{

"cells": [

{

"cell\_type": "code",

"execution\_count": null,

"id": "f5274329",

"metadata": {

"id": "f5274329"

},

"outputs": [],

"source": [

"import pandas as pd\n",

"import numpy as np\n",

"import logging\n",

"from numpy import random\n",

"import gensim\n",

"import nltk\n",

"from sklearn.model\_selection import train\_test\_split\n",

"from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer\n",

"from sklearn.metrics import accuracy\_score, confusion\_matrix\n",

"import matplotlib.pyplot as plt\n",

"from nltk.corpus import stopwords\n",

"import re\n",

"from bs4 import BeautifulSoup\n",

"import warnings\n",

"warnings.filterwarnings(\"ignore\")\n",

"from nltk.tokenize import word\_tokenize\n",

"from nltk.corpus import stopwords\n",

"from nltk.stem import SnowballStemmer, WordNetLemmatizer\n",

"from sklearn.feature\_extraction.text import TfidfVectorizer\n",

"\n",

"\n",

"%matplotlib inline"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "f5d49eac",

"metadata": {

"id": "f5d49eac"

},

"outputs": [],

"source": [

"import xgboost as xgb\n",

"from sklearn.ensemble import RandomForestClassifier\n",

"from sklearn.linear\_model import LogisticRegression\n",

"from sklearn.svm import SVC\n",

"from sklearn.neighbors import KNeighborsClassifier\n",

"from sklearn.tree import DecisionTreeClassifier\n",

"from sklearn.ensemble import AdaBoostClassifier\n",

"from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, recall\_score, precision\_score"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "f9273658",

"metadata": {

"id": "f9273658"

},

"outputs": [],

"source": [

"df= pd.read\_csv('train (1) .csv')"

]

},

{

"cell\_type": "code",

"source": [],

"metadata": {

"id": "ZIgbVrJ6tWvK"

},

"id": "ZIgbVrJ6tWvK",

"execution\_count": null,

"outputs": []

},

{

"cell\_type": "markdown",

"id": "c4a5b5f5",

"metadata": {

"id": "c4a5b5f5"

},

"source": [

"## Basic EDA"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "36fb3fd4",

"metadata": {

"id": "36fb3fd4",

"outputId": "7ffdb435-04b2-4cd5-b959-b1403d18bf8d",

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 206

}

},

"outputs": [

{

"output\_type": "execute\_result",

"data": {

"text/plain": [

" unique ID query\_text \\\n",

"0 0 HTTPS://VIMEO.COM/107297364﻿ \n",

"1 1 Art and Culture \n",

"2 2 Differentiate between chemical and biological ... \n",

"3 3 nth Term of an AP \n",

"4 4 bunmei kaika: aoiza ibunroku saien \n",

"\n",

" category \n",

"0 junk \n",

"1 general \n",

"2 academic\_servable \n",

"3 academic\_servable \n",

"4 junk "

],

"text/html": [

"\n",

" <div id=\"df-113b488e-b5b1-435d-89e9-4c76548069b2\" class=\"colab-df-container\">\n",

" <div>\n",

"<style scoped>\n",

" .dataframe tbody tr th:only-of-type {\n",

" vertical-align: middle;\n",

" }\n",

"\n",

" .dataframe tbody tr th {\n",

" vertical-align: top;\n",

" }\n",

"\n",

" .dataframe thead th {\n",

" text-align: right;\n",

" }\n",

"</style>\n",

"<table border=\"1\" class=\"dataframe\">\n",

" <thead>\n",

" <tr style=\"text-align: right;\">\n",

" <th></th>\n",

" <th>unique ID</th>\n",

" <th>query\_text</th>\n",

" <th>category</th>\n",

" </tr>\n",

" </thead>\n",

" <tbody>\n",

" <tr>\n",

" <th>0</th>\n",

" <td>0</td>\n",

" <td>HTTPS://VIMEO.COM/107297364﻿</td>\n",

" <td>junk</td>\n",

" </tr>\n",

" <tr>\n",

" <th>1</th>\n",

" <td>1</td>\n",

" <td>Art and Culture</td>\n",

" <td>general</td>\n",

" </tr>\n",

" <tr>\n",

" <th>2</th>\n",

" <td>2</td>\n",

" <td>Differentiate between chemical and biological ...</td>\n",

" <td>academic\_servable</td>\n",

" </tr>\n",

" <tr>\n",

" <th>3</th>\n",

" <td>3</td>\n",

" <td>nth Term of an AP</td>\n",

" <td>academic\_servable</td>\n",

" </tr>\n",

" <tr>\n",

" <th>4</th>\n",

" <td>4</td>\n",

" <td>bunmei kaika: aoiza ibunroku saien</td>\n",

" <td>junk</td>\n",

" </tr>\n",

" </tbody>\n",

"</table>\n",

"</div>\n",

" <div class=\"colab-df-buttons\">\n",

"\n",

" <div class=\"colab-df-container\">\n",

" <button class=\"colab-df-convert\" onclick=\"convertToInteractive('df-113b488e-b5b1-435d-89e9-4c76548069b2')\"\n",

" title=\"Convert this dataframe to an interactive table.\"\n",

" style=\"display:none;\">\n",

"\n",

" <svg xmlns=\"http://www.w3.org/2000/svg\" height=\"24px\" viewBox=\"0 -960 960 960\">\n",

" <path d=\"M120-120v-720h720v720H120Zm60-500h600v-160H180v160Zm220 220h160v-160H400v160Zm0 220h160v-160H400v160ZM180-400h160v-160H180v160Zm440 0h160v-160H620v160ZM180-180h160v-160H180v160Zm440 0h160v-160H620v160Z\"/>\n",

" </svg>\n",

" </button>\n",

"\n",

" <style>\n",

" .colab-df-container {\n",

" display:flex;\n",

" gap: 12px;\n",

" }\n",

"\n",

" .colab-df-convert {\n",

" background-color: #E8F0FE;\n",

" border: none;\n",

" border-radius: 50%;\n",

" cursor: pointer;\n",

" display: none;\n",

" fill: #1967D2;\n",

" height: 32px;\n",

" padding: 0 0 0 0;\n",

" width: 32px;\n",

" }\n",

"\n",

" .colab-df-convert:hover {\n",

" background-color: #E2EBFA;\n",

" box-shadow: 0px 1px 2px rgba(60, 64, 67, 0.3), 0px 1px 3px 1px rgba(60, 64, 67, 0.15);\n",

" fill: #174EA6;\n",

" }\n",

"\n",

" .colab-df-buttons div {\n",

" margin-bottom: 4px;\n",

" }\n",

"\n",

" [theme=dark] .colab-df-convert {\n",

" background-color: #3B4455;\n",

" fill: #D2E3FC;\n",

" }\n",

"\n",

" [theme=dark] .colab-df-convert:hover {\n",

" background-color: #434B5C;\n",

" box-shadow: 0px 1px 3px 1px rgba(0, 0, 0, 0.15);\n",

" filter: drop-shadow(0px 1px 2px rgba(0, 0, 0, 0.3));\n",

" fill: #FFFFFF;\n",

" }\n",

" </style>\n",

"\n",

" <script>\n",

" const buttonEl =\n",

" document.querySelector('#df-113b488e-b5b1-435d-89e9-4c76548069b2 button.colab-df-convert');\n",

" buttonEl.style.display =\n",

" google.colab.kernel.accessAllowed ? 'block' : 'none';\n",

"\n",

" async function convertToInteractive(key) {\n",

" const element = document.querySelector('#df-113b488e-b5b1-435d-89e9-4c76548069b2');\n",

" const dataTable =\n",

" await google.colab.kernel.invokeFunction('convertToInteractive',\n",

" [key], {});\n",

" if (!dataTable) return;\n",

"\n",

" const docLinkHtml = 'Like what you see? Visit the ' +\n",

" '<a target=\"\_blank\" href=https://colab.research.google.com/notebooks/data\_table.ipynb>data table notebook</a>'\n",

" + ' to learn more about interactive tables.';\n",

" element.innerHTML = '';\n",

" dataTable['output\_type'] = 'display\_data';\n",

" await google.colab.output.renderOutput(dataTable, element);\n",

" const docLink = document.createElement('div');\n",

" docLink.innerHTML = docLinkHtml;\n",

" element.appendChild(docLink);\n",

" }\n",

" </script>\n",

" </div>\n",

"\n",

"\n",

"<div id=\"df-1f8486fe-cbab-4427-81a2-126a24e0509e\">\n",

" <button class=\"colab-df-quickchart\" onclick=\"quickchart('df-1f8486fe-cbab-4427-81a2-126a24e0509e')\"\n",

" title=\"Suggest charts\"\n",

" style=\"display:none;\">\n",

"\n",

"<svg xmlns=\"http://www.w3.org/2000/svg\" height=\"24px\"viewBox=\"0 0 24 24\"\n",

" width=\"24px\">\n",

" <g>\n",

" <path d=\"M19 3H5c-1.1 0-2 .9-2 2v14c0 1.1.9 2 2 2h14c1.1 0 2-.9 2-2V5c0-1.1-.9-2-2-2zM9 17H7v-7h2v7zm4 0h-2V7h2v10zm4 0h-2v-4h2v4z\"/>\n",

" </g>\n",

"</svg>\n",

" </button>\n",

"\n",

"<style>\n",

" .colab-df-quickchart {\n",

" --bg-color: #E8F0FE;\n",

" --fill-color: #1967D2;\n",

" --hover-bg-color: #E2EBFA;\n",

" --hover-fill-color: #174EA6;\n",

" --disabled-fill-color: #AAA;\n",

" --disabled-bg-color: #DDD;\n",

" }\n",

"\n",

" [theme=dark] .colab-df-quickchart {\n",

" --bg-color: #3B4455;\n",

" --fill-color: #D2E3FC;\n",

" --hover-bg-color: #434B5C;\n",

" --hover-fill-color: #FFFFFF;\n",

" --disabled-bg-color: #3B4455;\n",

" --disabled-fill-color: #666;\n",

" }\n",

"\n",

" .colab-df-quickchart {\n",

" background-color: var(--bg-color);\n",

" border: none;\n",

" border-radius: 50%;\n",

" cursor: pointer;\n",

" display: none;\n",

" fill: var(--fill-color);\n",

" height: 32px;\n",

" padding: 0;\n",

" width: 32px;\n",

" }\n",

"\n",

" .colab-df-quickchart:hover {\n",

" background-color: var(--hover-bg-color);\n",

" box-shadow: 0 1px 2px rgba(60, 64, 67, 0.3), 0 1px 3px 1px rgba(60, 64, 67, 0.15);\n",

" fill: var(--button-hover-fill-color);\n",

" }\n",

"\n",

" .colab-df-quickchart-complete:disabled,\n",

" .colab-df-quickchart-complete:disabled:hover {\n",

" background-color: var(--disabled-bg-color);\n",

" fill: var(--disabled-fill-color);\n",

" box-shadow: none;\n",

" }\n",

"\n",

" .colab-df-spinner {\n",

" border: 2px solid var(--fill-color);\n",

" border-color: transparent;\n",

" border-bottom-color: var(--fill-color);\n",

" animation:\n",

" spin 1s steps(1) infinite;\n",

" }\n",

"\n",

" @keyframes spin {\n",

" 0% {\n",

" border-color: transparent;\n",

" border-bottom-color: var(--fill-color);\n",

" border-left-color: var(--fill-color);\n",

" }\n",

" 20% {\n",

" border-color: transparent;\n",

" border-left-color: var(--fill-color);\n",

" border-top-color: var(--fill-color);\n",

" }\n",

" 30% {\n",

" border-color: transparent;\n",

" border-left-color: var(--fill-color);\n",

" border-top-color: var(--fill-color);\n",

" border-right-color: var(--fill-color);\n",

" }\n",

" 40% {\n",

" border-color: transparent;\n",

" border-right-color: var(--fill-color);\n",

" border-top-color: var(--fill-color);\n",

" }\n",

" 60% {\n",

" border-color: transparent;\n",

" border-right-color: var(--fill-color);\n",

" }\n",

" 80% {\n",

" border-color: transparent;\n",

" border-right-color: var(--fill-color);\n",

" border-bottom-color: var(--fill-color);\n",

" }\n",

" 90% {\n",

" border-color: transparent;\n",

" border-bottom-color: var(--fill-color);\n",

" }\n",

" }\n",

"</style>\n",

"\n",

" <script>\n",

" async function quickchart(key) {\n",

" const quickchartButtonEl =\n",

" document.querySelector('#' + key + ' button');\n",

" quickchartButtonEl.disabled = true; // To prevent multiple clicks.\n",

" quickchartButtonEl.classList.add('colab-df-spinner');\n",

" try {\n",

" const charts = await google.colab.kernel.invokeFunction(\n",

" 'suggestCharts', [key], {});\n",

" } catch (error) {\n",

" console.error('Error during call to suggestCharts:', error);\n",

" }\n",

" quickchartButtonEl.classList.remove('colab-df-spinner');\n",

" quickchartButtonEl.classList.add('colab-df-quickchart-complete');\n",

" }\n",

" (() => {\n",

" let quickchartButtonEl =\n",

" document.querySelector('#df-1f8486fe-cbab-4427-81a2-126a24e0509e button');\n",

" quickchartButtonEl.style.display =\n",

" google.colab.kernel.accessAllowed ? 'block' : 'none';\n",

" })();\n",

" </script>\n",

"</div>\n",

"\n",

" </div>\n",

" </div>\n"

]

},

"metadata": {},

"execution\_count": 9

}

],

"source": [

"df.head(5)"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "aaa133d5",

"metadata": {

"id": "aaa133d5",

"outputId": "3ebe49b9-e4e4-4984-ee0f-a580f130cc8c",

"colab": {

"base\_uri": "https://localhost:8080/"

}

},

"outputs": [

{

"output\_type": "execute\_result",

"data": {

"text/plain": [

"unique ID 0\n",

"query\_text 0\n",

"category 0\n",

"dtype: int64"

]

},

"metadata": {},

"execution\_count": 10

}

],

"source": [

"df.isnull().sum()"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "cd3ae862",

"metadata": {

"id": "cd3ae862",

"outputId": "30c7f90b-7115-4def-e2e3-6c38374e5ada",

"colab": {

"base\_uri": "https://localhost:8080/"

}

},

"outputs": [

{

"output\_type": "execute\_result",

"data": {

"text/plain": [

"83817"

]

},

"metadata": {},

"execution\_count": 11

}

],

"source": [

"total\_word\_count = 0\n",

"for text in df['query\_text']:\n",

" total\_word\_count += len(text.split(' '))\n",

"total\_word\_count"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "96a7ea2a",

"metadata": {

"scrolled": true,

"id": "96a7ea2a",

"outputId": "52f42ba6-0303-4cf3-a5fb-f40dff12f147",

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 543

}

},

"outputs": [

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<Figure size 1000x400 with 1 Axes>"

],

"image/png": "\n"

},

"metadata": {}

}

],

"source": [

"import matplotlib.pyplot as plt\n",

"\n",

"category\_counts = df['category'].value\_counts()\n",

"total\_samples = len(df)\n",

"category\_percentages = (category\_counts / total\_samples) \* 100\n",

"\n",

"plt.figure(figsize=(10, 4))\n",

"category\_percentages.plot(kind='bar')\n",

"plt.ylabel('Percentage')\n",

"plt.title('Percentage of Each Category')\n",

"plt.show()\n"

]

},

{

"cell\_type": "markdown",

"id": "6d978822",

"metadata": {

"id": "6d978822"

},

"source": [

"## Feature Engineering"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "822d09fc",

"metadata": {

"id": "822d09fc"

},

"outputs": [],

"source": [

"def preprocess\_text(text):\n",

" text = re.sub(r'http\\S+', '', text) # Remove URLs\n",

" text = re.sub(r'\\d+', '', text) # Remove numbers\n",

" text = re.sub(r'\\W+', ' ', text) # Remove non-alphanumeric characters\n",

" text = text.lower() # Convert text to lowercase\n",

" text = re.sub(r'[^\\w\\s,]', '', text) # Remove emojis\n",

" return text\n"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "96f63ace",

"metadata": {

"id": "96f63ace"

},

"outputs": [],

"source": [

"df['query\_text'] = df['query\_text'].apply(preprocess\_text)"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "45b1d12d",

"metadata": {

"id": "45b1d12d"

},

"outputs": [],

"source": [

"def tokenize\_and\_process(text):\n",

" tokens = word\_tokenize(text)\n",

" stop\_words = set(stopwords.words('english'))\n",

" filtered\_tokens = [token for token in tokens if token.lower() not in stop\_words]\n",

" stemmer = SnowballStemmer('english')\n",

" stemmed\_tokens = [stemmer.stem(token) for token in filtered\_tokens]\n",

" lemmatizer = WordNetLemmatizer()\n",

" lemmatized\_tokens = [lemmatizer.lemmatize(token) for token in stemmed\_tokens]\n",

" return ' '.join(lemmatized\_tokens) # Convert tokens back to text"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "c1420b1e",

"metadata": {

"id": "c1420b1e"

},

"outputs": [],

"source": [

"df['query\_text'] = df['query\_text'].apply(tokenize\_and\_process)"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "e9f56bfe",

"metadata": {

"id": "e9f56bfe"

},

"outputs": [],

"source": [

"X = df['query\_text']\n",

"y = df['category']"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "1d8a2dd4",

"metadata": {

"id": "1d8a2dd4"

},

"outputs": [],

"source": [

"vectorizer = TfidfVectorizer()\n",

"\n",

"X\_tfidf = vectorizer.fit\_transform(X)"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "a53482c0",

"metadata": {

"id": "a53482c0"

},

"outputs": [],

"source": [

"from sklearn.model\_selection import train\_test\_split\n",

"X\_train, X\_test, y\_train, y\_test= train\_test\_split(X\_tfidf, y, test\_size=0.30, random\_state=42)"

]

},

{

"cell\_type": "markdown",

"id": "73a3fcb7",

"metadata": {

"id": "73a3fcb7"

},

"source": [

"## Doing OverSampling on Data"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "da8af5e3",

"metadata": {

"id": "da8af5e3"

},

"outputs": [],

"source": [

"from imblearn.over\_sampling import RandomOverSampler\n",

"\n",

"oversampler = RandomOverSampler(random\_state=42)\n",

"\n",

"X\_train\_balanced, y\_train\_balanced = oversampler.fit\_resample(X\_train, y\_train)\n"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "60ecc0b9",

"metadata": {

"id": "60ecc0b9",

"outputId": "03595689-efe4-4feb-8fa7-35a117d220e5",

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 957

}

},

"outputs": [

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<Figure size 800x500 with 1 Axes>"

],

"image/png": "\n"

},

"metadata": {}

},

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<Figure size 800x500 with 1 Axes>"

],

"image/png": "\n"

},

"metadata": {}

}

],

"source": [

"import matplotlib.pyplot as plt\n",

"\n",

"def plot\_class\_distribution(y, title):\n",

" class\_counts = y.value\_counts()\n",

" class\_labels = class\_counts.index\n",

" plt.figure(figsize=(8, 5))\n",

" plt.bar(class\_labels, class\_counts, color='skyblue')\n",

" plt.xlabel('Class')\n",

" plt.ylabel('Frequency')\n",

" plt.title(title)\n",

" plt.show()\n",

"\n",

"plot\_class\_distribution(y\_train, title='Class Distribution Before Oversampling')\n",

"\n",

"plot\_class\_distribution(y\_train\_balanced, title='Class Distribution After Oversampling')\n"

]

},

{

"cell\_type": "markdown",

"id": "7e4ad590",

"metadata": {

"id": "7e4ad590"

},

"source": [

"## Applying Naive Bayes Classifier"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "b4dd7b37",

"metadata": {

"id": "b4dd7b37"

},

"outputs": [],

"source": [

"from sklearn.naive\_bayes import MultinomialNB\n",

"nb\_classifier = MultinomialNB()\n",

"nb\_classifier.fit(X\_train\_balanced, y\_train\_balanced)\n",

"y\_pred = nb\_classifier.predict(X\_test)"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "2df7d982",

"metadata": {

"id": "2df7d982",

"outputId": "10e94df9-b557-49ba-87b7-e31554a1896d",

"colab": {

"base\_uri": "https://localhost:8080/"

}

},

"outputs": [

{

"output\_type": "stream",

"name": "stdout",

"text": [

"Accuracy: 0.6326666666666667\n",

"Classification Report:\n",

" precision recall f1-score support\n",

"\n",

"academic\_non\_servable 0.51 0.94 0.66 215\n",

" academic\_servable 0.78 0.45 0.57 519\n",

" conversational 0.58 0.78 0.66 153\n",

" general 0.70 0.70 0.70 313\n",

" junk 0.62 0.59 0.60 300\n",

"\n",

" accuracy 0.63 1500\n",

" macro avg 0.64 0.69 0.64 1500\n",

" weighted avg 0.67 0.63 0.63 1500\n",

"\n"

]

}

],

"source": [

"from sklearn.metrics import accuracy\_score, classification\_report\n",

"accuracy = accuracy\_score(y\_test, y\_pred)\n",

"report = classification\_report(y\_test, y\_pred)\n",

"from sklearn.naive\_bayes import MultinomialNB\n",

"nb\_classifier = MultinomialNB()\n",

"nb\_classifier.fit(X\_train, y\_train)\n",

"y\_pred = nb\_classifier.predict(X\_test)\n",

"print(\"Accuracy:\", accuracy)\n",

"print(\"Classification Report:\")\n",

"print(report)"

]

},

{

"cell\_type": "code",

"source": [

"from sklearn.preprocessing import LabelEncoder\n",

"\n",

"label\_encoder = LabelEncoder()\n",

"label\_encoder.fit(y\_train)\n",

"\n",

"y\_test\_encoded = label\_encoder.transform(y\_test)\n"

],

"metadata": {

"id": "jI3TSOKgvB1l"

},

"id": "jI3TSOKgvB1l",

"execution\_count": null,

"outputs": []

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "fd6a53cb",

"metadata": {

"id": "fd6a53cb"

},

"outputs": [],

"source": [

"y\_test\_encoded = label\_encoder.transform(y\_test)"

]

},

{

"cell\_type": "markdown",

"id": "cd6b0eb8",

"metadata": {

"id": "cd6b0eb8"

},

"source": [

"## Applying multiple Ensemble methods"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "a650e55f",

"metadata": {

"id": "a650e55f",

"outputId": "a64ab5cc-e860-4372-c490-25ca4be2996d",

"colab": {

"base\_uri": "https://localhost:8080/",

"height": 92

}

},

"outputs": [

{

"output\_type": "execute\_result",

"data": {

"text/plain": [

"RandomForestClassifier(bootstrap=False, max\_features='log2',\n",

" min\_samples\_split=10, n\_estimators=300)"

],

"text/html": [

"<style>#sk-container-id-1 {color: black;background-color: white;}#sk-container-id-1 pre{padding: 0;}#sk-container-id-1 div.sk-toggleable {background-color: white;}#sk-container-id-1 label.sk-toggleable\_\_label {cursor: pointer;display: block;width: 100%;margin-bottom: 0;padding: 0.3em;box-sizing: border-box;text-align: center;}#sk-container-id-1 label.sk-toggleable\_\_label-arrow:before {content: \"▸\";float: left;margin-right: 0.25em;color: #696969;}#sk-container-id-1 label.sk-toggleable\_\_label-arrow:hover:before {color: black;}#sk-container-id-1 div.sk-estimator:hover label.sk-toggleable\_\_label-arrow:before {color: black;}#sk-container-id-1 div.sk-toggleable\_\_content {max-height: 0;max-width: 0;overflow: hidden;text-align: left;background-color: #f0f8ff;}#sk-container-id-1 div.sk-toggleable\_\_content pre {margin: 0.2em;color: black;border-radius: 0.25em;background-color: #f0f8ff;}#sk-container-id-1 input.sk-toggleable\_\_control:checked~div.sk-toggleable\_\_content {max-height: 200px;max-width: 100%;overflow: auto;}#sk-container-id-1 input.sk-toggleable\_\_control:checked~label.sk-toggleable\_\_label-arrow:before {content: \"▾\";}#sk-container-id-1 div.sk-estimator input.sk-toggleable\_\_control:checked~label.sk-toggleable\_\_label {background-color: #d4ebff;}#sk-container-id-1 div.sk-label input.sk-toggleable\_\_control:checked~label.sk-toggleable\_\_label {background-color: #d4ebff;}#sk-container-id-1 input.sk-hidden--visually {border: 0;clip: rect(1px 1px 1px 1px);clip: rect(1px, 1px, 1px, 1px);height: 1px;margin: -1px;overflow: hidden;padding: 0;position: absolute;width: 1px;}#sk-container-id-1 div.sk-estimator {font-family: monospace;background-color: #f0f8ff;border: 1px dotted black;border-radius: 0.25em;box-sizing: border-box;margin-bottom: 0.5em;}#sk-container-id-1 div.sk-estimator:hover {background-color: #d4ebff;}#sk-container-id-1 div.sk-parallel-item::after {content: \"\";width: 100%;border-bottom: 1px solid gray;flex-grow: 1;}#sk-container-id-1 div.sk-label:hover label.sk-toggleable\_\_label {background-color: #d4ebff;}#sk-container-id-1 div.sk-serial::before {content: \"\";position: absolute;border-left: 1px solid gray;box-sizing: border-box;top: 0;bottom: 0;left: 50%;z-index: 0;}#sk-container-id-1 div.sk-serial {display: flex;flex-direction: column;align-items: center;background-color: white;padding-right: 0.2em;padding-left: 0.2em;position: relative;}#sk-container-id-1 div.sk-item {position: relative;z-index: 1;}#sk-container-id-1 div.sk-parallel {display: flex;align-items: stretch;justify-content: center;background-color: white;position: relative;}#sk-container-id-1 div.sk-item::before, #sk-container-id-1 div.sk-parallel-item::before {content: \"\";position: absolute;border-left: 1px solid gray;box-sizing: border-box;top: 0;bottom: 0;left: 50%;z-index: -1;}#sk-container-id-1 div.sk-parallel-item {display: flex;flex-direction: column;z-index: 1;position: relative;background-color: white;}#sk-container-id-1 div.sk-parallel-item:first-child::after {align-self: flex-end;width: 50%;}#sk-container-id-1 div.sk-parallel-item:last-child::after {align-self: flex-start;width: 50%;}#sk-container-id-1 div.sk-parallel-item:only-child::after {width: 0;}#sk-container-id-1 div.sk-dashed-wrapped {border: 1px dashed gray;margin: 0 0.4em 0.5em 0.4em;box-sizing: border-box;padding-bottom: 0.4em;background-color: white;}#sk-container-id-1 div.sk-label label {font-family: monospace;font-weight: bold;display: inline-block;line-height: 1.2em;}#sk-container-id-1 div.sk-label-container {text-align: center;}#sk-container-id-1 div.sk-container {/\* jupyter's `normalize.less` sets `[hidden] { display: none; }` but bootstrap.min.css set `[hidden] { display: none !important; }` so we also need the `!important` here to be able to override the default hidden behavior on the sphinx rendered scikit-learn.org. See: https://github.com/scikit-learn/scikit-learn/issues/21755 \*/display: inline-block !important;position: relative;}#sk-container-id-1 div.sk-text-repr-fallback {display: none;}</style><div id=\"sk-container-id-1\" class=\"sk-top-container\"><div class=\"sk-text-repr-fallback\"><pre>RandomForestClassifier(bootstrap=False, max\_features=&#x27;log2&#x27;,\n",

" min\_samples\_split=10, n\_estimators=300)</pre><b>In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook. <br />On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.</b></div><div class=\"sk-container\" hidden><div class=\"sk-item\"><div class=\"sk-estimator sk-toggleable\"><input class=\"sk-toggleable\_\_control sk-hidden--visually\" id=\"sk-estimator-id-1\" type=\"checkbox\" checked><label for=\"sk-estimator-id-1\" class=\"sk-toggleable\_\_label sk-toggleable\_\_label-arrow\">RandomForestClassifier</label><div class=\"sk-toggleable\_\_content\"><pre>RandomForestClassifier(bootstrap=False, max\_features=&#x27;log2&#x27;,\n",

" min\_samples\_split=10, n\_estimators=300)</pre></div></div></div></div></div>"

]

},

"metadata": {},

"execution\_count": 30

}

],

"source": [

"best\_logistic\_regression = LogisticRegression(C=10, solver='liblinear')\n",

"best\_svc = SVC(C=1, gamma='scale', kernel='linear')\n",

"best\_kneighbors = KNeighborsClassifier(algorithm='auto', n\_neighbors=3, weights='distance')\n",

"best\_decision\_tree = DecisionTreeClassifier(criterion='gini', max\_depth=None, min\_samples\_leaf=1, min\_samples\_split=2)\n",

"best\_random\_forest = RandomForestClassifier(bootstrap=False, max\_depth=None, max\_features='log2', min\_samples\_leaf=1, min\_samples\_split=10, n\_estimators=300)\n",

"\n",

"best\_logistic\_regression.fit(X\_train\_balanced, y\_train\_balanced)\n",

"best\_svc.fit(X\_train\_balanced, y\_train\_balanced)\n",

"best\_kneighbors.fit(X\_train\_balanced, y\_train\_balanced)\n",

"best\_decision\_tree.fit(X\_train\_balanced, y\_train\_balanced)\n",

"best\_random\_forest.fit(X\_train\_balanced, y\_train\_balanced)\n"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "00b62354",

"metadata": {

"id": "00b62354",

"outputId": "8d6d2fc5-366c-40c3-ea6d-4a64b75fef88",

"colab": {

"base\_uri": "https://localhost:8080/"

}

},

"outputs": [

{

"output\_type": "stream",

"name": "stdout",

"text": [

"Logistic Regression Metrics:\n",

"Training Accuracy: 0.9863525588952071\n",

"Training Classification Report:\n",

" precision recall f1-score support\n",

"\n",

"academic\_non\_servable 1.00 0.99 1.00 1231\n",

" academic\_servable 0.99 0.96 0.98 1231\n",

" conversational 0.99 1.00 0.99 1231\n",

" general 0.99 1.00 1.00 1231\n",

" junk 0.96 0.99 0.97 1231\n",

"\n",

" accuracy 0.99 6155\n",

" macro avg 0.99 0.99 0.99 6155\n",

" weighted avg 0.99 0.99 0.99 6155\n",

"\n",

"\n",

"==================================================\n",

"\n",

"Test Accuracy: 0.72\n",

"Test Classification Report:\n",

" precision recall f1-score support\n",

"\n",

"academic\_non\_servable 0.85 0.80 0.82 215\n",

" academic\_servable 0.75 0.74 0.74 519\n",

" conversational 0.68 0.58 0.63 153\n",

" general 0.83 0.67 0.74 313\n",

" junk 0.56 0.76 0.65 300\n",

"\n",

" accuracy 0.72 1500\n",

" macro avg 0.73 0.71 0.72 1500\n",

" weighted avg 0.74 0.72 0.72 1500\n",

"\n",

"\n",

"SVC Metrics:\n",

"Training Accuracy: 0.9707554833468724\n",

"Training Classification Report:\n",

" precision recall f1-score support\n",

"\n",

"academic\_non\_servable 0.99 0.99 0.99 1231\n",

" academic\_servable 0.97 0.94 0.95 1231\n",

" conversational 0.97 0.99 0.98 1231\n",

" general 0.99 0.96 0.98 1231\n",

" junk 0.94 0.98 0.96 1231\n",

"\n",

" accuracy 0.97 6155\n",

" macro avg 0.97 0.97 0.97 6155\n",

" weighted avg 0.97 0.97 0.97 6155\n",

"\n",

"\n",

"==================================================\n",

"\n",

"Test Accuracy: 0.7193333333333334\n",

"Test Classification Report:\n",

" precision recall f1-score support\n",

"\n",

"academic\_non\_servable 0.84 0.78 0.81 215\n",

" academic\_servable 0.75 0.75 0.75 519\n",

" conversational 0.70 0.58 0.63 153\n",

" general 0.84 0.64 0.73 313\n",

" junk 0.56 0.77 0.65 300\n",

"\n",

" accuracy 0.72 1500\n",

" macro avg 0.74 0.70 0.71 1500\n",

" weighted avg 0.74 0.72 0.72 1500\n",

"\n",

"\n",

"K-Neighbors Classifier Metrics:\n",

"Training Accuracy: 0.9907392363931763\n",

"Training Classification Report:\n",

" precision recall f1-score support\n",

"\n",

"academic\_non\_servable 1.00 1.00 1.00 1231\n",

" academic\_servable 0.96 1.00 0.98 1231\n",

" conversational 1.00 1.00 1.00 1231\n",

" general 1.00 1.00 1.00 1231\n",

" junk 1.00 0.96 0.98 1231\n",

"\n",

" accuracy 0.99 6155\n",

" macro avg 0.99 0.99 0.99 6155\n",

" weighted avg 0.99 0.99 0.99 6155\n",

"\n",

"\n",

"==================================================\n",

"\n",

"Test Accuracy: 0.504\n",

"Test Classification Report:\n",

" precision recall f1-score support\n",

"\n",

"academic\_non\_servable 0.84 0.35 0.50 215\n",

" academic\_servable 0.42 0.91 0.58 519\n",

" conversational 0.76 0.29 0.42 153\n",

" general 0.77 0.35 0.48 313\n",

" junk 0.62 0.17 0.26 300\n",

"\n",

" accuracy 0.50 1500\n",

" macro avg 0.68 0.42 0.45 1500\n",

" weighted avg 0.63 0.50 0.47 1500\n",

"\n",

"\n",

"Decision Tree Classifier Metrics:\n",

"Training Accuracy: 0.9907392363931763\n",

"Training Classification Report:\n",

" precision recall f1-score support\n",

"\n",

"academic\_non\_servable 1.00 1.00 1.00 1231\n",

" academic\_servable 0.96 1.00 0.98 1231\n",

" conversational 1.00 1.00 1.00 1231\n",

" general 1.00 1.00 1.00 1231\n",

" junk 1.00 0.96 0.98 1231\n",

"\n",

" accuracy 0.99 6155\n",

" macro avg 0.99 0.99 0.99 6155\n",

" weighted avg 0.99 0.99 0.99 6155\n",

"\n",

"\n",

"==================================================\n",

"\n",

"Test Accuracy: 0.6246666666666667\n",

"Test Classification Report:\n",

" precision recall f1-score support\n",

"\n",

"academic\_non\_servable 0.81 0.80 0.80 215\n",

" academic\_servable 0.55 0.80 0.65 519\n",

" conversational 0.59 0.40 0.47 153\n",

" general 0.78 0.58 0.66 313\n",

" junk 0.55 0.36 0.44 300\n",

"\n",

" accuracy 0.62 1500\n",

" macro avg 0.66 0.59 0.61 1500\n",

" weighted avg 0.64 0.62 0.62 1500\n",

"\n",

"\n",

"Random Forest Classifier Metrics:\n",

"Training Accuracy: 0.9896019496344436\n",

"Training Classification Report:\n",

" precision recall f1-score support\n",

"\n",

"academic\_non\_servable 1.00 0.99 1.00 1231\n",

" academic\_servable 0.96 1.00 0.98 1231\n",

" conversational 1.00 1.00 1.00 1231\n",

" general 1.00 1.00 1.00 1231\n",

" junk 1.00 0.96 0.98 1231\n",

"\n",

" accuracy 0.99 6155\n",

" macro avg 0.99 0.99 0.99 6155\n",

" weighted avg 0.99 0.99 0.99 6155\n",

"\n",

"\n",

"==================================================\n",

"\n",

"Test Accuracy: 0.738\n",

"Test Classification Report:\n",

" precision recall f1-score support\n",

"\n",

"academic\_non\_servable 0.87 0.86 0.87 215\n",

" academic\_servable 0.65 0.87 0.75 519\n",

" conversational 0.74 0.65 0.69 153\n",

" general 0.86 0.65 0.74 313\n",

" junk 0.75 0.56 0.64 300\n",

"\n",

" accuracy 0.74 1500\n",

" macro avg 0.77 0.72 0.74 1500\n",

" weighted avg 0.75 0.74 0.73 1500\n",

"\n"

]

}

],

"source": [

"from sklearn.metrics import accuracy\_score, classification\_report\n",

"\n",

"def print\_metrics(model, X\_train\_balanced, y\_train\_balanced, X\_test, y\_test\_encoded):\n",

" train\_predictions = model.predict(X\_train\_balanced)\n",

" train\_accuracy = accuracy\_score(y\_train\_balanced, train\_predictions)\n",

" train\_report = classification\_report(y\_train\_balanced, train\_predictions)\n",

"\n",

" test\_predictions = model.predict(X\_test)\n",

" test\_accuracy = accuracy\_score(y\_test, test\_predictions)\n",

" test\_report = classification\_report(y\_test, test\_predictions)\n",

"\n",

" print(f\"Training Accuracy: {train\_accuracy}\")\n",

" print(\"Training Classification Report:\")\n",

" print(train\_report)\n",

"\n",

" print(\"\\n\" + \"=\"\*50 + \"\\n\")\n",

"\n",

" print(f\"Test Accuracy: {test\_accuracy}\")\n",

" print(\"Test Classification Report:\")\n",

" print(test\_report)\n",

"\n",

"print(\"Logistic Regression Metrics:\")\n",

"print\_metrics(best\_logistic\_regression, X\_train\_balanced, y\_train\_balanced, X\_test, y\_test\_encoded)\n",

"\n",

"print(\"\\nSVC Metrics:\")\n",

"print\_metrics(best\_svc, X\_train\_balanced, y\_train\_balanced, X\_test, y\_test\_encoded)\n",

"\n",

"print(\"\\nK-Neighbors Classifier Metrics:\")\n",

"print\_metrics(best\_kneighbors, X\_train\_balanced, y\_train\_balanced, X\_test, y\_test\_encoded)\n",

"\n",

"print(\"\\nDecision Tree Classifier Metrics:\")\n",

"print\_metrics(best\_decision\_tree, X\_train\_balanced, y\_train\_balanced, X\_test, y\_test\_encoded)\n",

"\n",

"print(\"\\nRandom Forest Classifier Metrics:\")\n",

"print\_metrics(best\_random\_forest, X\_train\_balanced, y\_train\_balanced, X\_test, y\_test\_encoded)\n"

]

},

{

"cell\_type": "code",

"source": [

"import nltk\n",

"nltk.download('punkt')\n",

"import nltk\n",

"nltk.download('stopwords')\n",

"nltk.download('wordnet')\n",

"\n",

"\n"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/"

},

"id": "8maSKKKNwCT2",

"outputId": "366a88bf-8fb7-4bdd-9674-7c4995c575ef"

},

"id": "8maSKKKNwCT2",

"execution\_count": null,

"outputs": [

{

"output\_type": "stream",

"name": "stderr",

"text": [

"[nltk\_data] Downloading package punkt to /root/nltk\_data...\n",

"[nltk\_data] Package punkt is already up-to-date!\n",

"[nltk\_data] Downloading package stopwords to /root/nltk\_data...\n",

"[nltk\_data] Package stopwords is already up-to-date!\n",

"[nltk\_data] Downloading package wordnet to /root/nltk\_data...\n"

]

},

{

"output\_type": "execute\_result",

"data": {

"text/plain": [

"True"

]

},

"metadata": {},

"execution\_count": 39

}

]

},

{

"cell\_type": "markdown",

"id": "83e61e8f",

"metadata": {

"id": "83e61e8f"

},

"source": [

"## Creating model pipeline"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "d892c488",

"metadata": {

"id": "d892c488",

"outputId": "182ebe4a-e9fc-43c2-d964-4db7f4e2c089",

"colab": {

"base\_uri": "https://localhost:8080/"

}

},

"outputs": [

{

"output\_type": "stream",

"name": "stdout",

"text": [

"Accuracy: 0.7006666666666667\n",

"Classification Report:\n",

" precision recall f1-score support\n",

"\n",

"academic\_non\_servable 0.92 0.77 0.84 215\n",

" academic\_servable 0.73 0.71 0.72 519\n",

" conversational 0.67 0.57 0.61 153\n",

" general 0.83 0.65 0.73 313\n",

" junk 0.52 0.76 0.62 300\n",

"\n",

" accuracy 0.70 1500\n",

" macro avg 0.73 0.69 0.70 1500\n",

" weighted avg 0.73 0.70 0.71 1500\n",

"\n"

]

}

],

"source": [

"def preprocess\_text(text):\n",

" text = re.sub(r'http\\S+', '', text) # Remove URLs\n",

" text = re.sub(r'\\d+', '', text) # Remove numbers\n",

" text = re.sub(r'\\W+', ' ', text) # Remove non-alphanumeric characters\n",

" text = text.lower() # Convert text to lowercase\n",

" text = re.sub(r'[^\\w\\s,]', '', text) # Remove emojis\n",

" return text\n",

"\n",

"def tokenize\_and\_process(text):\n",

" tokens = word\_tokenize(text)\n",

" stop\_words = set(stopwords.words('english'))\n",

" filtered\_tokens = [token for token in tokens if token.lower() not in stop\_words]\n",

" stemmer = SnowballStemmer('english')\n",

" stemmed\_tokens = [stemmer.stem(token) for token in filtered\_tokens]\n",

" lemmatizer = WordNetLemmatizer()\n",

" lemmatized\_tokens = [lemmatizer.lemmatize(token) for token in stemmed\_tokens]\n",

" return ' '.join(lemmatized\_tokens)\n",

"\n",

"df = pd.read\_csv('train (1) .csv')\n",

"\n",

"df['query\_text'] = df['query\_text'].apply(preprocess\_text)\n",

"df['query\_text'] = df['query\_text'].apply(tokenize\_and\_process)\n",

"\n",

"X = df['query\_text']\n",

"y = df['category']\n",

"\n",

"vectorizer = TfidfVectorizer()\n",

"\n",

"X\_tfidf = vectorizer.fit\_transform(X)\n",

"\n",

"X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_tfidf, y, test\_size=0.30, random\_state=42)\n",

"\n",

"oversampler = RandomOverSampler(random\_state=42)\n",

"X\_train\_balanced, y\_train\_balanced = oversampler.fit\_resample(X\_train, y\_train)\n",

"\n",

"random\_forest\_classifier = RandomForestClassifier(bootstrap=False, max\_depth=None, max\_features='log2', min\_samples\_leaf=1, min\_samples\_split=10, n\_estimators=300)\n",

"random\_forest\_classifier.fit(X\_train\_balanced, y\_train\_balanced)\n",

"\n",

"def make\_predictions\_and\_print\_metrics(model, X\_test, y\_test):\n",

" y\_pred = model.predict(X\_test)\n",

" accuracy = accuracy\_score(y\_test, y\_pred)\n",

" report = classification\_report(y\_test, y\_pred)\n",

" print(\"Accuracy:\", accuracy)\n",

" print(\"Classification Report:\")\n",

" print(report)\n",

"\n",

"make\_predictions\_and\_print\_metrics(random\_forest\_classifier, X\_test, y\_test)\n"

]

},

{

"cell\_type": "markdown",

"id": "69338d55",

"metadata": {

"id": "69338d55"

},

"source": [

"## Testing model on test.csv"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "1ab9bcbe",

"metadata": {

"id": "1ab9bcbe",

"outputId": "0fe42fe7-c183-4593-9d8d-77b086ad0d6b",

"colab": {

"base\_uri": "https://localhost:8080/"

}

},

"outputs": [

{

"output\_type": "execute\_result",

"data": {

"text/plain": [

"['random\_forest\_classifier.pkl']"

]

},

"metadata": {},

"execution\_count": 41

}

],

"source": [

"import joblib\n",

"\n",

"joblib.dump(random\_forest\_classifier, 'random\_forest\_classifier.pkl')"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "a0e5ad0e",

"metadata": {

"id": "a0e5ad0e"

},

"outputs": [],

"source": [

"import pandas as pd\n",

"import joblib\n",

"\n",

"random\_forest\_classifier = joblib.load('random\_forest\_classifier.pkl')\n",

"\n",

"test\_df = pd.read\_csv('test (2).csv')\n",

"\n",

"test\_df['query\_text'] = test\_df['query\_text'].apply(preprocess\_text)\n",

"test\_df['query\_text'] = test\_df['query\_text'].apply(tokenize\_and\_process)\n",

"\n",

"X\_test\_tfidf = vectorizer.transform(test\_df['query\_text'])\n",

"\n",

"predictions = random\_forest\_classifier.predict(X\_test\_tfidf)\n",

"\n",

"submission\_df = pd.DataFrame({'query\_text': test\_df['query\_text'], 'prediction': predictions})\n",

"\n",

"submission\_df.to\_csv('sample\_submission.csv', index=False)\n"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"id": "aa94ae6e",

"metadata": {

"id": "aa94ae6e"

},

"outputs": [],

"source": []

}

],

"metadata": {

"kernelspec": {

"display\_name": "Python 3 (ipykernel)",

"language": "python",

"name": "python3"

},

"language\_info": {

"codemirror\_mode": {

"name": "ipython",

"version": 3

},

"file\_extension": ".py",

"mimetype": "text/x-python",

"name": "python",

"nbconvert\_exporter": "python",

"pygments\_lexer": "ipython3",

"version": "3.9.12"

},

"colab": {

"provenance": []

}

},

"nbformat": 4,

"nbformat\_minor": 5

}