросов

In [16]:

```
def diagnostic_plots(df, variable):
    fig, ax = plt.subplots(figsize=(10,7))
    # гистограмма
    plt.subplot(2, 2, 1)
    df[variable].hist(bins=30)
    ## Q-Q plot
    plt.subplot(2, 2, 2)
    stats.probplot(df[variable], dist="norm", plot=plt)
    # ящик с усами
    plt.subplot(2, 2, 3)
    sns.violinplot(x=df[variable])
    # ящик с усами
    plt.subplot(2, 2, 4)
    sns.boxplot(x=df[variable])
    fig.suptitle(variable)
    plt.show()

for col in num_cols:
    diagnostic_plots(dataset_complex, col)
```

nb_of_Cores

Processor_Base_Frequency

Как видим распределение признака nb_of_Cores ассиметричное, однако присутствуют выбросы. Выборсы можно определять по правилу трех сигм или 5% и 95% квантилей. Воспользуемся последним правилом. Устраним выбросы с помощью замены выбросов на найденные верхнюю и нижнюю границы:

In [17]:

```
col_with_outlier = "nb_of_Cores"
K2 = 1.5
IQR = dataset_complex[col].quantile(0.75) - dataset_complex[col_with_outlie
lower_boundary = dataset_complex[col].quantile(0.25) - (K2 * IQR)
upper_boundary = dataset_complex[col].quantile(0.75) + (K2 * IQR)
outliers_temp = np.where(dataset_complex[col_with_outlier] > upper_boundary
                         np.where(dataset_complex[col_with_outlier] < lower

dataset_complex[col_with_outlier] = np.where(dataset_complex[col_with_outli
                                     np.where(dataset_complex[col_with_outli
diagnostic_plots(dataset_complex, col_with_outlier)
```

Масштабирование

Т.к в качестве модели будет использвоаться RandomForestRegressor, выполнять масштабирование не обязательно

Отбор признаков

Воспользуемся heat map для обнаружения корреляций:

In [18]:

```
_, ax = plt.subplots(figsize = (12,12))
sns.heatmap(dataset_complex.corr(), annot=True, fmt='.1f', ax = ax)
plt.show()
```

Построение моделей

Классический способ:

In [19]:

```
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
        # 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_com
                                                        test_size=0.3, ranc
    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
```

In [20]:

```
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="datase
ax.bar(index + bar_width, results["dataset_simple"].values(), bar_width, la

ax.set_ylabel('Точность')
ax.set_xlabel('Классы')
ax.set_title("Сравнение точностей моделей построенных на наборах данных с р
ax.set_xticks(index + bar_width / 2)
ax.set_xticklabels(index)
ax.legend(loc = "lower right")

plt.show()
```

Auto ML:

In [21]:

```
from supervised.automl import AutoML

model = AutoML()
X_train, X_test, y_train, y_test = train_test_split(dataset[x_simple], data
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
results["auto_ml"] = accuracy_score_for_classes(y_test, y_pred)
```

```
AutoML directory: AutoML_4
The task is multiclass_classification with evaluation metric logloss
AutoML will use algorithms: ['Baseline', 'Linear', 'Decision Tree', 'Random
Forest', 'Xgboost', 'Neural Network']
AutoML will ensemble availabe models

Pass the following variable as a keyword arg: x. From version 0.12, the onl
y valid positional argument will be `data`, and passing other arguments wit
hout an explicit keyword will result in an error or misinterpretation.
AutoML steps: ['simple_algorithms', 'default_algorithms', 'ensemble']
* Step simple_algorithms will try to check up to 3 models
1_Baseline logloss 1.286907 trained in 0.29 seconds
2_DecisionTree logloss 0.716295 trained in 17.29 seconds
3_Linear logloss 0.615692 trained in 7.12 seconds
* Step default_algorithms will try to check up to 3 models

ntree_limit is deprecated, use `iteration_range` or model slicing instead.
ntree_limit is deprecated, use `iteration_range` or model slicing instead.
4_Default_Xgboost logloss 0.222682 trained in 10.69 seconds
5_Default_NeuralNetwork logloss 0.469341 trained in 1.14 seconds
```

```
6_Default_RandomForest logloss 0.488964 trained in 7.63 seconds
* Step ensemble will try to check up to 1 model

An input array is constant; the correlation coefficient is not defined.
AutoML fit time: 52.61 seconds
AutoML best model: 4_Default_Xgboost

ntree_limit is deprecated, use `iteration_range` or model slicing instead.
```

In [22]:

```
index = np.arange(len(dataset[y_name].value_counts()))
bar_width = 0.3
fig, ax = plt.subplots(figsize = (10, 5))
ax.bar(index, results["dataset_complex"].values(), bar_width, label="datase
ax.bar(index + bar_width, results["dataset_simple"].values(), bar_width, la
ax.bar(index + 2 * bar_width, results["auto_ml"].values(), bar_width, label

ax.set_ylabel('Точность')
ax.set_xlabel('Классы')
ax.set_title("Сравнение точностей моделей")
ax.set_xticks(index + bar_width / 2)
ax.set_xticklabels(results["auto_ml"].keys())
ax.legend(loc = "lower right")

plt.show()
```

In [2]:

```
jupyter nbconvert --to pdf Shutov_NIR_OAD.ipynb
```

```
  File "C:\Users\NoNam\AppData\Local\Temp/ipykernel_16584/2383002372.py", l
ine 1
    jupyter nbconvert --to pdf Shutov_NIR_OAD.ipynb
            ^
SyntaxError: invalid syntax
```

In [2]:

```
conda install nbconvert
```

```
Collecting package metadata (current_repodata.json): ...working... done
Note: you may need to restart the kernel to use updated packages.

Solving environment: ...working... done

# All requested packages already installed.
```

In [3]:

```
conda install nbconvert==6.0.0
```

```
Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... failed with initial frozen solve. Retryi
ng with flexible solve.
Collecting package metadata (repodata.json): ...working... done
Solving environment: ...working... failed with initial frozen solve. Retryi
ng with flexible solve.

Note: you may need to restart the kernel to use updated packages.
PackagesNotFoundError: The following packages are not available from curren
t channels:

  - nbconvert==6.0.0

Current channels:

  - https://repo.anaconda.com/pkgs/main/win-64
  - https://repo.anaconda.com/pkgs/main/noarch
  - https://repo.anaconda.com/pkgs/r/win-64
  - https://repo.anaconda.com/pkgs/r/noarch
  - https://repo.anaconda.com/pkgs/msys2/win-64
  - https://repo.anaconda.com/pkgs/msys2/noarch

To search for alternate channels that may provide the conda package you're
looking for, navigate to

    https://anaconda.org

and use the search bar at the top of the page.
```

In [ ]: