# ПРИЛОЖЕНИЕ В

Исходный текст программного модуля

Листов 21

# Step 1: Import necessary libraries

import pandas as pd

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

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

from sklearn.linear\_model import LogisticRegression

from sklearn.preprocessing import LabelEncoder

from sklearn.pipeline import Pipeline

from sklearn.metrics import classification\_report

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

import matplotlib.cm as cm

from pathlib import Path

from tqdm.notebook import tqdm

def plot(df, level, val):

m = {

"Group": "Subclass",

"Subclass": "Class",

"Class": "Section",

}

if level not in m:

distr = df[level].value\_counts()

print(distr)

distr.sort\_index().plot(kind='bar')

plt.xlabel('Год')

plt.ylabel('Число патентов')

plt.title(f'Распределение патентов по {level}')

plt.show()

return

distr = df[df[m[level]] == val][level].value\_counts()

print(distr)

distr.sort\_index().plot(kind='bar')

plt.xlabel('Год')

plt.ylabel('Число патентов')

plt.title(f'Распределение патентов {m[level]} {val} по {level}')

plt.show()

def get\_data(year: int) -> pd.DataFrame:

return pd.read\_csv(f"years/{year}.csv")

def filter(df: pd.DataFrame, level: str, vals: list[str]) -> pd.DataFrame:

return df[df[level].isin(vals)]

def limiter(df: pd.DataFrame, level: str, percent: float) -> pd.DataFrame:

return train\_test\_split(df, train\_size=percent, stratify=df[level], shuffle=True)[0]

def balance(abc: pd.DataFrame, level: str, sublevel: str, value: str, lower\_bound: int, upper\_bound: int) -> pd.DataFrame:

df = abc.copy()

distr = df[df[level] == value][sublevel].value\_counts()

df = df[~df[sublevel].isin(distr[distr < lower\_bound].index)]

higher\_groups = distr[distr > upper\_bound].index

for group in higher\_groups:

excess\_records = distr[group] - int(upper\_bound)

drop\_indices = df[df[sublevel] == group].sample(n=excess\_records, random\_state=42).index

df = df.drop(drop\_indices)

return df

df = p2000.copy()

sublevel = "Section"

distr = df[sublevel].value\_counts()

df = df[~df[sublevel].isin(distr[distr < 1000].index)]

higher\_groups = distr[distr > 6000].index

for group in higher\_groups:

excess\_records = distr[group] - int(6000)

drop\_indices = df[df[sublevel] == group].sample(n=excess\_records, random\_state=42).index

df = df.drop(drop\_indices)

p2000 = df.copy()

# Clean data

patents['IPC'] = patents['IPC'].str.replace(':', '/')

# Drop rows with no IPC

patents = patents.drop(patents[patents["IPC"].isna()].index)

# Add a column for the section

patents['Section'] = patents['IPC'].str[0]

# Drop the section I - unknown

patents = patents.drop(patents[patents['Section'] == 'I'].index)

# Add a column for the class

patents['Class'] = patents['IPC'].str[:3]

# Add a column for the subclass

patents['Subclass'] = patents['IPC'].str[:4]

# Add column for groups

patents["Group"] = patents['Subclass'] + "-" + patents['IPC'].str.split("/").str[0].str.split("-").str[1].fillna('0').str.zfill(3)

import pickle

with open("p2000\_x\_train\_v.pkl", "wb") as f:

pickle.dump(p2000\_x\_train\_v, f)

with open("p2000\_x\_test\_v.pkl", "wb") as f:

pickle.dump(p2000\_x\_test\_v, f)

from sklearn.manifold import TSNE

from sklearn.decomposition import PCA

from umap import UMAP

pca = PCA(n\_components=2, random\_state=412)

tsne = TSNE(n\_components=2, random\_state=42)

umap = UMAP(n\_components=2, n\_neighbors=15, min\_dist=0.1, metric='cosine')

# plt\_result\_train = pca.fit\_transform(pipeline.named\_steps['tfidf'].transform(train\_df['Text']).toarray())

# plt\_result\_test = pca.transform(pipeline.named\_steps['tfidf'].transform(test\_df['Text']).toarray())

# plt\_result\_train = tsne.fit\_transform(pipeline.named\_steps['tfidf'].transform(train\_df['Text']).toarray())

# plt\_result\_test = tsne.fit\_transform(pipeline.named\_steps['tfidf'].transform(test\_df['Text']).toarray())

plt\_result\_train = umap.fit\_transform(p2000\_x\_train\_v)

plt\_result\_test = umap.transform(p2000\_x\_test\_v)

disp\_limit = 100000

subset\_train\_indices = []

subset\_test\_indices = []

for class\_label in p2000\_y\_train.unique():

class\_indices\_train = np.where(p2000\_y\_train == class\_label)[0][:disp\_limit]

class\_indices\_test = np.where(p2000\_y\_test == class\_label)[0][:disp\_limit]

subset\_train\_indices.extend(class\_indices\_train)

subset\_test\_indices.extend(class\_indices\_test)

label\_encoder = LabelEncoder()

train\_labels\_encoded = label\_encoder.fit\_transform(p2000\_y\_train)

test\_labels\_encoded = label\_encoder.transform(p2000\_y\_test)

for class\_label in p2000\_y\_train.unique():

class\_indices\_train = np.where(p2000\_y\_train == class\_label)[0][:disp\_limit]

subset\_train\_indices.extend(class\_indices\_train)

class\_indices\_test = np.where(p2000\_y\_test == class\_label)[0][:disp\_limit]

subset\_test\_indices.extend(class\_indices\_test)

classes = p2000\_y\_train.unique()

colors = cm.rainbow(np.linspace(0, 1, len(classes)))

labels = label\_encoder.inverse\_transform(train\_labels\_encoded[subset\_train\_indices])

plt.figure(figsize=(8, 6))

for label, color in zip(classes, colors):

indices = np.where(labels == label)

plt.scatter(plt\_result\_train[subset\_train\_indices, 0][indices], plt\_result\_train[subset\_train\_indices, 1][indices], c=color, label=label, marker='o', alpha=0.1, linewidths=0, s=10)

plt.title('2000 год')

plt.legend()

plt.show()

labels = label\_encoder.inverse\_transform(test\_labels\_encoded[subset\_test\_indices])

plt.figure(figsize=(8, 6))

for label, color in zip(classes, colors):

indices = np.where(labels == label)

plt.scatter(plt\_result\_test[subset\_test\_indices, 0][indices], plt\_result\_test[subset\_test\_indices, 1][indices], c=color, label=label, marker='o', alpha=0.3, linewidths=0, s=10)

plt.title('2010 год')

plt.legend()

plt.show()

# Take only G06 class

g06 = df[df['Class'] == 'G06']

# Drop columns that are not needed (keep only Text and Subclass (it is the label))

g06 = g06[['Text', 'Subclass']]

# Display the number of patents per subclass

g06["Subclass"].value\_counts()

labels = label\_encoder.inverse\_transform(train\_labels\_encoded[subset\_train\_indices])

plt.figure(figsize=(12, 8))

for label, color in zip(classes, colors):

indices = np.where(labels == label)

plt.scatter(plt\_result\_train[subset\_train\_indices, 0][indices], plt\_result\_train[subset\_train\_indices, 1][indices], c=color, label=label)

plt.title('2000 год')

plt.legend()

# Extract class names and corresponding metrics

classes = list(cr.keys())[:-3] # Exclude 'accuracy', 'macro avg', 'weighted avg'

precision = [cr[class\_]['precision'] for class\_ in classes]

recall = [cr[class\_]['recall'] for class\_ in classes]

f1\_score = [cr[class\_]['f1-score'] for class\_ in classes]

# Plotting the bar charts

plt.figure(figsize=(10, 6))

# Precision

plt.bar(classes, precision, color=sns.color\_palette("viridis", len(classes)))

plt.title('Precision for Each Class')

plt.xlabel('Classes')

plt.ylabel('Precision')

plt.xticks(rotation=45, ha='right')

plt.show()

# Recall

plt.figure(figsize=(10, 6))

plt.bar(classes, recall, color=sns.color\_palette("viridis", len(classes)))

plt.title('Recall for Each Class')

plt.xlabel('Classes')

plt.ylabel('Recall')

plt.xticks(rotation=45, ha='right')

plt.show()

# F1-Score

plt.figure(figsize=(10, 6))

plt.bar(classes, f1\_score, color=sns.color\_palette("viridis", len(classes)))

plt.title('F1-Score for Each Class')

plt.xlabel('Classes')

plt.ylabel('F1-Score')

plt.xticks(rotation=45, ha='right')

plt.show()

# Accuracy Pie Chart

accuracy = cr['accuracy'] \* 100

labels = ['Accuracy', 'Incorrect']

sizes = [accuracy, 100 - accuracy]

colors = ['lightcoral', 'lightskyblue']

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

plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', startangle=140)

plt.title('Accuracy Pie Chart')

plt.show()

# Creating a DataFrame

df = pd.DataFrame.from\_dict(differences, orient='index', columns=['Precision'])

# Setting a more scientific style with Seaborn

sns.set(style="whitegrid")

plt.figure(figsize=(10, 6))

# Plotting the bar chart

sns.barplot(x=df.index, y='Precision', data=df, palette='viridis')

# Adding labels and title

plt.xlabel('Подклассы')

plt.ylabel('Точность')

plt.title('Относительные результаты классификации')

# Displaying the plot

plt.show()

stop\_words = set(stopwords.words('english')).union(['eg', 'etc', 'ie'])

lemmatizer = WordNetLemmatizer()

def preprocess(documents):

preprocessed\_docs = []

# Для каждлго документа в наборе

for doc in documents:

# Удаление знаков препинания

doc = re.sub(r'[^\w\s]', '', doc)

# Токенизация документа

tokens = word\_tokenize(doc.lower())

# Удаление стоп-слов

tokens = [token for token in tokens if token not in stop\_words]

# Лемматизация токенов (приведение к начальной форме)

lemmatized\_tokens = [lemmatizer.lemmatize(token) for token in tokens]

# Добавление предобработанных токенов в список

preprocessed\_docs.append(lemmatized\_tokens)

return preprocessed\_docs

def extract\_keywords\_tfidf(documents, n=10):

# Предобработка текстовых данных

preprocessed\_docs = preprocess(documents)

# Создание словаря

dictionary = corpora.Dictionary(preprocessed\_docs)

# Создание корпуса как список мешков слов

corpus = [dictionary.doc2bow(doc) for doc in preprocessed\_docs]

# Обучение модели TF-IDF

tfidf = TfidfModel(corpus)

tfidf\_scores = {}

# Для каждого документа в корпусе

for i, doc in enumerate(corpus):

doc\_tfidf = tfidf[doc]

# Для каждого термина в документе

for term\_id, score in doc\_tfidf:

# Получение термина по его идентификатору

term = dictionary.get(term\_id)

# Если термин еще не встречался, то добавить его в словарь

if term not in tfidf\_scores:

tfidf\_scores[term] = score

# Иначе добавить к существующему значению

else:

tfidf\_scores[term] += score

# Сортировка терминов по убыванию веса

sorted\_terms = sorted(tfidf\_scores.items(), key=lambda x: x[1], reverse=True)

return [term for term, score in sorted\_terms[:n]]

def extract\_keywords\_lda(documents, n=10):

# Предобработка текстовых данных

preprocessed\_docs = preprocess(documents)

# Создание словаря

dictionary = corpora.Dictionary(preprocessed\_docs)

# Создание корпуса как список мешков слов

corpus = [dictionary.doc2bow(doc) for doc in preprocessed\_docs]

# Обучение модели LDA

lda = LdaModel(corpus, num\_topics=1, id2word=dictionary)

# Извлечение ключевых слов

topic\_terms = lda.print\_topics()[0][1]

# Разбиение на список терминов и их весов

topic\_terms = [term.split('\*')[1].strip().replace('"', '') for term in topic\_terms.split('+')]

return topic\_terms[:n]

pipeline = Pipeline([

('tfidf', TfidfVectorizer(

ngram\_range=(1, 1),

lowercase=True

)),

# ('pca', PCA(n\_components=0.95)),

('clf', MultinomialNB())

])

patents = pd.read\_csv('patents.csv')

classes = patents['IPC'].unique()

colors = ["red", "blue", "green", "yellow", "orange", "purple", "pink", "brown", "black", "gray"]

for class\_ in classes:

print(class\_, patents[patents['IPC'] == class\_].shape[0])

# Fill NaN in 'Novelty' column with empty string

patents["Novelty"] = patents["Novelty"].fillna("")

patents['Text'] = patents["Title"] + patents["Novelty"]

patents[patents.isnull().any(axis=1)]

# Logistic Regression Classifier

from sklearn.linear\_model import LogisticRegression

clf = LogisticRegression(

multi\_class="multinomial",

solver="lbfgs",

random\_state=42,

max\_iter=1000,

)

print("Training the classifier...")

clf.fit(X\_train\_vectors, y\_train)

print("Evaluating...")

y\_pred\_lr = clf.predict(X\_test\_vectors)

print("Done!")

# plot 2d data using PCA

pca = PCA(n\_components=2, random\_state=42)

X\_pca = pca.fit\_transform(X\_test\_vectors.toarray())

plt.figure(figsize=(12, 8))

for label, color in zip(classes, colors):

plt.scatter(X\_pca[y\_test == label, 0], X\_pca[y\_test == label, 1], label=f"{label} real", alpha=0.5, s=10, c=color)

plt.scatter(X\_pca[y\_pred\_lr == label, 0], X\_pca[y\_pred\_lr == label, 1], label=f"{label} predicted", alpha=1, s=1, c=color)

plt.legend()

plt.show()

# Clustering with K-Means

from sklearn.cluster import KMeans

kmeans = KMeans(n\_clusters=4, random\_state=42, n\_init=10)

print("Training the classifier...")

kmeans.fit(X\_train\_vectors)

print("Evaluating...")

y\_pred\_km = kmeans.predict(X\_test\_vectors)

print("Done!")

# Сопоставление кластеров с классами

cluster\_labels = {}

for cluster in np.unique(y\_pred\_km):

counts = y\_test[y\_pred\_km == cluster].value\_counts()

class\_probas = counts / counts.sum()

print(f"Cluster {cluster} probabilities: {[f'{class\_}: {proba:.2f}' for class\_, proba in zip(class\_probas.index, class\_probas.values)]}")

cluster\_labels[cluster] = class\_probas.index[0]

y\_pred\_km = np.array([cluster\_labels[cluster] for cluster in y\_pred\_km])

# plot 2d data using PCA

pca = PCA(n\_components=2, random\_state=42)

X\_pca = pca.fit\_transform(X\_test\_vectors.toarray())

plt.figure(figsize=(12, 8))

for label, color in zip(classes, colors):

plt.scatter(X\_pca[y\_test == label, 0], X\_pca[y\_test == label, 1], c=color, label=f"Class {label}", alpha=0.5, s=10)

plt.scatter(X\_pca[y\_pred\_km == label, 0], X\_pca[y\_pred\_km == label, 1], c=color, label=f"Cluster {label}", alpha=1, s=1)

plt.legend()

plt.show()

sections = {

'A': 'Human Necessities',

'B': 'Performing Operations; Transporting',

'C': 'Chemistry; Metallurgy',

'D': 'Textiles; Paper',

'E': 'Fixed Constructions',

'F': 'Mechanical Engineering; Lighting; Heating; Weapons; Blasting',

'G': 'Physics',

'H': 'Electricity'

}

classes\_A = {

'A01': 'Agriculture; Forestry; Animal Husbandry; Hunting; Trapping; Fishing',

'A21': 'Baking; Equipment for Making or Processing Doughs; Doughs for Baking',

'A22': 'Butchering; Meat Treatment; Processing Poultry or Fish',

'A23': 'Foods or Foodstuffs; Their Treatment, Not Covered by Other Classes',

'A24': 'Tobacco; Cigars; Cigarettes; Smokers Requisites',

'A41': 'Wearing Apparel',

'A42': 'Headwear',

'A43': 'Footwear',

'A44': 'Haberdashery; Jewellery',

'A45': 'Hand or Travelling Articles',

'A46': 'Brushware',

'A47': 'Furniture; Domestic Articles or Appliances; Coffee Mills; Spice Mills; Suction Cleaners in General',

'A61': 'Medical or Veterinary Science; Hygiene',

'A62': 'Life-saving; Fire-fighting',

'A63': 'Sports; Games; Amusements',

'A99': 'Subject Matter not Provided for in Other Groups of this Subclass'

}

subclasses\_A01 = {

'A01B': 'Soil Working in Agriculture or Forestry; Parts, Details, or Accessories of Agricultural Machines or Implements, in General',

'A01C': 'Planting; Sowing; Fertilising',

'A01D': 'Harvesting; Mowing',

'A01F': 'Processing of Harvested Produce; Hay or Straw Presses; Devices for Storing Agricultural or Horticultural Produce',

'A01G': 'Horticulture; Cultivation of Vegetables, Flowers, Rice, Fruit, Vineyards, Hops or Seaweed; Forestry; Watering',

'A01H': 'New Plants or Processes for Obtaining Them; Plant Reproduction by Tissue Culture Techniques',

'A01J': 'Manufacture of Dairy Products',

'A01K': 'Animal Husbandry; Care of Birds, Fishes, Insects; Fishing; Rearing or Breeding Animals, Not Otherwise Provided for; New Breeds of Animals',

'A01L': 'Veterinary Preventive or Therapeutic Agents',

'A01M': 'Catching, Trapping or Scaring of Animals; Apparatus for the Destruction of Noxious Animals or Noxious Plants',

'A01N': 'Preservation of Bodies of Humans or Animals or Plants or Parts Thereof; Biocides, e.g. as Disinfectants, as Pesticides or as Herbicides; Pest Repellants or Attractants; Plant Growth Regulators',

'A01P': 'Biocides, e.g. as Disinfectants, as Pesticides or as Herbicides; Pest Repellants or Attractants; Plant Growth Regulators'

}

subclasses\_A21 = {

'A21B': "Bakers' Ovens; Machines or Equipment for Baking",

'A21C': 'Machines or Equipment for Making or Processing Doughs; Handling Baked Articles Made from Dough',

'A21D': 'Treatment, e.g. Preservation, of Flour or Dough for Baking, e.g. by Addition of Materials; Baking; Bakery Products; Preservation Thereof'

}

subclasses\_A22 = {

'A22B': 'Slaughtering',

'A22C': 'Processing Meat, Poultry, or Fish'

}

classes\_B = {

'B01': 'Physical or Chemical Processes or Apparatus in General',

'B02': 'Crushing, Pulverising, or Disintegrating; Preparatory Treatment of Grain for Milling',

'B03': 'Separation of Solid Materials Using Liquids or Using Pneumatic Tables or Jigs; Magnetic or Electrostatic Separation of Solid Materials from Solid Materials or Fluids; Separation by High-voltage Electric Fields',

'B04': 'Centrifugal Apparatus or Machines for Carrying-out Physical or Chemical Processes',

'B05': 'Spraying or Atomising in General; Applying Liquids or Other Fluids to Surfaces, in General',

'B06': 'Generating or Transmitting Mechanical Vibrations in General',

'B07': 'Separating Solids from Solids; Sorting',

'B08': 'Cleaning',

'B09': 'Disposal of Solid Waste; Reclamation of Contaminated Soil',

'B21': 'Mechanical Metal-working Without Essentially Removing Material; Punching Metal',

'B22': 'Casting; Powder Metallurgy',

'B23': 'Machine Tools; Metal-working Not Otherwise Provided for',

'B24': 'Grinding; Polishing',

'B25': 'Hand Tools; Portable Power-driven Tools; Handles for Hand Implements; Workshop Equipment; Manipulators',

'B26': 'Hand Cutting Tools; Cutting; Severing',

'B27': 'Working or Preserving Wood or Similar Material; Nails, Screws, or Fastening Devices; Moulds or Cores for Making Moulds',

'B28': 'Working Cement, Clay, or Stone',

'B29': 'Working of Plastics; Working of Substances in a Plastic State in General',

'B30': 'Presses',

'B31': 'Making Articles of Paper, Cardboard, or Material Worked in a Manner Analogous to Paper; Working Paper, Cardboard, or Material Worked in a Manner Analogous to Paper',

'B32': 'Layered Products',

'B33': 'Additive Manufacturing Technology',

'B41': 'Printing; Lining Machines; Typewriters; Stamps',

'B42': 'Bookbinding; Albums; Files; Special Printing Matter',

'B43': 'Writing or Drawing Implements; Bureau Accessories',

'B44': 'Decorative Arts',

'B60': 'Vehicles in General',

'B61': 'Railways',

'B62': 'Land Vehicles for Travelling Otherwise Than on Rails',

'B63': 'Ships or Other Waterborne Vessels; Related Equipment',

'B64': 'Aircraft; Aviation; Cosmonautics',

'B65': 'Conveying; Packing; Storing; Handling Thin or Filamentary Material',

'B66': 'Hoisting; Lifting; Hauling',

'B67': 'Opening or Closing Bottles, Jars or Similar Containers; Liquid Handling',

'B68': 'Saddlery; Harness',

'B81': 'Micro-structural Technology',

'B82': 'Nanotechnology',

'B99': 'Subject Matter not Provided for in Other Groups of this Subclass'

}

classes\_C = {

'C01': 'Inorganic Chemistry',

'C02': 'Treatment of Water, Waste Water, Sewage, or Sludge',

'C03': 'Glass; Mineral or Slag Wool',

'C04': 'Cements; Concrete; Artificial Stone; Ceramics; Refractories',

'C05': 'Fertilisers; Manufacture Thereof',

'C06': 'Explosives; Matches',

'C07': 'Organic Chemistry',

'C08': 'Organic Macromolecular Compounds; Their Preparation or Chemical Working-up; Compositions Based Thereon',

'C09': 'Dyes; Paints; Polishes; Natural Resins; Adhesives; Compositions Not Otherwise Provided for; Applications of Materials Not Otherwise Provided for',

'C10': 'Petroleum, Gas or Coke Industries; Technical Gases Containing Carbon Monoxide; Fuels; Lubricants; Peat',

'C11': 'Animal or Vegetable Oils, Fats, Fatty Substances or Waxes; Fatty Acids Therefrom; Detergents; Candles',

'C12': 'Biochemistry; Beer; Spirits; Wine; Vinegar; Microbiology; Enzymology; Mutation or Genetic Engineering',

'C13': 'Sugar Industry',

'C14': 'Skins; Hides; Pelts; Leather',

'C21': 'Metallurgy of Iron',

'C22': 'Metallurgy; Ferrous or Non-ferrous Alloys; Treatment of Alloys or Non-ferrous Metals',

'C23': 'Coating Metallic Material; Coating Material with Metallic Material; Chemical Surface Treatment; Diffusion Treatment of Metallic Material; Coating by Vacuum Evaporation, by Sputtering, by Ion Implantation or by Chemical Vapour Deposition, in General; Inhibiting Corrosion of Metallic Material or Incrustation in General',

'C25': 'Electrolytic or Electrophoretic Processes; Apparatus Therefor',

'C30': 'Crystal Growth',

'C40': 'Combination Technology',

'C99': 'Subject Matter not Provided for in Other Groups of this Section'

}

classes\_D = {

'D01': 'Natural or Man-made Threads or Fibres; Spinning',

'D02': 'Yarns; Mechanical Finishing of Yarns or Ropes; Warping or Beaming',

'D03': 'Weaving',

'D04': 'Braiding; Lace-making; Knitting; Trimmings; Non-woven Fabrics',

'D05': 'Sewing; Embroidering; Tufting',

'D06': 'Treatments of Textiles or the like; Laundering; Flexible Materials not Otherwise Provided for',

'D07': 'Ropes; Cables Other than Electric',

'D21': 'Paper-making; Production of Cellulose',

'D99': 'Subject Matter not Provided for in Other Groups of this Section'

}

classes\_E = {

'E01': 'Construction of Roads, Railways, or Bridges',

'E02': 'Hydraulic Engineering; Foundations; Soil-shifting',

'E03': 'Water Supply; Sewerage',

'E04': 'Building',

'E05': 'Locks; Keys; Window or Door Fittings; Safes',

'E06': 'Doors, Windows, Shutters, or Roller Blinds, in General; Ladders',

'E21': 'Earth or Rock Drilling; Mining',

'E99': 'Subject Matter not Provided for in Other Groups of this Section'

}

classes\_F = {

'F01': 'Machines or Engines in General; Engine Plants in General; Steam Engines',

'F02': 'Combustion Engines; Hot-gas or Combustion-product Engine Plants',

'F03': 'Machines or Engines for Liquids; Wind, Spring, or Weight Motors; Producing Mechanical Power or a Reactive Propulsive Thrust, not Otherwise Provided for',

'F04': 'Positive-displacement Machines for Liquids; Pumps for Liquids or Elastic Fluids',

'F15': 'Fluid-pressure Actuators; Hydraulics or Pneumatics in General',

'F16': 'Engineering Elements or Units; General Measures for Producing and Maintaining Effective Functioning of Machines or Installations; Thermal Insulation in General',

'F17': 'Storing or Distributing Gases or Liquids',

'F21': 'Lighting',

'F22': 'Steam Generation',

'F23': 'Combustion Apparatus; Combustion Processes',

'F24': 'Heating; Ranges; Ventilating',

'F25': 'Refrigeration or Cooling; Combined Heating and Refrigeration Systems; Heat Pump Systems; Manufacture or Storage of Ice; Liquefaction or Solidification of Gases',

'F26': 'Drying',

'F27': 'Furnaces; Kilns; Ovens; Retorts',

'F28': 'Heat Exchange in General',

'F41': 'Weapons',

'F42': 'Ammunition; Blasting',

'F99': 'Subject Matter not Provided for in Other Groups of this Section'

}

classes\_G = {

'G01': 'Measuring; Testing',

'G02': 'Optics',

'G03': 'Photography; Cinematography; Analogous Techniques Using Waves Other than Optical Waves; Electrography; Holography',

'G04': 'Horology',

'G05': 'Controlling; Regulating',

'G06': 'Computing; Calculating; Counting',

'G07': 'Checking-devices',

'G08': 'Signalling',

'G09': 'Educating; Cryptography; Display; Advertising; Seals',

'G10': 'Musical Instruments; Acoustics',

'G11': 'Information Storage',

'G12': 'Instrument Details',

'G16': 'Information and Communication Technology [ICT] Specially Adapted for Specific Application Fields',

'G21': 'Nuclear Physics; Nuclear Engineering',

'G99': 'Subject Matter not Provided for in Other Groups of this Section'

}

classes\_H = {

'H01': 'Electric Elements',

'H02': 'Generation; Conversion or Distribution of Electric Power',

'H03': 'Basic Electronic Circuitry',

'H04': 'Electric Communication Technique',

'H05': 'Electric Techniques not Otherwise Provided for',

'H10': 'Semiconductor Devices; Electric Solid State Devices not Otherwise Provided for',

'H99': 'Subject Matter not Provided for in Other Groups of this Section'

}

# Creating a DataFrame

df = pd.DataFrame.from\_dict(results, orient='index', columns=['Precision'])

# Setting a more scientific style with Seaborn

sns.set(style="whitegrid")

plt.figure(figsize=(10, 6))

# Plotting the bar chart

sns.barplot(x=df.index, y='Precision', data=df, palette='viridis')

# Adding labels and title

plt.xlabel('Секции')

plt.ylabel('Точность')

plt.title('Результаты кросс-валидации')

# Displaying the plot

plt.show()

# Creating a DataFrame

df = pd.DataFrame.from\_dict(new\_results, orient='index', columns=['Precision'])

# Setting a more scientific style with Seaborn

sns.set(style="whitegrid")

plt.figure(figsize=(10, 6))

# Plotting the bar chart

sns.barplot(x=df.index, y='Precision', data=df, palette='viridis')

# Adding labels and title

plt.xlabel('Секции')

plt.ylabel('Точность')

plt.title('Результаты классификации')