{ "cells": [{ "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "Tfl_Mt5IpdTE" }, "source": ["# DonorsChoose"] }, { "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "zPv-04PipdTH" }, "source": ["

\n", "DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of funding. Right now, a large number of volunteers is needed to manually screen each submission before it's approved to be posted on the DonorsChoose.org website.\n", "

\n", "

\n", " Next year, DonorsChoose.org expects to receive close to 500,000 project proposals. As a result, there are three main problems they need to solve:\n", "

\n", "
- \n", " How to scale current manual processes and resources to screen 500,000 projects so that they can be posted as quickly and as efficiently as possible
\n", "
- How to increase the consistency of project vetting across different volunteers to improve the experience for teachers
\n", "
- How to focus volunteer time on the applications that need the most assistance
\n", "

\n", "

\n", "

\n", "The goal of the competition is to predict whether or not a DonorsChoose.org project proposal submitted by a teacher will be approved, using the text of project descriptions as well as additional metadata about the project, teacher, and school. DonorsChoose.org can then use this information to identify projects most likely to need further review before approval.\n", "

"] }, { "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "fVyva7F1pdTK" }, "source": ["## About the DonorsChoose Data Set\n", "\n", "The `train.csv` data set provided by DonorsChoose contains the following features:\n", "\n", "Feature | Description \n", "----------|---------------\n", "**`project_id`** | A unique identifier for the proposed project. **Example:** `p036502` \n", "**`project_title`** | Title of the project. **Examples:**

- `Art Will Make You Happy!`
- `First Grade Fun`

\n", "**`project_grade_category`** | Grade level of students for which the project is targeted. One of the following enumerated values:

- `Grades PreK-2`
- `Grades 3-5`
- `Grades 6-8`
- `Grades 9-12`

\n", " **`project_subject_categories`** | One or more (comma-separated) subject categories for the project from the following enumerated list of values:

- `Applied Learning`
- `Care & Hunger`
- `Health & Sports`
- `History & Civics`
- `Literacy & Language`
- `Math & Science`
- `Music & The Arts`
- `Special Needs`
- `Warmth`

Examples:

- `Music & The Arts`
- `Literacy & Language, Math & Science`
  \n", " **`school_state`** | State where school is located ([Two-letter U.S. postal code](https://en.wikipedia.org/wiki/List_of_U.S._state_abbreviations#Postal_cod **Example:** `WY`\n", "**`project_subject_subcategories`** | One or more (comma-separated) subject subcategories for the project. **Examples:**
    - `Literacy`

- Literature & Writing, Social Sciences

\n", "**`project_resource_summary`** | An explanation of the resources needed for the project. **Example:**

- My students need hands on literacy materials to manage sensory needs!

\n", "**`project_essay_1`** | First application essay* \n", "**`project_essay_2`** | Second application essay* \n", "**`project_essay_3`** | Third application essay* \n", "**`project_essay_4`** | Fourth application essay* \n", "**`project_submitted_datetime`** | Datetime when project application was submitted. **Example:** `2016-04-28 12:43:56.245` \n", "**`teacher_id`** | A unique identifier for the teacher of the proposed project. **Example:** `bdf8baa8fedef6bfeec7ae4ff1c15c56` \n", "**`teacher_prefix`** | Teacher's title. One of the following enumerated values:

- nan
- Dr.
- Mr.
- Mrs.
- Ms.
- Teacher.

\n", "**`teacher_number_of_previously_posted_projects`** | Number of project applications previously submitted by the same teacher. **Example:** `2` \n", "\n", "* See the section **Notes on the Essay Data** for more details about these features.\n", "\n", "Additionally, the `resources.csv` data set provides more data about the resources required for each project. Each line in this file represents a resource required by a project:\n", "\n", "Feature | Description \n", "----------|---------------\n", "**`id`** | A `project_id` value from the `train.csv` file. **Example:** `p036502` \n", "**`description`** | Desciption of the resource. **Example:** `Tenor Saxophone Reeds, Box of 25` \n", "**`quantity`** | Quantity of the resource required. **Example:** `3` \n", "**`price`** | Price of the resource required. **Example:** `9.95` \n", "\n", "**Note:** Many projects require multiple resources. The `id` value corresponds to a `project_id` in train.csv, so you use it as a key to retrieve all resources needed for a project:\n", "\n", "The data set contains the following label (the value you will attempt to predict):\n", "\n", "Label | Description\n", "----------|---------------\n", "`project_is_approved` | A binary flag indicating whether DonorsChoose approved the project. A value of `0` indicates the project was not approved, and a value of `1` indicates the project was approved."] }, { "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "aUWd8jwjpdTO" }, "source": ["### Notes on the Essay Data\n", "\n", "

\n", "Prior to May 17, 2016, the prompts for the essays were as follows:\n", "

- __project_essay_1:__ \"Introduce us to your classroom\" \n", "
- __project_essay_2:__ \"Tell us more about your students\" \n", "
- __project_essay_3:__ \"Describe how your students will use the materials you're requesting\" \n", "
- __project_essay_3:__ \"Close by sharing why your project will make a difference\" \n", "

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

\n", "Starting on May 17, 2016, the number of essays was reduced from 4 to 2, and the prompts for the first 2 essays were changed to the following: \n", "

- __project_essay_1:__ \"Describe your students: What makes your students special? Specific details about their background, your neighborhood, and your school are all helpful.\" \n", "
- __project_essay_2:__ \"About your project: How will these materials make a difference in your students' learning and improve their school lives?\" \n", "

For all projects with project_submitted_datetime of 2016-05-17 and later, the values of project_essay_3 and project_essay_4 will be NaN.\n", "

\n"] }, { "cell_type": "code", "execution_count": 1, "metadata": { "colab": {}, "colab_type": "code", "id": "ooCk-ogypdTQ", "outputId": "b7affd77-ef78-46ac-ea64-bf1b2252106d" }, "outputs": [{ "name": "stderr", "output_type": "stream", "text": ["C:\\Users\\Public\\Anaconda3\\lib\\site-packages\\gensim\\utils.py:1197: UserWarning: detected Windows; aliasing chunkize to chunkize_serial\n", " warnings.warn(\"detected Windows; aliasing chunkize to chunkize_serial\")\n"] }, { "data": { "text/html": [" \n", " "] }, "metadata": {}, "output_type": "display_data" }], "source": ["%matplotlib inline\n", "import warnings\n", "warnings.filterwarnings(\"ignore\")\n", "\n", "import sqlite3\n", "import pandas as pd\n", "import numpy as np\n", "import nltk\n", "import string\n", "import matplotlib.pyplot as plt\n", "import seaborn as sns\n", "from sklearn.feature_extraction.text import TfidfTransformer\n", "from sklearn.feature_extraction.text import TfidfVectorizer\n", "\n", "from sklearn.feature_extraction.text import CountVectorizer\n", "from sklearn.metrics import confusion_matrix\n", "from sklearn import metrics\n", "from sklearn.metrics import roc_curve, auc\n", "from

nltk.stem.porter import PorterStemmer\n", "\n", "import re\n", "# Tutorial about Python regular expressions: https://pymotw.com/2/re/\n", "import string\n", "from nltk.corpus import stopwords\n", "from nltk.stem import PorterStemmer\n", "from nltk.stem.wordnet import WordNetLemmatizer\n", "\n", "from gensim.models import Word2Vec\n", "from gensim.models import KeyedVectors\n", "import pickle\n", "\n", "from tqdm import tqdm\n", "import os\n", "\n", "from plotly import plotly\n", "import plotly.offline as offline\n", "import plotly.graph_objs as go\n", "offline.init_notebook_mode()\n", "from collections import Counter"] }, { "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "6ZiFnmK5pdTb" }, "source": ["## 1.1 Reading Data"] }, { "cell_type": "code", "execution_count": 2, "metadata": { "colab": {}, "colab_type": "code", "id": "heCL7r2YpdTc" }, "outputs": [], "source": ["project_data = pd.read_csv('train_data.csv')\n", "resource_data = pd.read_csv('resources.csv')"] }, { "cell_type": "code", "execution_count": 3, "metadata": { "colab": {}, "colab_type": "code", "id": "4VOQhd4mpdTg", "outputId": "d40e533a-df7c-4de8-8909-3fb9506efa46" }, "outputs": [{ "name": "stdout", "output_type": "stream", "text": ["Number of data points in train data (109248, 17)\n", "--\n", "The attributes of data : ['Unnamed: 0' 'id' 'teacher_id' 'teacher_prefix' 'school_state'\n", " 'project_submitted_datetime' 'project_grade_category'\n", " 'project_subject_categories' 'project_subject_subcategories'\n", " 'project_title' 'project_essay_1' 'project_essay_2' 'project_essay_3'\n", " 'project_essay_4' 'project_resource_summary'\n", " 'teacher_number_of_previously_posted_projects' 'project_is_approved']\n"] }], "source": ["print(\"Number of data points in train data\", project_data.shape)\n", "print('-'*50)\n", "print(\"The attributes of data :\", project_data.columns.values)"] }, { "cell_type": "code", "execution_count": 4, "metadata": { "colab": {}, "colab_type": "code", "id": "RYA9DT_LpdTl", "outputId": "86faf2f2-e396-43db-eae6-f36482f83b47" }, "outputs": [{ "name": "stdout", "output_type": "stream", "text": ["Number of data points in train data (1541272, 4)\n", "['id' 'description' 'quantity' 'price']\n"] }, { "data": { "text/html": ["
\n", "

	id	description	quantity	price
0	p233245	LC652 - Lakeshore Double-Space Mobile Drying Rack	1	149.00
1	p069063	Bouncy Bands for Desks (Blue support pipes)	3	14.95

\n", "
"], "text/plain": [" id description quantity \\\n", "0
p233245 LC652 - Lakeshore Double-Space Mobile Drying Rack 1
\n", "1 p069063 Bouncy Bands for Desks (Blue support pipes) 3
\n", "\n", " price \n", "0 149.00 \n", "1 14.95 "] },
"execution_count": 4, "metadata": {}, "output_type":
"execute_result" }], "source": ["print(\"Number of data
points in train data\", resource_data.shape)\n",
"print(resource_data.columns.values)\n",
"resource_data.head(2)"] }, { "cell_type": "markdown",
"metadata": { "colab_type": "text", "id": "QgDz1hyFpdTp" },
"source": ["## 1.2 preprocessing of
`project_subject_categories`"] }, { "cell_type": "code",
"execution_count": 5, "metadata": { "colab": {}, "colab_type":
"code", "id": "-WWO7TUTpdTq" }, "outputs": [], "source": [
"catogories =
list(project_data['project_subject_categories'].values)\n", "#
remove special characters from list of strings python:
https://stackoverflow.com/a/47301924/4084039\n", "\n", "#
https://www.geeksforgeeks.org/removing-stop-words-nltk-
python/\n", "#
https://stackoverflow.com/questions/23669024/how-to-strip-a-
specific-word-from-a-string\n", "#
https://stackoverflow.com/questions/8270092/remove-all-
whitespace-in-a-string-in-python\n", "cat_list = []\n", "for i
in catogories:\n", " temp = \"\"\n", " # consider we have text
like this \"Math & Science, Warmth, Care & Hunger\"\n", " for j
in i.split(','): # it will split it in three parts [\"Math &
Science\", \"Warmth\", \"Care & Hunger\"]\n", " if 'The' in
j.split(): # this will split each of the catogory based on
space \"Math & Science\"=> \"Math\",\"&\", \"Science\"\n", "
j=j.replace('The','') # if we have the words \"The\" we are
going to replace it with ''(i.e removing 'The')\n", " j =
j.replace(' ','') # we are placeing all the ' '(space) with
''(empty) ex:\"Math & Science\"=>\"Math&Science\"\n", "
temp+=j.strip()+\" \" #\" abc \".strip() will return \"abc\",
remove the trailing spaces\n", " temp = temp.replace('&','_') #
we are replacing the & value into \n", "
cat_list.append(temp.strip())\n", " \n",
"project_data['clean_categories'] = cat_list\n",
"project_data.drop(['project_subject_categories'], axis=1,
inplace=True)\n", "\n", "from collections import Counter\n",
"my_counter = Counter()\n", "for word in
project_data['clean_categories'].values:\n", "
my_counter.update(word.split())\n", "\n", "cat_dict =
dict(my_counter)\n", "sorted_cat_dict =
dict(sorted(cat_dict.items(), key=lambda kv: kv[1]))\n"] }, {
"cell_type": "markdown", "metadata": { "colab_type": "text",
"id": "bfR_ds-updTt" }, "source": ["## 1.3 preprocessing of
`project_subject_subcategories`"] }, { "cell_type": "code",

"execution_count": 6, "metadata": { "colab": {}, "colab_type": "code", "id": "brFdNF6HpdTt" }, "outputs": [], "source": ["sub_catogories = list(project_data['project_subject_subcategories'].values)\n", "# remove special characters from list of strings python: https://stackoverflow.com/a/47301924/4084039\n", "\n", "# https://www.geeksforgeeks.org/removing-stop-words-nltk-python/\n", "# https://stackoverflow.com/questions/23669024/how-to-strip-a-specific-word-from-a-string\n", "# https://stackoverflow.com/questions/8270092/remove-all-whitespace-in-a-string-in-python\n", "\n", "sub_cat_list = []\n", "for i in sub_catogories:\n", " temp = \"\"\n", " # consider we have text like this \"Math & Science, Warmth, Care & Hunger\"\n", " for j in i.split(','): # it will split it in three parts [\"Math & Science\", \"Warmth\", \"Care & Hunger\"]\n", " if 'The' in j.split(): # this will split each of the catogory based on space \"Math & Science\"=> \"Math\",\"&\", \"Science\"\n", " j=j.replace('The','') # if we have the words \"The\" we are going to replace it with ''(i.e removing 'The')\n", " j = j.replace(' ','') # we are placeing all the ' '(space) with ''(empty) ex:\"Math & Science\"=>\"Math&Science\"\n", " temp +=j.strip()+\" \"#\" abc \".strip() will return \"abc\", remove the trailing spaces\n", " temp = temp.replace('&','_')\n", " sub_cat_list.append(temp.strip())\n", "\n", "project_data['clean_subcategories'] = sub_cat_list\n", "project_data.drop(['project_subject_subcategories'], axis=1, inplace=True)\n", "\n", "# count of all the words in corpus python: https://stackoverflow.com/a/22898595/4084039\n", "my_counter = Counter()\n", "for word in project_data['clean_subcategories'].values:\n", " my_counter.update(word.split())\n", " \n", "sub_cat_dict = dict(my_counter)\n", "sorted_sub_cat_dict = dict(sorted(sub_cat_dict.items(), key=lambda kv: kv[1]))"] }, { "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "P9wpaW9vpdTw" }, "source": ["## 1.3 Text preprocessing"] }, { "cell_type": "code", "execution_count": 7, "metadata": { "colab": {}, "colab_type": "code", "id": "IFdjWTgjpdTx" }, "outputs": [], "source": ["# merge two column text dataframe: \n", "project_data[\"essay\"] = project_data[\"project_essay_1\"].map(str) +\\\n", " project_data[\"project_essay_2\"].map(str) + \\\n", " project_data[\"project_essay_3\"].map(str) + \\\n", " project_data[\"project_essay_4\"].map(str)"] }, { "cell_type": "code", "execution_count": 8, "metadata": { "colab": {}, "colab_type": "code", "id": "-XgKYuMppdTz", "outputId": "a40b6156-4d4c-4311-bc28-dad2cfcfd02d" }, "outputs": [{ "data": { "text/html": ["
\n", "

	Unnamed: 0	id	teacher_id	teacher_pref
0	160221	p253737	c90749f5d961ff158d4b4d1e7dc665fc	Mrs.
1	140945	p258326	897464ce9ddc600bced1151f324dd63a	Mr.

\n", "
"], "text/plain": [" Unnamed: 0 id teacher_id teacher_prefix \\\n", "0 160221 p253737 c90749f5d961ff158d4b4d1e7dc665fc Mrs. \n", "1 140945 p258326 897464ce9ddc600bced1151f324dd63a Mr. \n", "\n", " school_state project_submitted_datetime project_grade_category \\\n", "0 IN 2016-12-05 13:43:57 Grades PreK-2 \n", "1 FL 2016-10-25 09:22:10 Grades 6-8 \n", "\n", " project_title \\\n", "0 Educational Support for English Learners at Home \n", "1 Wanted: Projector for Hungry Learners \n", "\n", " project_essay_1 \\\n", "0 My students are English learners that are work... \n", "1 Our students arrive to our school eager to lea... \n", "\n", " project_essay_2 project_essay_3 \\\n", "0 \\\"The limits of your language are the limits o... NaN \n", "1 The projector we need for our school is very c... NaN \n", "\n", " project_essay_4 project_resource_summary \\\n", "0 NaN My students need opportunities to practice beg... \n", "1 NaN My students need a projector to help with view... \n", "\n", " teacher_number_of_previously_posted_projects project_is_approved \\\n", "0 0 0 \n", "1 7 1 \n", "\n", " clean_categories clean_subcategories \\\n", "0 Literacy_Language ESL Literacy \n", "1 History_Civics Health_Sports Civics_Government TeamSports \n", "\n", " essay \n", "0 My students are English learners that are work... \n", "1 Our students arrive to our school eager to lea... "] }, "execution_count": 8, "metadata": {}, "output_type": "execute_result" }], "source": ["project_data.head(2)"] }, { "cell_type": "code", "execution_count": 9, "metadata": { "colab": {}, "colab_type": "code", "id": "xlR2k0Y0pdT2" }, "outputs": [], "source": ["#### 1.4.2.3 Using Pretrained Models: TFIDF weighted W2V"] }, { "cell_type": "code", "execution_count": 10, "metadata": { "colab": {}, "colab_type":

"code", "id": "mwcmY66LpdT3", "outputId": "6bd6a107-1a40-4e45-f4e1-6bf8e745a882" }, "outputs": [{ "name": "stdout", "output_type": "stream", "text": ["My students are English learners that are working on English as their second or third languages. We are a melting pot of refugees, immigrants, and native-born Americans bringing the gift of language to our school. \\r\\n\\r\\n We have over 24 languages represented in our English Learner program with students at every level of mastery. We also have over 40 countries represented with the families within our school. Each student brings a wealth of knowledge and experiences to us that open our eyes to new cultures, beliefs, and respect.\\\"The limits of your language are the limits of your world.\\\"-Ludwig Wittgenstein Our English learner's have a strong support system at home that begs for more resources. Many times our parents are learning to read and speak English along side of their children. Sometimes this creates barriers for parents to be able to help their child learn phonetics, letter recognition, and other reading skills.\\r\\n\\r\\nBy providing these dvd's and players, students are able to continue their mastery of the English language even if no one at home is able to assist. All families with students within the Level 1 proficiency status, will be a offered to be a part of this program. These educational videos will be specially chosen by the English Learner Teacher and will be sent home regularly to watch. The videos are to help the child develop early reading skills.\\r\\n\\r\\nParents that do not have access to a dvd player will have the opportunity to check out a dvd player to use for the year. The plan is to use these videos and educational dvd's for the years to come for other EL students.\\r\\nnannan\n", "==\n", "The 51 fifth grade students that will cycle through my classroom this year all love learning, at least most of the time. At our school, 97.3% of the students receive free or reduced price lunch. Of the 560 students, 97.3% are minority students. \\r\\nThe school has a vibrant community that loves to get together and celebrate. Around Halloween there is a whole school parade to show off the beautiful costumes that students wear. On Cinco de Mayo we put on a big festival with crafts made by the students, dances, and games. At the end of the year the school hosts a carnival to celebrate the hard work put in during the school year, with a dunk tank being the most popular activity.My students will use these five brightly colored Hokki stools in place of regular, stationary, 4-legged chairs. As I will only have a total of ten in the classroom and not enough for each student to have an individual one, they will be used in a variety of ways. During independent reading time they will be used as special chairs students will each use on occasion. I will utilize them in place of chairs at my small group tables during math and reading times. The rest of the day they will be

used by the students who need the highest amount of movement in
their life in order to stay focused on
school.\\r\\n\\r\\nWhenever asked what the classroom is
missing, my students always say more Hokki Stools. They can't
get their fill of the 5 stools we already have. When the
students are sitting in group with me on the Hokki Stools, they
are always moving, but at the same time doing their work.
Anytime the students get to pick where they can sit, the Hokki
Stools are the first to be taken. There are always students who
head over to the kidney table to get one of the stools who are
disappointed as there are not enough of them. \\r\\n\\r\\nWe
ask a lot of students to sit for 7 hours a day. The Hokki
stools will be a compromise that allow my students to do desk
work and move at the same time. These stools will help students
to meet their 60 minutes a day of movement by allowing them to
activate their core muscles for balance while they sit. For
many of my students, these chairs will take away the barrier
that exists in schools for a child who can't sit
still.nannan\n",
"==\n", "How do
you remember your days of school? Was it in a sterile
environment with plain walls, rows of desks, and a teacher in
front of the room? A typical day in our room is nothing like
that. I work hard to create a warm inviting themed room for my
students look forward to coming to each day.\\r\\n\\r\\nMy
class is made up of 28 wonderfully unique boys and girls of
mixed races in Arkansas.\\r\\nThey attend a Title I school,
which means there is a high enough percentage of free and
reduced-price lunch to qualify. Our school is an \\\"open
classroom\\\" concept, which is very unique as there are no
walls separating the classrooms. These 9 and 10 year-old
students are very eager learners; they are like sponges,
absorbing all the information and experiences and keep on
wanting more.With these resources such as the comfy red throw
pillows and the whimsical nautical hanging decor and the blue
fish nets, I will be able to help create the mood in our
classroom setting to be one of a themed nautical environment.
Creating a classroom environment is very important in the
success in each and every child's education. The nautical photo
props will be used with each child as they step foot into our
classroom for the first time on Meet the Teacher evening. I'll
take pictures of each child with them, have them developed, and
then hung in our classroom ready for their first day of 4th
grade. This kind gesture will set the tone before even the
first day of school! The nautical thank you cards will be used
throughout the year by the students as they create thank you
cards to their team groups.\\r\\n\\r\\nYour generous donations
will help me to help make our classroom a fun, inviting,
learning environment from day one.\\r\\n\\r\\nIt costs lost of
money out of my own pocket on resources to get our classroom

ready. Please consider helping with this project to make our new school year a very successful one. Thank you!nannan\n", "==\n", "My kindergarten students have varied disabilities ranging from speech and language delays, cognitive delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work their hardest working past their limitations. \\r\\n\\r\\nThe materials we have are the ones I seek out for my students. I teach in a Title I school where most of the students receive free or reduced price lunch. Despite their disabilities and limitations, my students love coming to school and come eager to learn and explore.Have you ever felt like you had ants in your pants and you needed to groove and move as you were in a meeting? This is how my kids feel all the time. The want to be able to move as they learn or so they say.Wobble chairs are the answer and I love then because they develop their core, which enhances gross motor and in Turn fine motor skills. \\r\\nThey also want to learn through games, my kids don't want to sit and do worksheets. They want to learn to count by jumping and playing. Physical engagement is the key to our success. The number toss and color and shape mats can make that happen. My students will forget they are doing work and just have the fun a 6 year old deserves.nannan\n", "==\n", "The mediocre teacher tells. The good teacher explains. The superior teacher demonstrates. The great teacher inspires. -William A. Ward\\r\\n\\r\\nMy school has 803 students which is makeup is 97.6% African-American, making up the largest segment of the student body. A typical school in Dallas is made up of 23.2% African-American students. Most of the students are on free or reduced lunch. We aren't receiving doctors, lawyers, or engineers children from rich backgrounds or neighborhoods. As an educator I am inspiring minds of young children and we focus not only on academics but one smart, effective, efficient, and disciplined students with good character.In our classroom we can utilize the Bluetooth for swift transitions during class. I use a speaker which doesn't amplify the sound enough to receive the message. Due to the volume of my speaker my students can't hear videos or books clearly and it isn't making the lessons as meaningful. But with the bluetooth speaker my students will be able to hear and I can stop, pause and replay it at any time.\\r\\nThe cart will allow me to have more room for storage of things that are needed for the day and has an extra part to it I can use. The table top chart has all of the letter, words and pictures for students to learn about different letters and it is more accessible.nannan\n", "==\n"] }], "source": ["# printing some random reviews\n", "print(project_data['essay'].values[0])\n", "print(\"=\"*50)\n",

"print(project_data['essay'].values[150])\n",
"print(\"=\"*50)\n",
"print(project_data['essay'].values[1000])\n",
"print(\"=\"*50)\n",
"print(project_data['essay'].values[20000])\n",
"print(\"=\"*50)\n",
"print(project_data['essay'].values[99999])\n",
"print(\"=\"*50)"] }, { "cell_type": "code",
"execution_count": 11, "metadata": { "colab": {}, "colab_type":
"code", "id": "IQnX07hHpdT6" }, "outputs": [], "source": ["#
https://stackoverflow.com/a/47091490/4084039\n", "import re\n",
"\n", "def decontracted(phrase):\n", " # specific\n", " phrase
= re.sub(r\"won't\", \"will not\", phrase)\n", " phrase =
re.sub(r\"can\\'t\", \"can not\", phrase)\n", "\n", " #
general\n", " phrase = re.sub(r\"n\\'t\", \" not\", phrase)\n",
" phrase = re.sub(r\"\\'re\", \" are\", phrase)\n", " phrase =
re.sub(r\"\\'s\", \" is\", phrase)\n", " phrase =
re.sub(r\"\\'d\", \" would\", phrase)\n", " phrase =
re.sub(r\"\\'ll\", \" will\", phrase)\n", " phrase =
re.sub(r\"\\'t\", \" not\", phrase)\n", " phrase =
re.sub(r\"\\'ve\", \" have\", phrase)\n", " phrase =
re.sub(r\"\\'m\", \" am\", phrase)\n", " return phrase"] }, {
"cell_type": "code", "execution_count": 12, "metadata": {
"colab": {}, "colab_type": "code", "id": "FsP1HBjNpdT8",
"outputId": "14b53bc7-f29e-47cc-f831-f06612cd12a4" },
"outputs": [{ "name": "stdout", "output_type": "stream",
"text": ["My kindergarten students have varied disabilities
ranging from speech and language delays, cognitive delays,
gross/fine motor delays, to autism. They are eager beavers and
always strive to work their hardest working past their
limitations. \\r\\n\\r\\nThe materials we have are the ones I
seek out for my students. I teach in a Title I school where
most of the students receive free or reduced price lunch.
Despite their disabilities and limitations, my students love
coming to school and come eager to learn and explore.Have you
ever felt like you had ants in your pants and you needed to
groove and move as you were in a meeting? This is how my kids
feel all the time. The want to be able to move as they learn or
so they say.Wobble chairs are the answer and I love then
because they develop their core, which enhances gross motor and
in Turn fine motor skills. \\r\\nThey also want to learn
through games, my kids do not want to sit and do worksheets.
They want to learn to count by jumping and playing. Physical
engagement is the key to our success. The number toss and color
and shape mats can make that happen. My students will forget
they are doing work and just have the fun a 6 year old
deserves.nannan\n",
"==\n"] }],
"source": ["sent =
decontracted(project_data['essay'].values[20000])\n",

"print(sent)\n", "print(\"=\"*50)"] }, { "cell_type": "code", "execution_count": 13, "metadata": { "colab": {}, "colab_type": "code", "id": "qYJiNOQmpdT_", "outputId": "28f217c0-2eae-4bc4-ea28-989cdd7c1223" }, "outputs": [{ "name": "stdout", "output_type": "stream", "text": ["My kindergarten students have varied disabilities ranging from speech and language delays, cognitive delays, gross/fine motor delays, to autism. They are eager beavers and always strive to work their hardest working past their limitations. The materials we have are the ones I seek out for my students. I teach in a Title I school where most of the students receive free or reduced price lunch. Despite their disabilities and limitations, my students love coming to school and come eager to learn and explore.Have you ever felt like you had ants in your pants and you needed to groove and move as you were in a meeting? This is how my kids feel all the time. The want to be able to move as they learn or so they say.Wobble chairs are the answer and I love then because they develop their core, which enhances gross motor and in Turn fine motor skills. They also want to learn through games, my kids do not want to sit and do worksheets. They want to learn to count by jumping and playing. Physical engagement is the key to our success. The number toss and color and shape mats can make that happen. My students will forget they are doing work and just have the fun a 6 year old deserves.nannan\n"] }], "source": ["# \\r \\n \\t remove from string python: http://texthandler.com/info/remove-line-breaks-python/\n", "sent = sent.replace('\\\\r', ' ')\n", "sent = sent.replace('\\\\\"', ' ')\n", "sent = sent.replace('\\\\n', ' ')\n", "print(sent)"] }, { "cell_type": "code", "execution_count": 14, "metadata": { "colab": {}, "colab_type": "code", "id": "y3glW4iYpdUC", "outputId": "e830f485-d2fe-4991-fcd4-db59f4fffc51" }, "outputs": [{ "name": "stdout", "output_type": "stream", "text": ["My kindergarten students have varied disabilities ranging from speech and language delays cognitive delays gross fine motor delays to autism They are eager beavers and always strive to work their hardest working past their limitations The materials we have are the ones I seek out for my students I teach in a Title I school where most of the students receive free or reduced price lunch Despite their disabilities and limitations my students love coming to school and come eager to learn and explore Have you ever felt like you had ants in your pants and you needed to groove and move as you were in a meeting This is how my kids feel all the time The want to be able to move as they learn or so they say Wobble chairs are the answer and I love then because they develop their core which enhances gross motor and in Turn fine motor skills They also want to learn through games my kids do not want to sit and do worksheets They want to learn to count by jumping and playing Physical engagement is the key to our success The number toss and color and shape mats can make that happen My students will forget they are doing work and just have the fun a 6 year old deserves nannan\n"] }], "source": ["#remove spacial character: https://stackoverflow.com/a/5843547/4084039\n", "sent = re.sub('[^A-Za-z0-9]+', ' ', sent)\n", "print(sent)"] }, { "cell_type": "code", "execution_count": 15, "metadata": { "colab": {}, "colab_type": "code", "id": "9HsssnlhpdUE" }, "outputs": [], "source": ["# https://gist.github.com/sebleier/554280\n", "# we are removing the words from the stop words list: 'no', 'nor', 'not'\n", "stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', \"you're\", \"you've\",\\\n", " \"you'll\", \"you'd\", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', \\\n", " 'she', \"she's\", 'her', 'hers', 'herself', 'it', \"it's\", 'its', 'itself', 'they', 'them', 'their',\\\n", " 'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', \"that'll\", 'these', 'those', \\\n", " 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', \\\n", " 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', \\\n", " 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',\\\n", " 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further',\\\n", " 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more',\\\n", " 'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too', 'very', \\\n", " 's', 't', 'can', 'will', 'just', 'don', \"don't\", 'should', \"should've\", 'now', 'd', 'll', 'm', 'o', 're', \\\n", " 've', 'y', 'ain', 'aren', \"aren't\", 'couldn', \"couldn't\", 'didn', \"didn't\", 'doesn', \"doesn't\", 'hadn',\\\n", " \"hadn't\", 'hasn', \"hasn't\", 'haven', \"haven't\", 'isn', \"isn't\", 'ma', 'mightn', \"mightn't\", 'mustn',\\\n", " \"mustn't\", 'needn', \"needn't\", 'shan', \"shan't\", 'shouldn', \"shouldn't\", 'wasn', \"wasn't\", 'weren', \"weren't\", \\\n", " 'won', \"won't\", 'wouldn', \"wouldn't\"]"] }, { "cell_type": "code", "execution_count": 16, "metadata": { "colab": {}, "colab_type": "code", "id": "MP4Cw6uPpdUG", "outputId": "a05d4b75-17ca-4da4-bacc-dbc93ec62aeb" }, "outputs": [{ "name": "stderr", "output_type": "stream", "text": ["100%|██████████| 109248/109248 [00:58<00:00, 1864.05it/s]\n"] }], "source": ["# Combining all the above stundents \n", "from tqdm import tqdm\n", "preprocessed_essays = []\n", "# tqdm is for printing the status bar\n", "for sentance in tqdm(project_data['essay'].values):\n", " sent = decontracted(sentance)\n", " sent = sent.replace('\\\\r', ' ')\n", " sent = sent.replace('\\\\\"', ' ')\n", " sent = sent.replace('\\\\n', ' ')\n", " sent = re.sub('[^A-Za-z0-9]+',

' ', sent)\n", " # https://gist.github.com/sebleier/554280\n",
" sent = ' '.join(e for e in sent.split() if e not in
stopwords)\n", "
preprocessed_essays.append(sent.lower().strip())"] }, {
"cell_type": "code", "execution_count": 17, "metadata": {
"colab": {}, "colab_type": "code", "id": "MnlZPliLpdUI",
"outputId": "f0c68dd7-56e4-4387-a0a4-f1cdad4bf150" },
"outputs": [{ "data": { "text/plain": ["'my kindergarten
students varied disabilities ranging speech language delays
cognitive delays gross fine motor delays autism they eager
beavers always strive work hardest working past limitations the
materials ones i seek students i teach title i school students
receive free reduced price lunch despite disabilities
limitations students love coming school come eager learn
explore have ever felt like ants pants needed groove move
meeting this kids feel time the want able move learn say wobble
chairs answer i love develop core enhances gross motor turn
fine motor skills they also want learn games kids not want sit
worksheets they want learn count jumping playing physical
engagement key success the number toss color shape mats make
happen my students forget work fun 6 year old deserves nannan'"
] }, "execution_count": 17, "metadata": {}, "output_type":
"execute_result" }], "source": ["# after preprocesing\n",
"preprocessed_essays[20000]"] }, { "cell_type": "code",
"execution_count": 18, "metadata": {}, "outputs": [], "source":
["#Project essay word count\n", "\n", "essay_word_count =
[]\n", "\n", "for ess in project_data[\"essay\"] :\n", " c =
len(ess.split())\n", " essay_word_count.append(c)\n", " \n",
"project_data[\"essay_word_count\"] = essay_word_count\n",
"\n", "project_data['preprocessed_essays'] =
preprocessed_essays"] }, { "cell_type": "code",
"execution_count": 19, "metadata": {}, "outputs": [], "source":
["import nltk\n", "#nltk.download()"] }, { "cell_type":
"code", "execution_count": 20, "metadata": {}, "outputs": [{
"name": "stderr", "output_type": "stream", "text": ["100%|
██████████| 109248/109248 [13:20<00:00, 136.52it/s]\n"] }],
"source": ["from nltk.sentiment.vader import
SentimentIntensityAnalyzer\n", "\n", "analyser =
SentimentIntensityAnalyzer()\n", "\n", "pos =[]\n", "neg =
[]\n", "neu = []\n", "compound = []\n", "\n", "for a in
tqdm(project_data[\"preprocessed_essays\"]) :\n", " b =
analyser.polarity_scores(a)['neg']\n", " c =
analyser.polarity_scores(a)['pos']\n", " d =
analyser.polarity_scores(a)['neu']\n", " e =
analyser.polarity_scores(a)['compound']\n", " neg.append(b)\n",
" pos.append(c)\n", " neu.append(d)\n", " compound.append(e)"]
}, { "cell_type": "code", "execution_count": 21, "metadata":
{}, "outputs": [], "source": ["project_data[\"pos\"] = pos\n",
"project_data[\"neg\"] = neg\n", "project_data[\"neu\"] =
neu\n", "project_data[\"compound\"] = compound"] }, {

"cell_type": "markdown", "metadata": { "colab_type": "text", "id": "NAViZD3bpdUK" }, "source": ["

1.4 Preprocessing of `project_title`

"] }, { "cell_type": "code", "execution_count": 22, "metadata": { "colab": {}, "colab_type": "code", "id": "Kzz1eTu9pdUL" }, "outputs": [{ "name": "stderr", "output_type": "stream", "text": ["100%|██████████| 109248/109248 [00:02<00:00, 45985.31it/s]\n"] }], "source": ["# similarly you can preprocess the titles also\n", "\n", "project_data.columns\n", "#sent1= decontracted(project_data['project_title'].values[20000])\n", "preprocessed_title = []\n", "# tqdm is for printing the status bar\n", "for sentance in tqdm(project_data['project_title'].values):\n", " sent1 = decontracted(sentance)\n", " sent1 = sent1.replace('\\\\r', ' ')\n", " sent1 = sent1.replace('\\\\\"', ' ')\n", " sent1 = sent1.replace('\\\\n', ' ')\n", " sent1 = re.sub('[^A-Za-z0-9]+', ' ', sent1)\n", " # https://gist.github.com/sebleier/554280\n", " sent1 = ' '.join(e for e in sent1.split() if e not in stopwords)\n", " preprocessed_title.append(sent.lower().strip())"] }, { "cell_type": "code", "execution_count": 23, "metadata": {}, "outputs": [], "source": ["#Project title word count\n", "title_word_count = []\n", "\n", "for a in project_data[\"project_title\"] :\n", " b = len(a.split())\n", " title_word_count.append(b)\n", "\n", "project_data[\"title_word_count\"] = title_word_count\n", "\n", "\n", "project_data['preprocessed_title'] = preprocessed_title"] }, { "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "yJd0B5QVpdUN" }, "source": ["## 1.5 Preparing data for models"] }, { "cell_type": "code", "execution_count": 24, "metadata": { "colab": {}, "colab_type": "code", "id": "ihafqEMWpdUN", "outputId": "7bd26423-47f1-440a-feaa-7608ba727d2e" }, "outputs": [{ "data": { "text/plain": ["Index(['Unnamed: 0', 'id', 'teacher_id', 'teacher_prefix', 'school_state',\n", " 'project_submitted_datetime', 'project_grade_category', 'project_title',\n", " 'project_essay_1', 'project_essay_2', 'project_essay_3',\n", " 'project_essay_4', 'project_resource_summary',\n", " 'teacher_number_of_previously_posted_projects', 'project_is_approved',\n", " 'clean_categories', 'clean_subcategories', 'essay', 'essay_word_count',\n", " 'preprocessed_essays', 'pos', 'neg', 'neu', 'compound',\n", " 'title_word_count', 'preprocessed_title'],\n", " dtype='object')"] }, "execution_count": 24, "metadata": {}, "output_type": "execute_result" }], "source": ["project_data.columns"] }, { "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "97JAGMlLpdUQ" }, "source": ["we are going to consider\n", "\n", " -

school_state : categorical data\n", " - clean_categories : categorical data\n", " - clean_subcategories : categorical data\n", " - project_grade_category : categorical data\n", " - teacher_prefix : categorical data\n", " \n", " - project_title : text data\n", " - text : text data\n", " - project_resource_summary: text data (optinal)\n", " \n", " - quantity : numerical (optinal)\n", " - teacher_number_of_previously_posted_projects : numerical\n", " - price : numerical"] }, { "cell_type": "code", "execution_count": 25, "metadata": {}, "outputs": [], "source": ["Y=project_data['project_is_approved']\n", "\n", "price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()\n", "project_data = pd.merge(project_data, price_data, on='id', how='left')"] }, { "cell_type": "code", "execution_count": 26, "metadata": {}, "outputs": [], "source": ["column_values=['clean_categories', 'clean_subcategories', 'school_state', 'project_grade_category', 'teacher_prefix','preprocessed_essays','preprocessed_title' ,'price','quantity','teacher_number_of_previously_posted_project
"\n", "def select_columns(dataframe, column_names):\n", " new_frame = dataframe.loc[:, column_names]\n", " return new_frame\n", "\n", "process_columns=select_columns(project_data,column_values)"] }, { "cell_type": "code", "execution_count": 27, "metadata": {}, "outputs": [{ "data": { "text/html": ["
\n", "

	clean_categories	clean_subcategories	school_state	project_grad
0	Literacy_Language	ESL Literacy	IN	Grades PreK
1	History_Civics Health_Sports	Civics_Government TeamSports	FL	Grades 6-8
2	Health_Sports	Health_Wellness TeamSports	AZ	Grades 6-8
3	Literacy_Language Math_Science	Literacy Mathematics	KY	Grades PreK
4	Math_Science	Mathematics	TX	Grades PreK

\n", "
"], "text/plain": [" clean_categories clean_subcategories school_state \\\n", "0 Literacy_Language ESL Literacy IN \n", "1 History_Civics Health_Sports Civics_Government TeamSports FL

\n", "2 Health_Sports Health_Wellness TeamSports AZ \n", "3 Literacy_Language Math_Science Literacy Mathematics KY \n", "4 Math_Science Mathematics TX \n", "\n", " project_grade_category teacher_prefix \\\n", "0 Grades PreK-2 Mrs. \n", "1 Grades 6-8 Mr. \n", "2 Grades 6-8 Ms. \n", "3 Grades PreK-2 Mrs. \n", "4 Grades PreK-2 Mrs. \n", "\n", " preprocessed_essays \\\n", "0 my students english learners working english s... \n", "1 our students arrive school eager learn they po... \n", "2 true champions not always ones win guts by mia... \n", "3 i work unique school filled esl english second... \n", "4 our second grade classroom next year made arou... \n", "\n", " preprocessed_title price quantity \\\n", "0 when last time used math probably within last ... 154.60 23 \n", "1 when last time used math probably within last ... 299.00 1 \n", "2 when last time used math probably within last ... 516.85 22 \n", "3 when last time used math probably within last ... 232.90 4 \n", "4 when last time used math probably within last ... 67.98 4 \n", "\n", " teacher_number_of_previously_posted_projects pos neg neu \\\n", "0 0 0.144 0.012 0.844 \n", "1 7 0.283 0.048 0.669 \n", "2 1 0.219 0.122 0.659 \n", "3 4 0.246 0.106 0.649 \n", "4 1 0.143 0.066 0.791 \n", "\n", " compound title_word_count essay_word_count project_is_approved \n", "0 0.9694 7 272 0 \n", "1 0.9856 5 221 1 \n", "2 0.9816 7 361 0 \n", "3 0.9656 2 213 1 \n", "4 0.8524 3 234 1 "] }, "execution_count": 27, "metadata": {}, "output_type": "execute_result" }], "source": ["process_columns.head()"] }, { "cell_type": "code", "execution_count": 28, "metadata": {}, "outputs": [{ "name": "stdout", "output_type": "stream", "text": ["(73196, 17) (73196,)\n", "(36052, 17) (36052,)\n", "==
] }], "source": ["# https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.trair
"from sklearn.model_selection import train_test_split\n", "\n", "# X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.33, shuffle=Flase)# this is for time series split\n", "X_train, X_test, y_train, y_test = train_test_split(process_columns, Y, test_size=0.33,random_state=42,stratify = Y) # this is random splitting\n", "\n", "\n", "print(X_train.shape, y_train.shape)\n", "print(X_test.shape, y_test.shape)\n", "\n", "print(\"=\"*100)"] }, { "cell_type": "code", "execution_count": 29, "metadata": {}, "outputs": [{ "name": "stdout", "output_type": "stream", "text": ["train columns Index(['clean_categories', 'clean_subcategories', 'school_state',\n", " 'project_grade_category', 'teacher_prefix', 'preprocessed_essays',\n", " 'preprocessed_title', 'price', 'quantity',\n", " 'teacher_number_of_previously_posted_projects', 'pos', 'neg', 'neu',\n", " 'compound', 'title_word_count', 'essay_word_count',\n", " 'project_is_approved'],\n", "

dtype='object')\n", "test columns Index(['clean_categories', 'clean_subcategories', 'school_state',\n", " 'project_grade_category', 'teacher_prefix', 'preprocessed_essays',\n", " 'preprocessed_title', 'price', 'quantity',\n", " 'teacher_number_of_previously_posted_projects', 'pos', 'neg', 'neu',\n", " 'compound', 'title_word_count', 'essay_word_count',\n", " 'project_is_approved'],\n", " dtype='object')\n"] }], "source": ["print(\"train columns\",X_train.columns)\n", "print(\"test columns\",X_test.columns)"] }, { "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "FACWWSZXpdUY" }, "source": ["### 1.5.2 Vectorizing Text data"] }, { "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "31MSXhlCpdUZ" }, "source": ["#### 1.5.2.1 Bag of words"] }, { "cell_type": "code", "execution_count": 30, "metadata": { "colab": {}, "colab_type": "code", "id": "sNFTyikxpdUZ", "outputId": "7a77b0d7-44a7-433d-90ee-f91af6401453" }, "outputs": [{ "name": "stdout", "output_type": "stream", "text": ["Shape of train matrix after one hot encodig (73196, 5000)\n", "Shape of test matrix after one hot encodig (36052, 5000)\n"] }], "source": ["# We are considering only the words which appeared in at least 10 documents(rows or projects).\n", "from sklearn.feature_extraction.text import CountVectorizer\n", "\n", "vectorizer_bow_essay = CountVectorizer(min_df=10, ngram_range = (1,2),max_features=5000)\n", "vectorizer_bow_essay.fit(X_train['preprocessed_essays'])\n", "\n", "text_bow_train= vectorizer_bow_essay.transform(X_train['preprocessed_essays'])\n "text_bow_test= vectorizer_bow_essay.transform(X_test['preprocessed_essays'])\n" "\n", "\n", "print(\"Shape of train matrix after one hot encodig \",text_bow_train.shape)\n", "print(\"Shape of test matrix after one hot encodig \",text_bow_test.shape)"] }, { "cell_type": "code", "execution_count": 31, "metadata": { "colab": {}, "colab_type": "code", "id": "sNFTyikxpdUZ", "outputId": "7a77b0d7-44a7-433d-90ee-f91af6401453" }, "outputs": [{ "name": "stdout", "output_type": "stream", "text": ["Shape of train matrix after one hot encodig title_bow (73196, 132)\n", "Shape of test matrix after one hot encodig title_bow (36052, 132)\n"] }], "source": ["# before you vectorize the title make sure you preprocess it\n", "from sklearn.feature_extraction.text import CountVectorizer\n", "\n", "vectorizer_bow_title = CountVectorizer(min_df=10)\n", "vectorizer_bow_title.fit(X_train['preprocessed_title'])\n", "\n", "title_bow_train = vectorizer_bow_title.transform(X_train['preprocessed_title'])\n" "title_bow_test = vectorizer_bow_title.transform(X_test['preprocessed_title'])\n",

"\n", "\n", "print(\"Shape of train matrix after one hot encodig title_bow\",title_bow_train.shape)\n", "print(\"Shape of test matrix after one hot encodig title_bow\",title_bow_test.shape)"] }, { "cell_type": "markdown", "metadata": { "colab": {}, "colab_type": "code", "id": "sNFTyikxpdUZ", "outputId": "7a77b0d7-44a7-433d-90ee-f91af6401453" }, "source": ["#### 1.5.2.2 TFIDF vectorizer"] }, { "cell_type": "code", "execution_count": 32, "metadata": { "colab": {}, "colab_type": "code", "id": "sNFTyikxpdUZ", "outputId": "7a77b0d7-44a7-433d-90ee-f91af6401453" }, "outputs": [{ "name": "stdout", "output_type": "stream", "text": ["Shape of train matrix after one hot encodig (73196, 5000)\n", "Shape of test matrix after one hot encodig (36052, 5000)\n"] }], "source": ["from sklearn.feature_extraction.text import TfidfVectorizer\n", "\n", "vectorizer_tfidf_essay= TfidfVectorizer(min_df=10,ngram_range = (1,2),max_features=5000)\n", "vectorizer_tfidf_essay.fit(X_train['preprocessed_essays'])\n", "\n", "text_tfidf_train= vectorizer_tfidf_essay.transform(X_train['preprocessed_essays'])
"text_tfidf_test= vectorizer_tfidf_essay.transform(X_test['preprocessed_essays'])\
"\n", "print(\"Shape of train matrix after one hot encodig \",text_tfidf_train.shape) \n", "print(\"Shape of test matrix after one hot encodig \",text_tfidf_test.shape)"] }, { "cell_type": "code", "execution_count": 33, "metadata": { "colab": {}, "colab_type": "code", "id": "sNFTyikxpdUZ", "outputId": "7a77b0d7-44a7-433d-90ee-f91af6401453" }, "outputs": [{ "name": "stdout", "output_type": "stream", "text": ["Shape of train matrix after one hot encodig (73196, 132)\n", "Shape of test matrix after one hot encodig (36052, 132)\n"] }], "source": ["# Similarly you can vectorize for title also\n", "\n", "from sklearn.feature_extraction.text import TfidfVectorizer\n", "\n", "vectorizer_tfidf_title = TfidfVectorizer(min_df=10)\n", "vectorizer_tfidf_title.fit(X_train['preprocessed_title'])\n", "\n", "title_tfidf_train = vectorizer_tfidf_title.transform(X_train['preprocessed_title'])\
"title_tfidf_test = vectorizer_tfidf_title.transform(X_test['preprocessed_title'])\r
"\n", "print(\"Shape of train matrix after one hot encodig \",title_tfidf_train.shape)\n", "print(\"Shape of test matrix after one hot encodig \",title_tfidf_test.shape)"] }, { "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "d0MRbILdpdUe" }, "source": ["#### 1.5.2.3 Using Pretrained Models: Avg W2V"] }, { "cell_type": "code", "execution_count": 34, "metadata": {}, "outputs": [], "source": ["from gensim.models import Word2Vec\n", "from gensim.models import KeyedVectors"] }, { "cell_type": "code",

"execution_count": 35, "metadata": {}, "outputs": [], "source":
["i=0\n", "list_of_sentance_train=[]\n", "for sentance in
X_train['preprocessed_essays']:\n", "
list_of_sentance_train.append(sentance.split())"] }, {
"cell_type": "code", "execution_count": 36, "metadata": {},
"outputs": [], "source": ["# this line of code trains your w2v
model on the give list of sentances\n",
"w2v_model=Word2Vec(list_of_sentance_train,min_count=25,size=50,
workers=32)"] }, { "cell_type": "code", "execution_count": 37,
"metadata": {}, "outputs": [{ "name": "stdout", "output_type":
"stream", "text": ["number of words that occured minimum 25
times 10205\n", "sample words ['all', 'students', 'least',
'year', 'behind', 'as', 'interventionist', 'school', 'i',
'close', 'achievement', 'gap', 'given', 'no', 'budget', 'buy',
'necessary', 'supplies', 'books', 'rti', 'leader', 'analyze',
'wide', 'student', 'data', 'indicates', 'struggling', 'math',
'understanding', 'number', 'sense', 'place', 'value', 'the',
'lowest', 'performing', 'sub', 'group', '60', 'readers',
'work', 'daily', 'basis', 'due', 'excited', 'next', 'provide',
'reading', 'and', 'intervention']\n"] }], "source": [
"w2v_words = list(w2v_model.wv.vocab)\n", "print(\"number of
words that occured minimum 25 times \",len(w2v_words))\n",
"print(\"sample words \", w2v_words[0:50])"] }, { "cell_type":
"code", "execution_count": 38, "metadata": {}, "outputs": [{
"name": "stderr", "output_type": "stream", "text": ["100%|
██████████| 73196/73196 [03:47<00:00, 321.98it/s]\n"] }, {
"name": "stdout", "output_type": "stream", "text": ["(73196,
50)\n", "[-0.20969025 -0.17036675 0.18304946 0.30050331
-0.90464496 -0.20548479\n", " 0.59605581 -0.57968779 1.20987622
0.65106853 -1.07405388 -0.14562401\n", " -0.07715248
-0.15373315 -0.54806496 -0.02373823 0.31004367 0.15334937\n", "
-0.76353239 -0.36864351 -0.04180904 -0.04664254 -0.45299031
0.41015909\n", " -0.56603439 0.69138712 -0.21798502 0.51443151
0.45593692 0.09271958\n", " 0.15253069 -0.04223945 0.00752853
-0.40325668 -0.46356833 0.57186428\n", " 0.04291661 -0.17928035
-0.34632963 -0.31110657 0.26182621 -0.15202924\n", "
-0.09256956 -0.60987374 0.23740318 0.30221608 0.13734471
-0.2828066\n", " -0.28291729 0.53367092]\n"] }], "source": [
"# average Word2Vec of essays \n", "# compute average word2vec
for each review.\n", "essay_vectors_train = []; # the avg-w2v
for each sentence/review is stored in this list\n", "for sent
in tqdm(list_of_sentance_train): # for each review/sentence\n",
" sent_vec = np.zeros(50) # as word vectors are of zero length
50, you might need to change this to 300 if you use google's
w2v\n", " cnt_words =0; # num of words with a valid vector in
the sentence/review\n", " for word in sent: # for each word in
a review/sentence\n", " if word in w2v_words:\n", " vec =
w2v_model.wv[word]\n", " sent_vec += vec\n", " cnt_words +=
1\n", " if cnt_words != 0:\n", " sent_vec /= cnt_words\n", "
essay_vectors_train.append(sent_vec)\n", "essay_vectors_train =

np.array(essay_vectors_train)\n",
"print(essay_vectors_train.shape)\n",
"print(essay_vectors_train[0])"] }, { "cell_type": "code",
"execution_count": 39, "metadata": {}, "outputs": [], "source":
["i=0\n", "list_of_sentance_test=[]\n", "for sentance in
X_test['preprocessed_essays']:\n", "
list_of_sentance_test.append(sentance.split())"] }, {
"cell_type": "code", "execution_count": 40, "metadata": {},
"outputs": [{ "name": "stderr", "output_type": "stream",
"text": ["100%|██████████| 36052/36052 [01:55<00:00,
312.72it/s]\n"] }, { "name": "stdout", "output_type":
"stream", "text": ["(36052, 50)\n", "[-1.03234189e-01
-6.35210380e-01 2.74948929e-01 -4.65499115e-01\n", "
-6.10591979e-01 -2.11303504e-01 5.42927572e-01 -5.94091248e-
01\n", " 1.81108788e+00 8.65892551e-01 -1.20850002e+00
4.95633176e-01\n", " -1.15563638e-01 -1.06693171e+00
-1.04405001e+00 2.74153395e-01\n", " -2.24650460e-01
1.14813298e-01 -7.22415451e-01 -3.07578754e-01\n", "
2.10898896e-01 -4.81398088e-01 -2.71826735e-01 4.77185513e-
01\n", " -1.85589388e-01 5.44004956e-01 -7.01724069e-01
8.84563500e-02\n", " -1.08500102e-01 1.35094144e-01
1.47851465e-01 2.04794185e-01\n", " -1.38356088e-01
-2.30276257e-01 -8.01001296e-02 5.39432944e-01\n", "
-7.67724017e-01 4.08281422e-01 -1.92833994e-01 -1.26087673e-
01\n", " 2.63593540e-01 -7.18116557e-02 -1.79755956e-03
-8.12065057e-01\n", " 3.90696931e-02 5.44397560e-01
4.56496848e-01 -3.60808174e-01\n", " -5.84212653e-01
1.84677317e-01]\n"] }], "source": ["# average Word2Vec\n",
"# compute average word2vec for each review.\n",
"essay_vectors_test = []; # the avg-w2v for each
sentence/review is stored in this list\n", "for sent in
tqdm(list_of_sentance_test): # for each review/sentence\n", "
sent_vec = np.zeros(50) # as word vectors are of zero length
50, you might need to change this to 300 if you use google's
w2v\n", " cnt_words =0; # num of words with a valid vector in
the sentence/review\n", " for word in sent: # for each word in
a review/sentence\n", " if word in w2v_words:\n", " vec =
w2v_model.wv[word]\n", " sent_vec += vec\n", " cnt_words +=
1\n", " if cnt_words != 0:\n", " sent_vec /= cnt_words\n", "
essay_vectors_test.append(sent_vec)\n", "essay_vectors_test =
np.array(essay_vectors_test)\n",
"print(essay_vectors_test.shape)\n",
"print(essay_vectors_test[0])"] }, { "cell_type": "code",
"execution_count": 41, "metadata": {}, "outputs": [], "source":
["#similarly doing it for preprocessed title\n", "i=0\n",
"list_of_sentance_train=[]\n", "for sentance in
X_train['preprocessed_title']:\n", "
list_of_sentance_train.append(sentance.split())"] }, {
"cell_type": "code", "execution_count": 42, "metadata": {},
"outputs": [], "source": ["# this line of code trains your w2v

model on the give list of sentances\n",
"w2v_model=Word2Vec(list_of_sentance_train,min_count=5,size=50,
workers=16)"] }, { "cell_type": "code", "execution_count": 43,
"metadata": {}, "outputs": [{ "name": "stdout", "output_type":
"stream", "text": ["number of words that occured minimum 5
times 133\n", "sample words ['when', 'last', 'time', 'used',
'math', 'probably', 'within', 'hour', 'yet', 'go', 'school',
'believing', 'never', 'use', 'my', 'students', 'engage',
'authentic', 'experiences', 'routinely', 'help', 'understand',
'critical', 'truly', 'i', 'teach', 'small', 'town', 'big',
'dreams', 'fantastic', 'opportunities', 'surround', 'ultimate',
'goal', 'achieve', 'success', 'seeking', 'drive', 'potential',
'take', 'world', 'storm', 'graduation', 'all', 'need',
'little', 'according', 'forbes', 'magazine']\n"] }],
"source": ["w2v_words = list(w2v_model.wv.vocab)\n",
"print(\"number of words that occured minimum 5 times
\",len(w2v_words))\n", "print(\"sample words \",
w2v_words[0:50])"] }, { "cell_type": "code",
"execution_count": 44, "metadata": {}, "outputs": [{ "name":
"stderr", "output_type": "stream", "text": ["100%|██████████|
73196/73196 [00:48<00:00, 1517.48it/s]\n"] }, { "name":
"stdout", "output_type": "stream", "text": ["(73196, 50)\n", "
[-0.08766698 0.03098032 -0.17588627 -0.16100769 0.12978251
0.35685455\n", " -0.19642146 0.22584495 -0.39482379 0.18408063
-0.1603591 -0.2101878\n", " -0.15226938 -0.30864978 -0.08631066
0.22628395 0.10543454 -0.14973936\n", " -0.2318005 -0.10813686
-0.19975651 0.11067983 0.19342739 0.08796791\n", " 0.01189079
-0.10595054 0.2002089 0.1779408 -0.28190624 0.12459674\n", "
0.24873456 0.25143815 -0.02553013 -0.01477906 -0.06000585
-0.08835318\n", " 0.06294654 0.14935194 0.1616493 -0.1234625
0.05740457 0.05319793\n", " 0.0898553 -0.47352764 0.24496199
0.19794599 0.0081433 -0.038764\n", " -0.09199591
-0.22871841]\n"] }], "source": ["# compute average word2vec
for each review.\n", "title_vectors_train = []; # the avg-w2v
for each sentence/review is stored in this list\n", "for sent
in tqdm(list_of_sentance_train): # for each review/sentence\n",
" sent_vec = np.zeros(50) # as word vectors are of zero length
50, you might need to change this to 300 if you use google's
w2v\n", " cnt_words =0; # num of words with a valid vector in
the sentence/review\n", " for word in sent: # for each word in
a review/sentence\n", " if word in w2v_words:\n", " vec =
w2v_model.wv[word]\n", " sent_vec += vec\n", " cnt_words +=
1\n", " if cnt_words != 0:\n", " sent_vec /= cnt_words\n", "
title_vectors_train.append(sent_vec)\n", "title_vectors_train =
np.array(title_vectors_train)\n",
"print(title_vectors_train.shape)\n",
"print(title_vectors_train[0])"] }, { "cell_type": "code",
"execution_count": 45, "metadata": {}, "outputs": [], "source":
["i=0\n", "list_of_sentance_test=[]\n", "for sentance in
X_test['preprocessed_title']:\n", "

list_of_sentance_test.append(sentance.split())"] }, { "cell_type": "code", "execution_count": 46, "metadata": {}, "outputs": [{ "name": "stderr", "output_type": "stream", "text": ["100%|██████████| 36052/36052 [00:24<00:00, 1490.02it/s]\n"] }, { "name": "stdout", "output_type": "stream", "text": ["(36052, 50)\n", "[-0.08766698 0.03098032 -0.17588627 -0.16100769 0.12978251 0.35685455\n", " -0.19642146 0.22584495 -0.39482379 0.18408063 -0.1603591 -0.2101878\n", " -0.15226938 -0.30864978 -0.08631066 0.22628395 0.10543454 -0.14973936\n", " -0.2318005 -0.10813686 -0.19975651 0.11067983 0.19342739 0.08796791\n", " 0.01189079 -0.10595054 0.2002089 0.1779408 -0.28190624 0.12459674\n", " 0.24873456 0.25143815 -0.02553013 -0.01477906 -0.06000585 -0.08835318\n", " 0.06294654 0.14935194 0.1616493 -0.1234625 0.05740457 0.05319793\n", " 0.0898553 -0.47352764 0.24496199 0.19794599 0.0081433 -0.038764\n", " -0.09199591 -0.22871841]\n"] }], "source": ["# compute average word2vec for each review.\n", "title_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list\n", "for sent in tqdm(list_of_sentance_test): # for each review/sentence\n", " sent_vec = np.zeros(50) # as word vectors are of zero length 50, you might need to change this to 300 if you use google's w2v\n", " cnt_words =0; # num of words with a valid vector in the sentence/review\n", " for word in sent: # for each word in a review/sentence\n", " if word in w2v_words:\n", " vec = w2v_model.wv[word]\n", " sent_vec += vec\n", " cnt_words += 1\n", " if cnt_words != 0:\n", " sent_vec /= cnt_words\n", " title_vectors_test.append(sent_vec)\n", "title_vectors_test = np.array(title_vectors_test)\n", "print(title_vectors_test.shape)\n", "print(title_vectors_test[0])"] }, { "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "yHkIu6gbpdUj" }, "source": ["#### 1.5.2.3 Using Pretrained Models: TFIDF weighted W2V"] }, { "cell_type": "code", "execution_count": 47, "metadata": { "colab": {}, "colab_type": "code", "id": "DMYMgvztpdUj" }, "outputs": [], "source": ["# S = [\"abc def pqr\", \"def def def abc\", \"pqr pqr def\"]\n", "tfidf_model = TfidfVectorizer()\n", "tfidf_model.fit(X_train['preprocessed_essays'])\n", "# we are converting a dictionary with word as a key, and the idf as a value\n", "dictionary = dict(zip(tfidf_model.get_feature_names(), list(tfidf_model.idf_)))\n", "tfidf_words = set(tfidf_model.get_feature_names())"] }, { "cell_type": "code", "execution_count": 48, "metadata": {}, "outputs": [], "source": ["# storing variables into pickle files python: http://www.jessicayung.com/how-to-use-pickle-to-save-and-load-variables-in-python/\n", "# make sure you have the glove_vectors file\n", "with open('C:/Users/pramod reddy chandi/Desktop/pram/applied ai course/DonorsChoose_2018/glove_vectors', 'rb') as f:\n", " model = pickle.load(f)\n", " glove_words = set(model.keys())"] }, { "cell_type": "code", "execution_count": 49, "metadata": { "colab": {}, "colab_type": "code", "id": "5H21HVjnpdUk", "outputId": "5d2cd664-eab0-44d9-c71a-aa30074587e3", "scrolled": true }, "outputs": [{ "name": "stderr", "output_type": "stream", "text": ["100%|██████████| 73196/73196 [02:30<00:00, 486.73it/s]\n"] }, { "name": "stdout", "output_type": "stream", "text": ["73196\n", "300\n"] }], "source": ["tfidf_w2v_vectors_train = []; # the avg-w2v for each sentence/review is stored in this list\n", "for sentence in tqdm(X_train['preprocessed_essays']): # for each review/sentence\n", " vector = np.zeros(300) # as word vectors are of zero length\n", " tf_idf_weight =0; # num of words with a valid vector in the sentence/review\n", " for word in sentence.split(): # for each word in a review/sentence\n", " if (word in glove_words) and (word in tfidf_words):\n", " vec = model[word] # getting the vector for each word\n", " # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))\n", " tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word\n", " vector += (vec * tf_idf) # calculating tfidf weighted w2v\n", " tf_idf_weight += tf_idf\n", " if tf_idf_weight != 0:\n", " vector /= tf_idf_weight\n", " tfidf_w2v_vectors_train.append(vector)\n", "\n", "print(len(tfidf_w2v_vectors_train))\n", "print(len(tfidf_w2v_vectors_train[0]))"] }, { "cell_type": "code", "execution_count": 50, "metadata": {}, "outputs": [{ "name": "stderr", "output_type": "stream", "text": ["100%|██████████| 36052/36052 [01:14<00:00, 485.06it/s]\n"] }, { "name": "stdout", "output_type": "stream", "text": ["36052\n", "300\n"] }], "source": ["# compute average word2vec for each review.\n", "tfidf_w2v_vectors_test = []; # the avg-w2v for each sentence/review is stored in this list\n", "for sentence in tqdm(X_test['preprocessed_essays']): # for each review/sentence\n", " vector = np.zeros(300) # as word vectors are of zero length\n", " tf_idf_weight =0; # num of words with a valid vector in the sentence/review\n", " for word in sentence.split(): # for each word in a review/sentence\n", " if (word in glove_words) and (word in tfidf_words):\n", " vec = model[word] # getting the vector for each word\n", " # here we are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))\n", " tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word\n", " vector += (vec * tf_idf) # calculating tfidf weighted w2v\n", " tf_idf_weight += tf_idf\n", " if tf_idf_weight != 0:\n", " vector /= tf_idf_weight\n", " tfidf_w2v_vectors_test.append(vector)\n",

"\n", "print(len(tfidf_w2v_vectors_test))\n",
"print(len(tfidf_w2v_vectors_test[0]))"] }, { "cell_type":
"code", "execution_count": 51, "metadata": {}, "outputs": [],
"source": ["# Similarly you can vectorize for title also\n",
"tfidf_model = TfidfVectorizer()\n",
"tfidf_model.fit(X_train['preprocessed_title'])\n", "# we are
converting a dictionary with word as a key, and the idf as a
value\n", "dictionary =
dict(zip(tfidf_model.get_feature_names(),
list(tfidf_model.idf_)))\n", "tfidf_words =
set(tfidf_model.get_feature_names())"] }, { "cell_type":
"code", "execution_count": 52, "metadata": {}, "outputs": [{
"name": "stderr", "output_type": "stream", "text": ["100%|
██████████| 73196/73196 [02:27<00:00, 496.86it/s]\n"] }, {
"name": "stdout", "output_type": "stream", "text": ["73196\n",
"300\n"] }], "source": ["# average Word2Vec\n", "# compute
average word2vec for each review.\n", "tfidf_w2v_title_train =
[]; # the avg-w2v for each sentence/review is stored in this
list\n", "for sentence in tqdm(X_train['preprocessed_title']):
for each review/sentence\n", " vector = np.zeros(300) # as
word vectors are of zero length\n", " tf_idf_weight =0; # num
of words with a valid vector in the sentence/review\n", " for
word in sentence.split(): # for each word in a
review/sentence\n", " if (word in glove_words) and (word in
tfidf_words):\n", " vec = model[word] # getting the vector for
each word\n", " # here we are multiplying idf
value(dictionary[word]) and the tf
value((sentence.count(word)/len(sentence.split())))\n", "
tf_idf = dictionary[word]*
(sentence.count(word)/len(sentence.split())) # getting the
tfidf value for each word\n", " vector += (vec * tf_idf) #
calculating tfidf weighted w2v\n", " tf_idf_weight +=
tf_idf\n", " if tf_idf_weight != 0:\n", " vector /=
tf_idf_weight\n", " tfidf_w2v_title_train.append(vector)\n",
"\n", "print(len(tfidf_w2v_title_train))\n",
"print(len(tfidf_w2v_title_train[0]))"] }, { "cell_type":
"code", "execution_count": 53, "metadata": {}, "outputs": [{
"name": "stderr", "output_type": "stream", "text": ["100%|
██████████| 36052/36052 [01:13<00:00, 492.10it/s]\n"] }, {
"name": "stdout", "output_type": "stream", "text": ["36052\n",
"300\n"] }], "source": ["# average Word2Vec\n", "# compute
average word2vec for each review.\n", "tfidf_w2v_title_test =
[]; # the avg-w2v for each sentence/review is stored in this
list\n", "for sentence in tqdm(X_test['preprocessed_title']): #
for each review/sentence\n", " vector = np.zeros(300) # as word
vectors are of zero length\n", " tf_idf_weight =0; # num of
words with a valid vector in the sentence/review\n", " for word
in sentence.split(): # for each word in a review/sentence\n", "
if (word in glove_words) and (word in tfidf_words):\n", " vec =
model[word] # getting the vector for each word\n", " # here we

are multiplying idf value(dictionary[word]) and the tf value((sentence.count(word)/len(sentence.split())))\n", " tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # getting the tfidf value for each word\n", " vector += (vec * tf_idf) # calculating tfidf weighted w2v\n", " tf_idf_weight += tf_idf\n", " if tf_idf_weight != 0:\n", " vector /= tf_idf_weight\n", " tfidf_w2v_title_test.append(vector)\n", "\n", "print(len(tfidf_w2v_title_test))\n", "print(len(tfidf_w2v_title_test[0]))"] }, { "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "e5-6e7fFpdUn" }, "source": ["### 1.5.3 Vectorizing Numerical features"] }, { "cell_type": "code", "execution_count": 54, "metadata": { "colab": {}, "colab_type": "code", "id": "TPlY_FtypdUn" }, "outputs": [], "source": ["price_data = resource_data.groupby('id').agg({'price':'sum', 'quantity':'sum'}).reset_index()\n", "project_data = pd.merge(project_data, price_data, on='id', how='left')"] }, { "cell_type": "code", "execution_count": 55, "metadata": { "colab": {}, "colab_type": "code", "id": "2M8pbtZwpdUo" }, "outputs": [{ "name": "stdout", "output_type": "stream", "text": ["After vectorizations\n", "(73196, 1) (73196,)\n", "(36052, 1) (36052,)\n"] }], "source": ["#scaling of price feature\n", "\n", "# check this one: https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s\n", "# standardization sklearn: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Standar
"from sklearn.preprocessing import Normalizer\n", "\n", "# price_standardized = standardScalar.fit(project_data['price'].values)\n", "# this will rise the error\n", "# ValueError: Expected 2D array, got 1D array instead: array=[725.05 213.03 329. ... 399. 287.73 5.5].\n", "# Reshape your data either using array.reshape(-1, 1)\n", "\n", "price_scalar = Normalizer()\n", "price_scalar.fit(X_train['price'].values.reshape(-1,1)) # finding the mean and standard deviation of this data\n", "\n", "\n", "# Now standardize the data with above maen and variance.\n", "price_standardized_train= price_scalar.transform(X_train['price'].values.reshape(-1, 1))\n", "price_standardized_test= price_scalar.transform(X_test['price'].values.reshape(-1, 1))\n", "\n", "print(\"After vectorizations\")\n", "print(price_standardized_train.shape, y_train.shape)\n", "print(price_standardized_test.shape, y_test.shape)"] }, { "cell_type": "code", "execution_count": 56, "metadata": { "colab": {}, "colab_type": "code", "id": "2M8pbtZwpdUo" }, "outputs": [{ "name": "stdout", "output_type": "stream", "text": ["After vectorizations\n", "(73196, 1) (73196,)\n", "(36052, 1) (36052,)\n"] }], "source": ["#scaling of qunatity feature\n", "\n", "# check this one:

https://www.youtube.com/watch?v=0HOqOcln3Z4&t=530s\n", "#
standardization sklearn: https://scikit-
learn.org/stable/modules/generated/sklearn.preprocessing.Standar
"from sklearn.preprocessing import Normalizer\n", "\n", "#
price_standardized =
standardScalar.fit(project_data['price'].values)\n", "# this
will rise the error\n", "# ValueError: Expected 2D array, got
1D array instead: array=[725.05 213.03 329. ... 399. 287.73 5.5
].\n", "# Reshape your data either using array.reshape(-1,
1)\n", "\n", "quantity_scalar = Normalizer()\n",
"quantity_scalar.fit(X_train['quantity'].values.reshape(-1,1))
finding the mean and standard deviation of this data\n",
"\n", "\n", "# Now standardize the data with above maen and
variance.\n", "quantity_standardized_train=
quantity_scalar.transform(X_train['quantity'].values.reshape(-1,
1))\n", "quantity_standardized_test=
quantity_scalar.transform(X_test['quantity'].values.reshape(-1,
1))\n", "\n", "print(\"After vectorizations\")\n",
"print(quantity_standardized_train.shape, y_train.shape)\n",
"print(quantity_standardized_test.shape, y_test.shape)"] }, {
"cell_type": "code", "execution_count": 57, "metadata": {
"colab": {}, "colab_type": "code", "id": "2M8pbtZwpdUo" },
"outputs": [{ "name": "stdout", "output_type": "stream",
"text": ["After vectorizations\n", "(73196, 1) (73196,)\n", "
(36052, 1) (36052,)\n"] }], "source": ["#scaling of teachers
number of previously posted projects \n", "\n", "from
sklearn.preprocessing import Normalizer\n", "\n",
"normalizer_projects_num = Normalizer()\n", "\n", "#
normalizer.fit(X_train['price'].values)\n", "# this will rise
an error Expected 2D array, got 1D array instead: \n", "#
array=[105.22 215.96 96.01 ... 368.98 80.53 709.67].\n", "#
Reshape your data either using \n", "# array.reshape(-1, 1) if
your data has a single feature \n", "# array.reshape(1, -1) if
it contains a single sample.\n", "\n",
"normalizer_projects_num.fit(X_train['teacher_number_of_previous
"\n", "prev_projects_train =
normalizer_projects_num.transform(X_train['teacher_number_of_pre
"prev_projects_test =
normalizer_projects_num.transform(X_test['teacher_number_of_prev
"\n", "print(\"After vectorizations\")\n",
"print(prev_projects_train.shape, y_train.shape)\n",
"print(prev_projects_test.shape, y_test.shape)"] }, {
"cell_type": "code", "execution_count": 58, "metadata": {
"colab": {}, "colab_type": "code", "id": "2M8pbtZwpdUo" },
"outputs": [{ "name": "stdout", "output_type": "stream",
"text": ["After vectorizations\n", "(73196, 1) (73196,)\n", "
(36052, 1) (36052,)\n",
"==
] }], "source": ["# normalixing the title word count\n",
"\n", "from sklearn.preprocessing import Normalizer\n", "\n",

"normalizer_title_word = Normalizer()\n", "\n",
"normalizer_title_word.fit(X_train['title_word_count'].values.re
"\n", "title_word_count_train =
normalizer_title_word.transform(X_train['title_word_count'].valu
"title_word_count_test =
normalizer_title_word.transform(X_test['title_word_count'].value
"\n", "print(\"After vectorizations\")\n",
"print(title_word_count_train.shape, y_train.shape)\n",
"print(title_word_count_test.shape, y_test.shape)\n",
"print(\"=\"*100)"] }, { "cell_type": "code",
"execution_count": 59, "metadata": { "colab": {}, "colab_type":
"code", "id": "2M8pbtZwpdUo" }, "outputs": [{ "name":
"stdout", "output_type": "stream", "text": ["After
vectorizations\n", "(73196, 1) (73196,)\n", "(36052, 1)
(36052,)\n"] }], "source": ["# normalixing the essay word
count\n", "\n", "from sklearn.preprocessing import
Normalizer\n", "\n", "normalizer_ess_count = Normalizer()\n",
"\n",
"normalizer_ess_count.fit(X_train['essay_word_count'].values.res
"\n", "essay_word_count_train =
normalizer_ess_count.transform(X_train['essay_word_count'].value
"essay_word_count_test =
normalizer_ess_count.transform(X_test['essay_word_count'].values
"\n", "print(\"After vectorizations\")\n",
"print(essay_word_count_train.shape, y_train.shape)\n",
"print(essay_word_count_test.shape, y_test.shape)"] }, {
"cell_type": "code", "execution_count": 60, "metadata": {
"colab": {}, "colab_type": "code", "id": "2M8pbtZwpdUo" },
"outputs": [{ "name": "stdout", "output_type": "stream",
"text": ["After vectorizations\n", "(73196, 1) (73196,)\n", "
(36052, 1) (36052,)\n"] }], "source": ["#normalizing the
data for essay sentiment-pos\n", "from sklearn.preprocessing
import Normalizer\n", "normalizer_pos = Normalizer()\n", "\n",
"normalizer_pos.fit(X_train['pos'].values.reshape(-1,1))\n",
"\n", "essay_sent_pos_train =
normalizer_pos.transform(X_train['pos'].values.reshape(-1,1))\n'
"essay_sent_pos_test =
normalizer_pos.transform(X_test['pos'].values.reshape(-1,1))\n",
"\n", "print(\"After vectorizations\")\n",
"print(essay_sent_pos_train.shape, y_train.shape)\n",
"print(essay_sent_pos_test.shape, y_test.shape)"] }, {
"cell_type": "code", "execution_count": 61, "metadata": {
"colab": {}, "colab_type": "code", "id": "2M8pbtZwpdUo" },
"outputs": [{ "name": "stdout", "output_type": "stream",
"text": ["After vectorizations\n", "(73196, 1) (73196,)\n", "
(36052, 1) (36052,)\n"] }], "source": ["#normalizing the
data for essay sentiment-neg\n", "from sklearn.preprocessing
import Normalizer\n", "\n", "normalizer_neg= Normalizer()\n",
"\n",
"normalizer_neg.fit(X_train['neg'].values.reshape(-1,1))\n",

"\n", "essay_sent_neg_train =
normalizer_neg.transform(X_train['neg'].values.reshape(-1,1))\n'
"essay_sent_neg_test =
normalizer_neg.transform(X_test['neg'].values.reshape(-1,1))\n",
"\n", "print(\"After vectorizations\")\n",
"print(essay_sent_neg_train.shape, y_train.shape)\n",
"print(essay_sent_neg_test.shape, y_test.shape)"] }, {
"cell_type": "code", "execution_count": 62, "metadata": {
"colab": {}, "colab_type": "code", "id": "2M8pbtZwpdUo" },
"outputs": [{ "name": "stdout", "output_type": "stream",
"text": ["After vectorizations\n", "(73196, 1) (73196,)\n", "
(36052, 1) (36052,)\n"] }], "source": ["#normalizing the
data for essay sentiment-neu\n", "from sklearn.preprocessing
import Normalizer\n", "\n", "normalizer_nue= Normalizer()\n",
"\n",
"normalizer_nue.fit(X_train['neu'].values.reshape(-1,1))\n",
"\n", "essay_sent_nue_train =
normalizer_nue.transform(X_train['neu'].values.reshape(-1,1))\n'
"essay_sent_nue_test =
normalizer_nue.transform(X_test['neu'].values.reshape(-1,1))\n",
"\n", "print(\"After vectorizations\")\n",
"print(essay_sent_nue_train.shape, y_train.shape)\n",
"print(essay_sent_nue_test.shape, y_test.shape)"] }, {
"cell_type": "code", "execution_count": 63, "metadata": {
"colab": {}, "colab_type": "code", "id": "2M8pbtZwpdUo" },
"outputs": [{ "name": "stdout", "output_type": "stream",
"text": ["After vectorizations\n", "(73196, 1) (73196,)\n", "
(36052, 1) (36052,)\n",
"==
] }], "source": ["#normalizing the data for essay sentiment-
compound\n", "from sklearn.preprocessing import Normalizer\n",
"\n", "normalizer_compound= Normalizer()\n", "\n",
"normalizer_compound.fit(X_train['compound'].values.reshape(-1,1
"\n", "essay_sent_comp_train =
normalizer_compound.transform(X_train['compound'].values.reshape
"essay_sent_comp_test =
normalizer_compound.transform(X_test['compound'].values.reshape(
"\n", "print(\"After vectorizations\")\n",
"print(essay_sent_comp_train.shape, y_train.shape)\n",
"print(essay_sent_comp_test.shape, y_test.shape)\n",
"print(\"=\"*100)"] }, { "cell_type": "code",
"execution_count": 64, "metadata": { "colab": {}, "colab_type":
"code", "id": "2UdzJV_epdUp", "outputId": "1ac2d13d-4f77-4a2f-
fa8e-c995b2d4d53b" }, "outputs": [], "source": ["# response
encoding for categocal features\n", "def response(var):\n", "
\n", " X_train_pos = X_train.loc[X_train['project_is_approved']
== 1]\n", " \n", " var_state_pos = {}\n", " \n", " for a in
X_train_pos[var] :\n", " if a not in var_state_pos :\n", "
var_state_pos[a] = 1\n", " else :\n", " var_state_pos[a] +=
1\n", " \n", " X_train_neg =

X_train.loc[X_train['project_is_approved'] == 0]\n", " \n", "
var_state_neg = {}\n", "\n", " for a in X_train_neg[var] :\n",
" if a not in var_state_neg :\n", " var_state_neg[a] = 1\n", "
else :\n", " var_state_neg[a] += 1\n", " \n", "
var_state_neg[np.nan] =0 \n", " \n", " var_state_total = {}\n",
"\n", " for a in X_train[var] :\n", " if a not in
var_state_total :\n", " var_state_total[a] = 1\n", " else :\n",
" var_state_total[a] += 1\n", " \n", " \n", " pos_prob_state =
{}\n", "\n", " for state in var_state_total.keys():\n", "
pos_prob_state[state] =
(var_state_pos[state])/float(var_state_total[state])\n", " \n",
" neg_prob_state = {}\n", "\n", " for state in
var_state_total.keys():\n", " neg_prob_state[state] =
(var_state_neg[state])/float(var_state_total[state])\n", " \n",
" state_0_train = []\n", " state_1_train = []\n", "\n", " for a
in X_train[var] :\n", "
state_0_train.append(neg_prob_state[a])\n", "
state_1_train.append(pos_prob_state[a]) \n", " \n", "
state_0_test = []\n", " state_1_test = []\n", "\n", " for a in
X_test[var] :\n", " state_0_test.append(neg_prob_state[a])\n",
" state_1_test.append(pos_prob_state[a])\n", " return
state_0_train,state_1_train,state_0_test,state_1_test"] }, {
"cell_type": "code", "execution_count": 65, "metadata": {},
"outputs": [{ "name": "stdout", "output_type": "stream",
"text": ["train columns Index(['clean_categories',
'clean_subcategories', 'school_state',\n", "
'project_grade_category', 'teacher_prefix',
'preprocessed_essays',\n", " 'preprocessed_title', 'price',
'quantity',\n", "
'teacher_number_of_previously_posted_projects', 'pos', 'neg',
'neu',\n", " 'compound', 'title_word_count',
'essay_word_count',\n", " 'project_is_approved'],\n", "
dtype='object')\n", "test columns Index(['clean_categories',
'clean_subcategories', 'school_state',\n", "
'project_grade_category', 'teacher_prefix',
'preprocessed_essays',\n", " 'preprocessed_title', 'price',
'quantity',\n", "
'teacher_number_of_previously_posted_projects', 'pos', 'neg',
'neu',\n", " 'compound', 'title_word_count',
'essay_word_count',\n", " 'project_is_approved'],\n", "
dtype='object')\n"] }], "source": ["print(\"train
columns\",X_train.columns)\n", "print(\"test
columns\",X_test.columns)"] }, { "cell_type": "code",
"execution_count": 66, "metadata": {}, "outputs": [], "source":
["# response encoding for school_state\n", "\n",
"X_train[\"state_0\"],X_train[\"state_1\"],X_test[\"state_0\"],X
] }, { "cell_type": "code", "execution_count": 67, "metadata":
{}, "outputs": [{ "name": "stdout", "output_type": "stream",
"text": ["After vectorizations\n", "(73196, 1) (73196,)\n", "
(36052, 1) (36052,)\n",

"===
] }], "source": ["#normalize for 0\n", "from
sklearn.preprocessing import Normalizer\n", "\n", "normalizer =
Normalizer()\n", "\n", "#
normalizer.fit(X_train['price'].values)\n", "# this will rise
an error Expected 2D array, got 1D array instead: \n", "#
array=[105.22 215.96 96.01 ... 368.98 80.53 709.67].\n", "#
Reshape your data either using \n", "# array.reshape(-1, 1) if
your data has a single feature \n", "# array.reshape(1, -1) if
it contains a single sample.\n", "\n",
"normalizer.fit(X_train[\"state_0\"].values.reshape(-1,1))\n",
"\n", "state_0_train =
normalizer.transform(X_train[\"state_0\"].values.reshape(-1,1))\
"state_0_test =
normalizer.transform(X_test[\"state_0\"].values.reshape(-1,1))\r
"\n", "print(\"After vectorizations\")\n",
"print(state_0_train.shape, y_train.shape)\n",
"print(state_0_test.shape, y_test.shape)\n", "print(\"=\"*100)"
] }, { "cell_type": "code", "execution_count": 68, "metadata":
{}, "outputs": [{ "name": "stdout", "output_type": "stream",
"text": ["After vectorizations\n", "(73196, 1) (73196,)\n", "
(36052, 1) (36052,)\n",
"===
] }], "source": ["#normalize for 1\n", "from
sklearn.preprocessing import Normalizer\n", "\n", "normalizer =
Normalizer()\n", "\n", "#
normalizer.fit(X_train['price'].values)\n", "# this will rise
an error Expected 2D array, got 1D array instead: \n", "#
array=[105.22 215.96 96.01 ... 368.98 80.53 709.67].\n", "#
Reshape your data either using \n", "# array.reshape(-1, 1) if
your data has a single feature \n", "# array.reshape(1, -1) if
it contains a single sample.\n", "\n",
"normalizer.fit(X_train[\"state_1\"].values.reshape(-1,1))\n",
"\n", "state_1_train =
normalizer.transform(X_train[\"state_1\"].values.reshape(-1,1))\
"state_1_test =
normalizer.transform(X_test[\"state_1\"].values.reshape(-1,1))\r
"\n", "print(\"After vectorizations\")\n",
"print(state_1_train.shape, y_train.shape)\n",
"print(state_1_test.shape, y_test.shape)\n", "print(\"=\"*100)"
] }, { "cell_type": "code", "execution_count": 69, "metadata":
{}, "outputs": [], "source": ["# response encoding for project
grade category\n", "X_train[\"proj_grade_0\"],
X_train[\"proj_grade_1\"], X_test[\"proj_grade_0\"],
X_test[\"proj_grade_1\"]= response(\"project_grade_category\")"
] }, { "cell_type": "code", "execution_count": 70, "metadata":
{}, "outputs": [{ "name": "stdout", "output_type": "stream",
"text": ["After vectorizations\n", "(73196, 1) (73196,)\n", "
(36052, 1) (36052,)\n",
"===

] }], "source": ["#normalize for 0\n", "from
sklearn.preprocessing import Normalizer\n", "\n", "normalizer =
Normalizer()\n", "\n",
"normalizer.fit(X_train[\"proj_grade_0\"].values.reshape(-1,1))\
"\n", "proj_grade_0_train =
normalizer.transform(X_train[\"proj_grade_0\"].values.reshape(-1
"proj_grade_0_test =
normalizer.transform(X_test[\"proj_grade_0\"].values.reshape(-1,
"\n", "print(\"After vectorizations\")\n",
"print(proj_grade_0_train.shape, y_train.shape)\n",
"print(proj_grade_0_test.shape, y_test.shape)\n",
"print(\"=\"*100)"] }, { "cell_type": "code",
"execution_count": 71, "metadata": {}, "outputs": [{ "name":
"stdout", "output_type": "stream", "text": ["After
vectorizations\n", "(73196, 1) (73196,)\n", "(36052, 1)
(36052,)\n",
"==
] }], "source": ["#normalize for 1\n", "\n", "from
sklearn.preprocessing import Normalizer\n", "\n", "normalizer =
Normalizer()\n", "\n",
"normalizer.fit(X_train[\"proj_grade_1\"].values.reshape(-1,1))\
"\n", "proj_grade_1_train =
normalizer.transform(X_train[\"proj_grade_1\"].values.reshape(-1
"proj_grade_1_test =
normalizer.transform(X_test[\"proj_grade_1\"].values.reshape(-1,
"\n", "print(\"After vectorizations\")\n",
"print(proj_grade_1_train.shape, y_train.shape)\n",
"print(proj_grade_1_test.shape, y_test.shape)\n",
"print(\"=\"*100)"] }, { "cell_type": "code",
"execution_count": 72, "metadata": {}, "outputs": [], "source":
["\n", "X_train[\"teacher_prefix_0\"],
X_train[\"teacher_prefix_1\"], X_test[\"teacher_prefix_0\"],
X_test[\"teacher_prefix_1\"]= response(\"teacher_prefix\")"]
}, { "cell_type": "code", "execution_count": 73, "metadata":
{}, "outputs": [{ "name": "stdout", "output_type": "stream",
"text": ["After vectorizations\n", "(73196, 1) (73196,)\n", "
(36052, 1) (36052,)\n",
"==
] }], "source": ["#normalize for 0\n", "from
sklearn.preprocessing import Normalizer\n", "\n", "normalizer =
Normalizer()\n", "\n",
"normalizer.fit(X_train[\"teacher_prefix_0\"].values.reshape(-1,
"\n", "teacher_prefix_0_train =
normalizer.transform(X_train[\"teacher_prefix_0\"].values.reshap
"teacher_prefix_0_test =
normalizer.transform(X_test[\"teacher_prefix_0\"].values.reshape
"\n", "print(\"After vectorizations\")\n",
"print(teacher_prefix_0_train.shape, y_train.shape)\n",
"print(teacher_prefix_0_test.shape, y_test.shape)\n",
"print(\"=\"*100)"] }, { "cell_type": "code",

"execution_count": 74, "metadata": {}, "outputs": [{ "name": "stdout", "output_type": "stream", "text": ["After vectorizations\n", "(73196, 1) (73196,)\n", "(36052, 1) (36052,)\n", "==
] }], "source": ["#normalize for 1\n", "\n", "from sklearn.preprocessing import Normalizer\n", "\n", "normalizer = Normalizer()\n", "\n", "normalizer.fit(X_train[\"teacher_prefix_1\"].values.reshape(-1,
"\n", "teacher_prefix_1_train = normalizer.transform(X_train[\"teacher_prefix_1\"].values.reshap
"teacher_prefix_1_test = normalizer.transform(X_test[\"teacher_prefix_1\"].values.reshape
"\n", "print(\"After vectorizations\")\n", "print(teacher_prefix_1_train.shape, y_train.shape)\n", "print(teacher_prefix_1_test.shape, y_test.shape)\n", "print(\"=\"*100)"] }, { "cell_type": "code", "execution_count": 75, "metadata": {}, "outputs": [], "source": ["#response encoding for clean categories\n", "\n", "clean_pos = {}\n", "\n", "X_train_pos = X_train.loc[X_train['project_is_approved'] == 1]\n", "\n", "for a in X_train_pos['clean_categories'] :\n", " for b in a.split():\n", " if b not in clean_pos :\n", " clean_pos[b] = 1\n", " else :\n", " clean_pos[b] += 1"] }, { "cell_type": "code", "execution_count": 76, "metadata": {}, "outputs": [], "source": ["clean_neg = {}\n", "\n", "X_train_neg = X_train.loc[X_train['project_is_approved'] == 0]\n", " \n", "for a in X_train_neg['clean_categories'] :\n", " for b in a.split():\n", " if b not in clean_neg :\n", " clean_neg[b] = 1\n", " else :\n", " clean_neg[b] += 1"] }, { "cell_type": "code", "execution_count": 77, "metadata": {}, "outputs": [], "source": ["clean_total = {}\n", "\n", "for a in X_train['clean_categories'] :\n", " for b in a.split():\n", " if b not in clean_total :\n", " clean_total[b] = 1\n", " else :\n", " clean_total[b] += 1"] }, { "cell_type": "code", "execution_count": 78, "metadata": {}, "outputs": [], "source": ["pos_prob_category = {}\n", "\n", "for st in clean_total.keys():\n", " pos_prob_category[st] = (clean_pos[st])/float(clean_total[st])"] }, { "cell_type": "code", "execution_count": 79, "metadata": {}, "outputs": [], "source": ["neg_prob_category = {}\n", "\n", "for stt in clean_total.keys():\n", " neg_prob_category[stt] = (clean_neg[stt])/float(clean_total[stt])"] }, { "cell_type": "code", "execution_count": 80, "metadata": {}, "outputs": [], "source": ["cat_0_train = []\n", "cat_1_train = []\n", "\n", "for a in X_train[\"clean_categories\"] :\n", " b = a.split()\n", " if len(b) == 1 :\n", " cat_0_train.append(neg_prob_category[a])\n", " cat_1_train.append(pos_prob_category[a])\n", " else :\n", " c = neg_prob_category[b[0]]\n", " d = neg_prob_category[b[1]]\n", "

e = pos_prob_category[b[0]]\n", " f =
pos_prob_category[b[1]]\n", " \n", "
cat_0_train.append(c*d)\n", " cat_1_train.append(e*f)"] }, {
"cell_type": "code", "execution_count": 81, "metadata": {},
"outputs": [], "source": ["cat_0_test = []\n", "cat_1_test =
[]\n", "\n", "for a in X_test[\"clean_categories\"] :\n", " b =
a.split()\n", " if len(b) == 1 :\n", "
cat_0_test.append(neg_prob_category[a])\n", "
cat_1_test.append(pos_prob_category[a])\n", " else :\n", " c =
neg_prob_category[b[0]]\n", " d = neg_prob_category[b[1]]\n", "
e = pos_prob_category[b[0]]\n", " f =
pos_prob_category[b[1]]\n", " \n", " cat_0_test.append(c*d)\n",
" cat_1_test.append(e*f)"] }, { "cell_type": "code",
"execution_count": 82, "metadata": {}, "outputs": [], "source":
["X_train[\"cat_0\"] = cat_0_train"] }, { "cell_type":
"code", "execution_count": 83, "metadata": {}, "outputs": [],
"source": ["X_train[\"cat_1\"] = cat_1_train"] }, {
"cell_type": "code", "execution_count": 84, "metadata": {},
"outputs": [], "source": ["X_test[\"cat_0\"] = cat_0_test"]
}, { "cell_type": "code", "execution_count": 85, "metadata":
{}, "outputs": [], "source": ["X_test[\"cat_1\"] = cat_1_test"
] }, { "cell_type": "code", "execution_count": 86, "metadata":
{}, "outputs": [{ "name": "stdout", "output_type": "stream",
"text": ["After vectorizations\n", "(73196, 1) (73196,)\n", "
(36052, 1) (36052,)\n",
"===
] }], "source": ["# normalise for set 0\n", "from
sklearn.preprocessing import Normalizer\n", "\n", "normalizer =
Normalizer()\n", "\n", "#
normalizer.fit(X_train['price'].values)\n", "# this will rise
an error Expected 2D array, got 1D array instead: \n", "#
array=[105.22 215.96 96.01 ... 368.98 80.53 709.67].\n", "#
Reshape your data either using \n", "# array.reshape(-1, 1) if
your data has a single feature \n", "# array.reshape(1, -1) if
it contains a single sample.\n", "\n",
"normalizer.fit(X_train[\"cat_0\"].values.reshape(-1,1))\n",
"\n", "cat_0_train =
normalizer.transform(X_train[\"cat_0\"].values.reshape(-1,1))\n'
"cat_0_test =
normalizer.transform(X_test[\"cat_0\"].values.reshape(-1,1))\n",
"\n", "print(\"After vectorizations\")\n",
"print(cat_0_train.shape, y_train.shape)\n",
"print(cat_0_test.shape, y_test.shape)\n", "print(\"=\"*100)"]
}, { "cell_type": "code", "execution_count": 87, "metadata":
{}, "outputs": [{ "name": "stdout", "output_type": "stream",
"text": ["After vectorizations\n", "(73196, 1) (73196,)\n", "
(36052, 1) (36052,)\n",
"===
] }], "source": ["# normalize for set1\n", "from
sklearn.preprocessing import Normalizer\n", "\n", "normalizer =

Normalizer()\n", "\n", "#
normalizer.fit(X_train['price'].values)\n", "# this will rise
an error Expected 2D array, got 1D array instead: \n", "#
array=[105.22 215.96 96.01 ... 368.98 80.53 709.67].\n", "#
Reshape your data either using \n", "# array.reshape(-1, 1) if
your data has a single feature \n", "# array.reshape(1, -1) if
it contains a single sample.\n", "\n",
"normalizer.fit(X_train[\"cat_1\"].values.reshape(-1,1))\n",
"\n", "cat_1_train =
normalizer.transform(X_train[\"cat_1\"].values.reshape(-1,1))\n'
"cat_1_test =
normalizer.transform(X_test[\"cat_1\"].values.reshape(-1,1))\n",
"\n", "print(\"After vectorizations\")\n",
"print(cat_1_train.shape, y_train.shape)\n",
"print(cat_1_test.shape, y_test.shape)\n", "print(\"=\"*100)"]
}, { "cell_type": "markdown", "metadata": {}, "source": [
"#response encoding for clean_subcategories"] }, {
"cell_type": "code", "execution_count": 88, "metadata": {},
"outputs": [], "source": ["clean_pos = {}\n", "\n",
"X_train_pos = X_train.loc[X_train['project_is_approved'] ==
1]\n", "\n", "for a in X_train_pos['clean_subcategories'] :\n",
" for b in a.split():\n", " if b not in clean_pos :\n", "
clean_pos[b] = 1\n", " else :\n", " clean_pos[b] += 1"] }, {
"cell_type": "code", "execution_count": 89, "metadata": {},
"outputs": [], "source": ["clean_neg = {}\n", "\n",
"X_train_neg = X_train.loc[X_train['project_is_approved'] ==
0]\n", " \n", "for a in X_train_neg['clean_subcategories']
:\n", " for b in a.split():\n", " if b not in clean_neg :\n", "
clean_neg[b] = 1\n", " else :\n", " clean_neg[b] += 1"] }, {
"cell_type": "code", "execution_count": 90, "metadata": {},
"outputs": [], "source": ["clean_total = {}\n", "\n", "for a
in X_train['clean_subcategories'] :\n", " for b in
a.split():\n", " if b not in clean_total :\n", " clean_total[b]
= 1\n", " else :\n", " clean_total[b] += 1"] }, { "cell_type":
"code", "execution_count": 91, "metadata": {}, "outputs": [],
"source": ["pos_prob_category = {}\n", "\n", "for st in
clean_total.keys():\n", " pos_prob_category[st] =
(clean_pos[st])/float(clean_total[st])"] }, { "cell_type":
"code", "execution_count": 92, "metadata": {}, "outputs": [],
"source": ["neg_prob_category = {}\n", "\n", "for stt in
clean_total.keys():\n", " neg_prob_category[stt] =
(clean_neg[stt])/float(clean_total[stt])"] }, { "cell_type":
"code", "execution_count": 93, "metadata": {}, "outputs": [],
"source": ["subcat_0_train = []\n", "subcat_1_train = []\n",
"\n", "for a in X_train[\"clean_subcategories\"] :\n", " b =
a.split()\n", " if len(b) == 1 :\n", "
subcat_0_train.append(neg_prob_category[a])\n", "
subcat_1_train.append(pos_prob_category[a])\n", " else :\n", "
c = neg_prob_category[b[0]]\n", " d =
neg_prob_category[b[1]]\n", " e = pos_prob_category[b[0]]\n", "

f = pos_prob_category[b[1]]\n", " \n", "
subcat_0_train.append(c*d)\n", " subcat_1_train.append(e*f)"]
}, { "cell_type": "code", "execution_count": 94, "metadata":
{}, "outputs": [], "source": ["subcat_0_test = []\n",
"subcat_1_test = []\n", "\n", "for a in
X_test[\"clean_subcategories\"] :\n", " b = a.split()\n", " if
len(b) == 1 :\n", "
subcat_0_test.append(neg_prob_category[a])\n", "
subcat_1_test.append(pos_prob_category[a])\n", " else :\n", " c
= neg_prob_category[b[0]]\n", " d = neg_prob_category[b[1]]\n",
" e = pos_prob_category[b[0]]\n", " f =
pos_prob_category[b[1]]\n", " \n", "
subcat_0_test.append(c*d)\n", " subcat_1_test.append(e*f)"] },
{ "cell_type": "code", "execution_count": 95, "metadata": {},
"outputs": [], "source": ["X_train[\"subcat_0\"] =
subcat_0_train\n", "\n", "X_train[\"subcat_1\"] =
subcat_1_train"] }, { "cell_type": "code", "execution_count":
96, "metadata": {}, "outputs": [], "source": [
"X_test[\"subcat_0\"] = subcat_0_test\n", "\n",
"X_test[\"subcat_1\"] = subcat_1_test"] }, { "cell_type":
"code", "execution_count": 97, "metadata": {}, "outputs": [{
"name": "stdout", "output_type": "stream", "text": ["After
vectorizations\n", "(73196, 1) (73196,)\n", "(36052, 1)
(36052,)\n",
"==
] }], "source": ["# normalise for set 0\n", "from
sklearn.preprocessing import Normalizer\n", "\n", "normalizer =
Normalizer()\n", "\n", "#
normalizer.fit(X_train['price'].values)\n", "# this will rise
an error Expected 2D array, got 1D array instead: \n", "#
array=[105.22 215.96 96.01 ... 368.98 80.53 709.67].\n", "#
Reshape your data either using \n", "# array.reshape(-1, 1) if
your data has a single feature \n", "# array.reshape(1, -1) if
it contains a single sample.\n", "\n",
"normalizer.fit(X_train[\"subcat_0\"].values.reshape(-1,1))\n",
"\n", "subcat_0_train =
normalizer.transform(X_train[\"subcat_0\"].values.reshape(-1,1))
"subcat_0_test =
normalizer.transform(X_test[\"subcat_0\"].values.reshape(-1,1))\
"\n", "print(\"After vectorizations\")\n",
"print(subcat_0_train.shape, y_train.shape)\n",
"print(subcat_0_test.shape, y_test.shape)\n",
"print(\"=\"*100)"] }, { "cell_type": "code",
"execution_count": 98, "metadata": {}, "outputs": [{ "name":
"stdout", "output_type": "stream", "text": ["After
vectorizations\n", "(73196, 1) (73196,)\n", "(36052, 1)
(36052,)\n",
"==
] }], "source": ["# normalize for set1\n", "from
sklearn.preprocessing import Normalizer\n", "\n", "normalizer =

Normalizer()\n", "\n", "#
normalizer.fit(X_train['price'].values)\n", "# this will rise
an error Expected 2D array, got 1D array instead: \n", "#
array=[105.22 215.96 96.01 ... 368.98 80.53 709.67].\n", "#
Reshape your data either using \n", "# array.reshape(-1, 1) if
your data has a single feature \n", "# array.reshape(1, -1) if
it contains a single sample.\n", "\n",
"normalizer.fit(X_train[\"subcat_1\"].values.reshape(-1,1))\n",
"\n", "subcat_1_train =
normalizer.transform(X_train[\"subcat_1\"].values.reshape(-1,1))
"subcat_1_test =
normalizer.transform(X_test[\"subcat_1\"].values.reshape(-1,1))\
"\n", "print(\"After vectorizations\")\n",
"print(subcat_1_train.shape, y_train.shape)\n",
"print(subcat_1_test.shape, y_test.shape)\n",
"print(\"=\"*100)"] }, { "cell_type": "markdown", "metadata":
{ "colab_type": "text", "id": "mKUm7hY4pdUr" }, "source": [
"### 1.5.4 Merging all the above features"] }, { "cell_type":
"code", "execution_count": 99, "metadata": {}, "outputs": [],
"source": ["# merge two sparse matrices:
https://stackoverflow.com/a/19710648/4084039\n", "#set1\n",
"from scipy.sparse import hstack\n",
"set1_train=hstack((price_standardized_train,
quantity_standardized_train, prev_projects_train,
title_word_count_train, essay_word_count_train,
essay_sent_pos_train,essay_sent_neg_train,
essay_sent_nue_train, essay_sent_comp_train, state_0_train,
state_1_train, proj_grade_0_train, proj_grade_1_train,
teacher_prefix_0_train, teacher_prefix_1_train, cat_0_train,
cat_1_train, subcat_0_train, subcat_1_train,text_bow_train,
title_bow_train\n", "))\n", "\n",
"set1_test=hstack((price_standardized_test,
quantity_standardized_test, prev_projects_test,
title_word_count_test, essay_word_count_test,
essay_sent_pos_test,essay_sent_neg_test, essay_sent_nue_test,
essay_sent_comp_test, state_0_test, state_1_test,
proj_grade_0_test, proj_grade_1_test, teacher_prefix_0_test,
teacher_prefix_1_test, cat_0_test, cat_1_test, subcat_0_test,
subcat_1_test,text_bow_test, title_bow_test\n", "))"] }, {
"cell_type": "code", "execution_count": 100, "metadata": {},
"outputs": [], "source": ["#set2\n",
"set2_train=hstack((price_standardized_train,
quantity_standardized_train, prev_projects_train,
title_word_count_train, essay_word_count_train,
essay_sent_pos_train,essay_sent_neg_train,
essay_sent_nue_train, essay_sent_comp_train, state_0_train,
state_1_train, proj_grade_0_train, proj_grade_1_train,
teacher_prefix_0_train, teacher_prefix_1_train, cat_0_train,
cat_1_train, subcat_0_train, subcat_1_train,text_tfidf_train,
title_tfidf_train\n", "))\n",

"set2_test=hstack((price_standardized_test,
quantity_standardized_test, prev_projects_test,
title_word_count_test, essay_word_count_test,
essay_sent_pos_test,essay_sent_neg_test, essay_sent_nue_test,
essay_sent_comp_test, state_0_test, state_1_test,
proj_grade_0_test, proj_grade_1_test, teacher_prefix_0_test,
teacher_prefix_1_test, cat_0_test, cat_1_test, subcat_0_test,
subcat_1_test,text_tfidf_test, title_tfidf_test\n", "))"] }, {
"cell_type": "code", "execution_count": 101, "metadata": {},
"outputs": [], "source": ["#set3\n", "# conversion of w2v
essay and title to 2d array and applying npstack\n",
"essay_vectors_train_2d=np.array(essay_vectors_train)\n",
"title_vectors_train_2d=np.array(title_vectors_train)\n", "\n",
"essay_vectors_test_2d=np.array(essay_vectors_test)\n",
"title_vectors_test_2d=np.array(title_vectors_test)"] }, {
"cell_type": "code", "execution_count": 102, "metadata": {},
"outputs": [], "source": [
"set3_train=np.hstack((price_standardized_train,
quantity_standardized_train, prev_projects_train,
title_word_count_train, essay_word_count_train,
essay_sent_pos_train,essay_sent_neg_train,
essay_sent_nue_train, essay_sent_comp_train, state_0_train,
state_1_train, proj_grade_0_train, proj_grade_1_train,
teacher_prefix_0_train, teacher_prefix_1_train, cat_0_train,
cat_1_train, subcat_0_train, subcat_1_train,
essay_vectors_train_2d, title_vectors_train_2d\n", "))\n",
"set3_test=np.hstack((price_standardized_test,
quantity_standardized_test, prev_projects_test,
title_word_count_test, essay_word_count_test,
essay_sent_pos_test,essay_sent_neg_test, essay_sent_nue_test,
essay_sent_comp_test, state_0_test, state_1_test,
proj_grade_0_test, proj_grade_1_test, teacher_prefix_0_test,
teacher_prefix_1_test, cat_0_test, cat_1_test, subcat_0_test,
subcat_1_test, essay_vectors_test_2d, title_vectors_test_2d\n",
"))"] }, { "cell_type": "code", "execution_count": 103,
"metadata": {}, "outputs": [], "source": ["#set4\n", "#
conversion of w2v essay and title to 2d array and applying
npstack\n",
"tfidf_w2v_vectors_train_2d=np.array(tfidf_w2v_vectors_train)\n'
"tfidf_w2v_title_train_2d=np.array(tfidf_w2v_title_train)\n",
"tfidf_w2v_vectors_test_2d=np.array(tfidf_w2v_vectors_test)\n",
"tfidf_w2v_title_test_2d=np.array(tfidf_w2v_title_test)"] }, {
"cell_type": "code", "execution_count": 104, "metadata": {},
"outputs": [], "source": [
"set4_train=np.hstack((price_standardized_train,
quantity_standardized_train, prev_projects_train,
title_word_count_train, essay_word_count_train,
essay_sent_pos_train,essay_sent_neg_train,
essay_sent_nue_train, essay_sent_comp_train, state_0_train,
state_1_train, proj_grade_0_train, proj_grade_1_train,

teacher_prefix_0_train, teacher_prefix_1_train, cat_0_train, cat_1_train, subcat_0_train, subcat_1_train, tfidf_w2v_vectors_train_2d, tfidf_w2v_title_train_2d))\n", "set4_test=np.hstack((price_standardized_test, quantity_standardized_test, prev_projects_test, title_word_count_test, essay_word_count_test, essay_sent_pos_test,essay_sent_neg_test, essay_sent_nue_test, essay_sent_comp_test, state_0_test, state_1_test, proj_grade_0_test, proj_grade_1_test, teacher_prefix_0_test, teacher_prefix_1_test, cat_0_test, cat_1_test, subcat_0_test, subcat_1_test, tfidf_w2v_vectors_test_2d, tfidf_w2v_title_test_2d))"] }, { "cell_type": "code", "execution_count": 130, "metadata": {}, "outputs": [{ "data": { "text/plain": ["scipy.sparse.coo.coo_matrix"] }, "execution_count": 130, "metadata": {}, "output_type": "execute_result" }], "source": ["type(set1_train)"] }, { "cell_type": "code", "execution_count": 109, "metadata": {}, "outputs": [], "source": ["import pickle\n", "f=open('9set1.pckl','wb')\n", "pickle.dump([set1_train, set1_test],f)\n", "f.close()"] }, { "cell_type": "code", "execution_count": 110, "metadata": {}, "outputs": [], "source": ["import pickle\n", "f=open('9set2.pckl','wb')\n", "pickle.dump([set2_train, set2_test],f)\n", "f.close()"] }, { "cell_type": "code", "execution_count": 111, "metadata": {}, "outputs": [], "source": ["import pickle\n", "f=open('9set3.pckl','wb')\n", "pickle.dump([set3_train, set3_test],f)\n", "f.close()"] }, { "cell_type": "code", "execution_count": 112, "metadata": {}, "outputs": [], "source": ["import pickle\n", "f=open('9set4.pckl','wb')\n", "pickle.dump([set4_train, set4_test],f)\n", "f.close()"] }, { "cell_type": "code", "execution_count": 113, "metadata": {}, "outputs": [], "source": ["import pickle\n", "f=open('9y_values.pckl','wb')\n", "pickle.dump([y_train,y_test],f)\n", "f.close()"] }, { "cell_type": "code", "execution_count": 1, "metadata": {}, "outputs": [], "source": ["import pickle as pickle\n", "#with open('C:/Users/pramod reddy chandi/Desktop/pram/applied ai course/DonorsChoose_2018/cat_num.pckl', 'rb') as f:\n", "f=open('C:/Users/pramod reddy chandi/Desktop/pram/applied ai course/DonorsChoose_2018/9set1.pckl','rb')\n", "set1_train, set1_test=pickle.load(f)\n", "f.close()"] }, { "cell_type": "code", "execution_count": 2, "metadata": {}, "outputs": [], "source": ["import pickle as pickle\n", "#with open('C:/Users/pramod reddy chandi/Desktop/pram/applied ai course/DonorsChoose_2018/cat_num.pckl', 'rb') as f:\n", "f=open('C:/Users/pramod reddy chandi/Desktop/pram/applied ai course/DonorsChoose_2018/9set2.pckl','rb')\n", "set2_train, set2_test=pickle.load(f)\n", "f.close()"] }, { "cell_type": "code", "execution_count": 3, "metadata": {}, "outputs": [], "source": ["import pickle as pickle\n", "#with open('C:/Users/pramod reddy chandi/Desktop/pram/applied ai course/DonorsChoose_2018/cat_num.pckl', 'rb') as f:\n", "f=open('C:/Users/pramod reddy chandi/Desktop/pram/applied ai course/DonorsChoose_2018/9set3.pckl','rb')\n", "set3_train, set3_test=pickle.load(f)\n", "f.close()"] }, { "cell_type": "code", "execution_count": 4, "metadata": {}, "outputs": [], "source": ["import pickle as pickle\n", "#with open('C:/Users/pramod reddy chandi/Desktop/pram/applied ai course/DonorsChoose_2018/cat_num.pckl', 'rb') as f:\n", "f=open('C:/Users/pramod reddy chandi/Desktop/pram/applied ai course/DonorsChoose_2018/9set4.pckl','rb')\n", "set4_train, set4_test=pickle.load(f)\n", "f.close()"] }, { "cell_type": "code", "execution_count": 5, "metadata": {}, "outputs": [], "source": ["import pickle as pickle\n", "#with open('C:/Users/pramod reddy chandi/Desktop/pram/applied ai course/DonorsChoose_2018/cat_num.pckl', 'rb') as f:\n", "f=open('C:/Users/pramod reddy chandi/Desktop/pram/applied ai course/DonorsChoose_2018/9y_values.pckl','rb')\n", "y_train, y_test=pickle.load(f)\n", "f.close()"] }, { "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "4d-Zb0b5pdUr" }, "source": ["- we need to merge all the numerical vectors i.e catogorical, text, numerical vectors"] }, { "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "paRNuUWJpdUx" }, "source": ["# Assignment 9: RF and GBDT"] }, { "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "ilMDKELQpdUy" }, "source": ["#### Response Coding: Example"] }, { "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "DM6wDoX_pdUy" }, "source": ["

\n", "\n", "> The response tabel is built only on train dataset.\n", "> For a category which is not there in train data and present in test data, we will encode them with default values\n", "Ex: in our test data if have State: D then we encode it as [0.5, 0.05]"] }, { "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "YFxlMjpMpdUy" }, "source": ["
\n", "

1. **Apply both Random Forrest and GBDT on these feature sets**\n", "
\n", "
   - Set 1: categorical(instead of one hot encoding, try [response coding](): use probability values), numerical features + project_title(BOW) + preprocessed_eassay (BOW)
\n", "

- Set 2: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF)+ preprocessed_eassay (TFIDF)
  \n", "
- Set 3: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(AVG W2V)+ preprocessed_eassay (AVG W2V)
  \n", "
- Set 4: categorical(instead of one hot encoding, try response coding: use probability values), numerical features + project_title(TFIDF W2V)+ preprocessed_eassay (TFIDF W2V)

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

2. **The hyper paramter tuning (Consider any two hyper parameters preferably n_estimators, max_depth)**\n", "
   \n", "
   - Find the best hyper parameter which will give the maximum AUC value
     \n", "
   - find the best hyper paramter using k-fold cross validation/simple cross validation data
     \n", "
   - use gridsearch cv or randomsearch cv or you can write your own for loops to do this task
     \n", "

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

3. \n", " **Representation of results**\n", "
   \n", "
   - You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure\n", "

with X-axis as **n_estimators**, Y-axis as **max_depth**, and Z-axis as **AUC Score** , we have given the notebook which explains how to plot this 3d plot, you can find it in the same drive *3d_scatter_plot.ipynb*
\n", "

or

\n", "

- You need to plot the performance of model both on train data and cross validation data for each hyper parameter, like shown in the figure\n", "

[seaborn heat maps](#) with rows as **n_estimators**, columns as **max_depth**, and values inside the cell representing **AUC Score**
\n", "

- You can choose either of the plotting techniques: 3d plot or heat map
\n", "
- Once after you found the best hyper parameter, you need to train your model with it, and find the AUC on test data and plot the ROC curve on both train and test.\n",
"
\n", "
- Along with plotting ROC curve, you need to print the [confusion matrix](#) with predicted and original labels of

test data points\n", "
\n", "
\n", "
\n", "

4. **Conclusion**\n", "
\n", "
   - You need to summarize the results at the end of the notebook, summarize it in the table format. To print out a table please refer to this prettytable library

link \n", "
\n", "
\n", "

\n", "
"] }, { "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "Ufe50MBppdUz" }, "source": ["

Note: Data Leakage

\n", "\n", "1. There will be an issue of data-leakage if you vectorize the entire data and then split it into train/cv/test.\n", "2. To avoid the issue of data-leakag, make sure to split your data first and then vectorize it. \n", "3. While vectorizing your data, apply the method fit_transform() on you train data, and apply the method transform() on cv/test data.\n", "4. For more details please go through this link."] }, { "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "h9vP1ObDpdUz" }, "source": ["

2. Random Forest and GBDT

"] }, { "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "5A9anTSypdU3" }, "source": ["

2.4 Applying Random Forest

\n", "\n", "
Apply Random Forest on different kind of featurization as mentioned in the instructions\n", "
For Every model that you work on make sure you do the step 2 and step 3 of instrucations"] }, { "cell_type": "markdown", "metadata": { "colab_type": "text", "id": "05E7WFn9pdU3" }, "source": ["### 2.4.1 Applying Random Forests on BOW, SET 1"] }, { "cell_type": "code", "execution_count": 6, "metadata": { "colab": {}, "colab_type": "code", "id": "e60FAUPBpdU3" }, "outputs": [], "source": ["X_tr=set1_train.tocsr()\n", "X_te=set1_test.tocsr()\n", "\n"] }, { "cell_type": "code", "execution_count": 147, "metadata": {}, "outputs": [{ "data": { "text/html": [" \n", " "] }, "metadata": {}, "output_type": "display_data" }], "source": ["from sklearn.ensemble import RandomForestClassifier\n", "import matplotlib.pyplot as plt\n", "from sklearn.metrics import roc_auc_score\n", "import plotly.offline as offline\n", "import plotly.graph_objs as go\n", "offline.init_notebook_mode()\n", "import numpy as np\n", "\n", "\n", "n_estimators_param = [10, 100, 500]\n", "max_depth_param = [10, 50, 100, 500]\n", "\n", "train_results=[]\n", "test_results=[]\n", "\n", "for i in n_estimators_param:\n", " for j in max_depth_param:\n", "\n", " classifier = RandomForestClassifier(n_estimators = i,max_depth=j,class_weight='balanced',n_jobs=-1)\n", " classifier.fit(X_tr, y_train)\n", " \n", "\n", " y_train_pred = classifier.predict_proba(X_tr)[:,1]\n", " false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, y_train_pred)\n", " roc_auc = auc(false_positive_rate, true_positive_rate)\n", " # Add auc score to previous train results\n", " train_results.append(roc_auc) \n", " \n", " y_test_pred = classifier.predict_proba(X_te)[:,1]\n", " false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_test_pred)\n", " roc_auc = auc(false_positive_rate, true_positive_rate)\n", " # Add auc score to previous test results\n", " test_results.append(roc_auc)\n", " \n", " "] }, { "cell_type": "code", "execution_count": 8, "metadata": {}, "outputs": [{ "data": { "text/plain": ["[0.7204941297515751,\n", " 0.9874313801950426,\n", " 0.998546375136075,\n", " 0.9994137892936554,\n", " 0.7802034038490959,\n", " 0.9998700265678572,\n", " 0.9999589053361206,\n", " 0.9999703870598451,\n", " 0.7861982233110401,\n", " 0.9999457690471989,\n", " 0.999996989388611,\n", " 0.9999977229754633]"] }, "execution_count": 8, "metadata": {}, "output_type": "execute_result" }], "source": ["train_results"] }, { "cell_type": "code", "execution_count":

10, "metadata": {}, "outputs": [{ "data": { "text/plain": ["
[0.6536462469725929,\n", " 0.6462157433979174,\n", "
0.6157491803032404,\n", " 0.6060700788882816,\n", "
0.6887199201792435,\n", " 0.6964140800718954,\n", "
0.6986572799408926,\n", " 0.6884760234899352,\n", "
0.6954558009530452,\n", " 0.7052703845613543,\n", "
0.7129049332469746,\n", " 0.7149138497853987]"] },
"execution_count": 10, "metadata": {}, "output_type":
"execute_result" }], "source": ["test_results"] }, {
"cell_type": "code", "execution_count": 11, "metadata": {},
"outputs": [{ "data": { "text/html": [" \n", " "] },
"metadata": {}, "output_type": "display_data" }], "source": [
"#plotting\n", "import plotly.offline as offline\n", "import
plotly.graph_objs as go\n", "offline.init_notebook_mode()\n",
"import numpy as np\n", "\n", "x1=[10, 10, 10, 10, 100, 100,
100, 100, 500, 500, 500, 500]\n", "y1=[10, 50, 100, 500, 10,
50, 100, 500, 10, 50, 100, 500]\n", "z1= train_results\n", "z2=
test_results\n"] }, { "cell_type": "code", "execution_count":
12, "metadata": {}, "outputs": [{ "data": {
"application/vnd.plotly.v1+json": { "config": { "linkText":
"Export to plot.ly", "plotlyServerURL": "https://plot.ly",
"responsive": true, "showLink": false }, "data": [{ "name":
"train", "type": "scatter3d", "uid": "5a5d1164-d200-48e2-ad84-
edbb92d5beaa", "x": [10, 10, 10, 10, 100, 100, 100, 100, 500,
500, 500, 500], "y": [10, 50, 100, 500, 10, 50, 100, 500, 10,
50, 100, 500], "z": [0.7204941297515751, 0.9874313801950426,
0.998546375136075, 0.9994137892936554, 0.7802034038490959,
0.9998700265678572, 0.9999589053361206, 0.9999703870598451,
0.7861982233110401, 0.9999457690471989, 0.999996989388611,
0.9999977229754633] }, { "name": "test validation", "type":
"scatter3d", "uid": "11c93e56-018d-46d4-a82c-dc9d10f5396c",
"x": [10, 10, 10, 10, 100, 100, 100, 100, 500, 500, 500, 500
], "y": [10, 50, 100, 500, 10, 50, 100, 500, 10, 50, 100, 500
], "z": [0.6536462469725929, 0.6462157433979174,
0.6157491803032404, 0.6060700788882816, 0.6887199201792435,
0.6964140800718954, 0.6986572799408926, 0.6884760234899352,
0.6954558009530452, 0.7052703845613543, 0.7129049332469746,
0.7149138497853987] }], "layout": { "scene": { "xaxis": {
"title": { "text": "n_estimators" } }, "yaxis": { "title": {
"text": "max_depth" } }, "zaxis": { "title": { "text": "AUC" }
} } } }, "text/html": ["
\n", " \n", " \n", "
\n", " \n", "
"] }, "metadata": {}, "output_type": "display_data" }],
"source": ["# https://plot.ly/python/3d-axes/\n", "trace1 =
go.Scatter3d(x=x1,y=y1,z=z1, name = 'train')\n", "trace2 =
go.Scatter3d(x=x1,y=y1,z=z2, name = 'test validation')\n",
"data = [trace1, trace2]\n", "\n", "layout = go.Layout(scene =
dict(\n", " xaxis = dict(title='n_estimators'),\n", " yaxis =
dict(title='max_depth'),\n", " zaxis = dict(title='AUC'),))\n",

"\n", "fig = go.Figure(data=data, layout=layout)\n", "offline.iplot(fig, filename='set1')"] }, { "cell_type": "code", "execution_count": 19, "metadata": {}, "outputs": [{ "data": { "text/html": [" \n", " "] }, "metadata": {}, "output_type": "display_data" }], "source": ["#hyperprameter cross checking again for max depth\n", "from sklearn.ensemble import RandomForestClassifier\n", "import matplotlib.pyplot as plt\n", "from sklearn.metrics import roc_auc_score\n", "import plotly.offline as offline\n", "import plotly.graph_objs as go\n", "offline.init_notebook_mode()\n", "import numpy as np\n", "from sklearn.metrics import roc_curve, auc\n", "\n", "max_depth_param=[2,3,4,5,6,7,8,9,10]\n", "\n", "train_results1=[]\n", "test_results1=[]\n", "\n", "\n", "\n", "for i in max_depth_param:\n", " classifier = RandomForestClassifier(n_estimators = 500,max_depth= i,class_weight='balanced',n_jobs=-1)\n", " classifier.fit(X_tr, y_train)\n", " \n", "\n", " y_train_pred = classifier.predict_proba(X_tr)[:,1]\n", " false_positive_rate, true_positive_rate, thresholds = roc_curve(y_train, y_train_pred)\n", " roc_auc = auc(false_positive_rate, true_positive_rate)\n", " # Add auc score to previous train results\n", " train_results1.append(roc_auc) \n", " \n", " y_test_pred = classifier.predict_proba(X_te)[:,1]\n", " false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_test_pred)\n", " roc_auc = auc(false_positive_rate, true_positive_rate)\n", " # Add auc score to previous test results\n", " test_results1.append(roc_auc)\n", " "] }, { "cell_type": "code", "execution_count": 21, "metadata": {}, "outputs": [{ "data": { "image/png": "iVBORw0KGgoAAAANSUhEUgAAAYsAAAELCAYAAAAoUKpTAAAABHNCSVQICAgIfAh
"text/plain": ["

"] }, "metadata": { "needs_background": "light" }, "output_type": "display_data" }], "source": ["from matplotlib.legend_handler import HandlerLine2D\n", "line1, = plt.plot(max_depth_param, train_results1, 'b', label=\"Train AUC\")\n", "line2, = plt.plot(max_depth_param, test_results1, 'r', label=\"Test AUC\")\n", "\n", "plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})\n", "plt.ylabel(\"AUC score\")\n", "plt.xlabel(\"max_depth\")\n", "plt.show() "] }, { "cell_type": "markdown", "metadata": {}, "source": ["#as dept increase s we could see model is overfitting .therefore considering 4 as the max depth"] }, { "cell_type": "code", "execution_count": 22, "metadata": {}, "outputs": [{ "data": { "text/html": [" \n", " "] }, "metadata": {}, "output_type": "display_data" }, { "data": { "image/png":

"iVBORw0KGgoAAAANSUhEUgAAAYsAAAEKCAYAAADjDHn2AAAABHNCSVQICA(
"text/plain": ["

"] }, "metadata": { "needs_background":
"light" }, "output_type": "display_data" }],
"source": ["#hyperprameter cross checking
again for n estimators\n", "from
sklearn.ensemble import
RandomForestClassifier\n", "import
matplotlib.pyplot as plt\n", "from
sklearn.metrics import roc_auc_score\n",
"import plotly.offline as offline\n", "import
plotly.graph_objs as go\n",
"offline.init_notebook_mode()\n", "import
numpy as np\n", "\n", "\n",
"n_estimators_param=
[5,10,50,100,200,500,1000]\n", "\n",
"train_results2=[]\n", "test_results2=[]\n",
"\n", "\n", "\n", "for i in
n_estimators_param:\n", " classifier =
RandomForestClassifier(n_estimators =
i,max_depth=
4,class_weight='balanced',n_jobs=-1)\n", "
classifier.fit(X_tr, y_train)\n", " \n", "\n",
" y_train_pred =
classifier.predict_proba(X_tr)[:,1]\n", "
false_positive_rate, true_positive_rate,
thresholds = roc_curve(y_train,
y_train_pred)\n", " roc_auc =
auc(false_positive_rate,
true_positive_rate)\n", " # Add auc score to
previous train results\n", "
train_results2.append(roc_auc) \n", " \n", "
y_test_pred = classifier.predict_proba(X_te)
[:,1]\n", " false_positive_rate,
true_positive_rate, thresholds =
roc_curve(y_test, y_test_pred)\n", " roc_auc =
auc(false_positive_rate,
true_positive_rate)\n", " # Add auc score to
previous test results\n", "
test_results2.append(roc_auc)\n", " \n", "from
matplotlib.legend_handler import
HandlerLine2D\n", "line1, =
plt.plot(n_estimators_param, train_results2,
'b', label=\"Train AUC\")\n", "line2, =
plt.plot(n_estimators_param, test_results2,
'r', label=\"Test AUC\")\n", "\n",
"plt.legend(handler_map={line1:
HandlerLine2D(numpoints=2)})\n",
"plt.ylabel(\"AUC score\")\n", "plt.xlabel(\"n

estimators\")\n", "plt.show() "] }, {
"cell_type": "markdown", "metadata": {},
"source": ["#we could see that
hyperparameters are n_estimators=200
,max_depth=4"] }, { "cell_type": "code",
"execution_count": 23, "metadata": {},
"outputs": [{ "data": { "image/png":
"iVBORw0KGgoAAAANSUhEUgAAAYUAAAEWCAYAAACJ0YulAAAABHNCSV
"text/plain": ["

"] }, "metadata": {
"needs_background": "light" },
"output_type": "display_data" }],
"source": ["# finding the train and
test AU\n", "from sklearn.ensemble
import RandomForestClassifier\n",
"from sklearn.metrics import
roc_curve, auc\n", "model =
RandomForestClassifier(n_estimators =
200, max_depth= 4,
class_weight='balanced',
n_jobs=-1)\n", "model.fit(X_tr,
y_train)\n", "\n", "#
roc_auc_score(y_true, y_score) the 2nd
parameter should be probability
estimates of the positive class\n", "#
not the predicted outputs\n", "\n",
"y_train_pred =
model.predict_proba(X_tr)[:,1] \n",
"y_test_pred =
model.predict_proba(X_te)[:,1]\n",
"\n", "\n", "train_fpr, train_tpr,
tr_thresholds = roc_curve(y_train,
y_train_pred)\n", "test_fpr, test_tpr,
te_thresholds = roc_curve(y_test,
y_test_pred)\n", "\n",
"plt.plot(train_fpr, train_tpr,
label=\"Train AUC
=\"+str(auc(train_fpr,
train_tpr)))\n", "plt.plot(test_fpr,
test_tpr, label=\"Test AUC
=\"+str(auc(test_fpr, test_tpr)))\n",
"plt.legend()\n", "plt.xlabel(\"True
Positive Rate(TPR)\")\n",
"plt.ylabel(\"False Positive
Rate(FPR)\")\n", "plt.title(\"set1
AUC\")\n", "plt.grid()\n",
"plt.show()"] }, { "cell_type":
"code", "execution_count": 24,
"metadata": {}, "outputs": [],

"source": ["def predict(proba, threshould, fpr, tpr):\n", " \n", " t = threshould[np.argmax(fpr*(1-tpr))]\n", " \n", " # (tpr*(1-fpr)) will be maximum if your fpr is very low and tpr is very high\n", " \n", " print(\"the maximum value of tpr*(1-fpr)\", max(tpr*(1-fpr)), \"for threshold\", np.round(t,3))\n", " predictions = []\n", " for i in proba:\n", " if i>=t:\n", " predictions.append(1)\n", " else:\n", " predictions.append(0)\n", " return predictions"] }, { "cell_type": "code", "execution_count": 25, "metadata": {}, "outputs": [{ "name": "stdout", "output_type": "stream", "text": ["Train confusion matrix\n", "the maximum value of tpr*(1-fpr) 0.2499999979647145 for threshold 0.493\n", "[[5542 5541]\n", " [14427 47686]]\n"] }], "source": ["from sklearn.metrics import confusion_matrix\n", "print(\"Train confusion matrix\")\n", "print(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)))"] }, { "cell_type": "code", "execution_count": 27, "metadata": {}, "outputs": [{ "name": "stdout", "output_type": "stream", "text": ["the maximum value of tpr*(1-fpr) 0.2499999979647145 for threshold 0.493\n"] }], "source": ["import pandas as pd\n", "conf_matr_df_train_1 = pd.DataFrame(confusion_matrix(y_train, predict(y_train_pred, tr_thresholds, train_fpr, train_fpr)), range(2),range(2))"] }, { "cell_type": "code", "execution_count": 29, "metadata": {}, "outputs": [{ "data": { "text/plain": ["Text(0.5, 1.0, 'Confusion matrix - Train data set 1')"] }, "execution_count": 29, "metadata": {}, "output_type": "execute_result" }, { "data": { "image/png": "iVBORw0KGgoAAAANSUhEUgAAAZQAAAEkCAYAAAAID8fVAAAABH "text/plain": ["

"] }, "metadata": {},
"output_type": "display_data"
}], "source": ["import
seaborn as sns\n",
"sns.set(font_scale=1.4)#for
label size\n",
"sns.heatmap(conf_matr_df_train_1,
annot=True,annot_kws=
{\"size\": 16}, fmt='g')\n",
"\n", "plt.xlabel(\"Actual
class\")\n",
"plt.ylabel(\"Predicted
class\")\n",
"plt.title(\"Confusion matrix
-Train data set 1\")"] }, {
"cell_type": "code",
"execution_count": 30,
"metadata": {}, "outputs": [{
"name": "stdout",
"output_type": "stream",
"text": ["Test confusion
matrix\n", "the maximum value
of tpr*(1-fpr)
0.24999999161092998 for
threshold 0.504\n", "[[4134
1325]\n", " [15587 15006]]\n"
] }], "source": ["from
sklearn.metrics import
confusion_matrix\n",
"print(\"Test confusion
matrix\")\n",
"print(confusion_matrix(y_test,
predict(y_test_pred,
tr_thresholds, test_fpr,
test_fpr)))"] }, {
"cell_type": "code",
"execution_count": 31,
"metadata": {}, "outputs": [{
"name": "stdout",
"output_type": "stream",
"text": ["the maximum value
of tpr*(1-fpr)
0.24999999161092998 for
threshold 0.504\n"] }],
"source": [
"conf_matr_df_test_1 =
pd.DataFrame(confusion_matrix(y_test,
predict(y_test_pred,

tr_thresholds, test_fpr,
test_fpr)),
range(2),range(2))"] }, {
"cell_type": "code",
"execution_count": 32,
"metadata": {}, "outputs": [{
"data": { "text/plain": [
"Text(0.5, 1.0, 'Confusion
matrix -Test data-set 1')"]
}, "execution_count": 32,
"metadata": {}, "output_type":
"execute_result" }, { "data":
{ "image/png":
"iVBORw0KGgoAAAANSUhEUgAAAZQAAAEkCAYAAAAID8fVAA
"text/plain": ["

"] }, "metadata":
{}, "output_type":
"display_data" }],
"source": [
"sns.set(font_scale=1.4)#for
label size\n",
"sns.heatmap(conf_matr_df_test_1,
annot=True,annot_kws=
{\"size\": 16},
fmt='g')\n", "\n",
"plt.xlabel(\"Actual
class\")\n",
"plt.ylabel(\"Predicted
class\")\n",
"plt.title(\"Confusion
matrix -Test data-set
1\")"] }, {
"cell_type":
"markdown",
"metadata": {
"colab_type": "text",
"id": "leF7Ix_GpdU4"
}, "source": ["###
2.4.2 Applying Random
Forests on TFIDF, SET
2"] }, {
"cell_type": "code",
"execution_count":
33, "metadata": {
"colab": {},
"colab_type": "code",
"id": "G2v7iPsipdU4"
}, "outputs": [],
"source": [

"X_tr=set2_train.tocsr()\n",
"X_te=set2_test.tocsr()"
] }, { "cell_type":
"code",
"execution_count":
170, "metadata": {
"colab": {},
"colab_type": "code",
"id": "G2v7iPsipdU4"
}, "outputs": [{
"data": {
"text/html": [" \n",
" "] }, "metadata":
{}, "output_type":
"display_data" }],
"source": ["from
sklearn.ensemble
import
RandomForestClassifier\n",
"import
matplotlib.pyplot as
plt\n", "from
sklearn.metrics
import
roc_auc_score\n",
"import
plotly.offline as
offline\n", "import
plotly.graph_objs as
go\n",
"offline.init_notebook_mode()\n",
"import numpy as
np\n", "\n", "\n",
"n_estimators_param =
[10, 100, 500]\n",
"max_depth_param =
[10, 50, 100,
500]\n", "\n",
"train_results=[]\n",
"test_results=[]\n",
"\n", "for i in
n_estimators_param:\n",
" for j in
max_depth_param:\n",
"\n", " classifier =
RandomForestClassifier(n_estimators
= i,max_depth=
j,class_weight='balanced',n_jobs=-1)\n",
"
classifier.fit(X_tr,

y_train)\n", " \n",
"\n", " y_train_pred
=
classifier.predict_proba(X_tr)
[:,1]\n", "
false_positive_rate,
true_positive_rate,
thresholds =
roc_curve(y_train,
y_train_pred)\n", "
roc_auc =
auc(false_positive_rate,
true_positive_rate)\n",
" # Add auc score to
previous train
results\n", "
train_results.append(roc_auc)
\n", " \n", "
y_test_pred =
classifier.predict_proba(X_te)
[:,1]\n", "
false_positive_rate,
true_positive_rate,
thresholds =
roc_curve(y_test,
y_test_pred)\n", "
roc_auc =
auc(false_positive_rate,
true_positive_rate)\n",
" # Add auc score to
previous test
results\n", "
test_results.append(roc_auc)\n",
" \n", " "] }, {
"cell_type": "code",
"execution_count":
37, "metadata": {
"colab": {},
"colab_type": "code",
"id": "G2v7iPsipdU4"
}, "outputs": [{
"data": {
"text/plain": ["
[0.7417557900728293,\n",
"
0.993180828655031,\n",
"
0.9993455017999106,\n",
"
0.9994975344065997,\n",

"
0.7935303105935989,\n",
"
0.999931965993197,\n",
"
0.9999580497269009,\n",
"
0.9999717336638296,\n",
"
0.8028513196135809,\n",
"
0.9999664721755539,\n",
"
0.9999764721755539,\n",
"
0.9999964721755539]"
] },
"execution_count":
37, "metadata": {},
"output_type":
"execute_result" }],
"source": [
"train_results"] },
{ "cell_type":
"code",
"execution_count":
38, "metadata": {
"colab": {},
"colab_type": "code",
"id": "G2v7iPsipdU4"
}, "outputs": [{
"data": {
"text/plain": ["
[0.6474052580743127,\n",
"
0.6265276984756352,\n",
"
0.5968250007108975,\n",
"
0.5948936526905276,\n",
"
0.6854534469824943,\n",
"
0.688632373048712,\n",
"
0.6846656635202172,\n",
"
0.6849438012509007,\n",
"
0.6920002041588785,\n",

"
0.7001789030791831,\n",
"
0.7124532041588785,\n",
"
0.7254832041588785]"
] },
"execution_count":
38, "metadata": {},
"output_type":
"execute_result" }],
"source": [
"test_results"] }, {
"cell_type": "code",
"execution_count":
39, "metadata": {
"colab": {},
"colab_type": "code",
"id": "G2v7iPsipdU4"
}, "outputs": [{
"data": {
"text/html": [" \n",
" "] }, "metadata":
{}, "output_type":
"display_data" }],
"source": [
"#plotting\n",
"import
plotly.offline as
offline\n", "import
plotly.graph_objs as
go\n",
"offline.init_notebook_mode()\n",
"import numpy as
np\n", "\n", "x1=[10,
10, 10, 10, 100, 100,
100, 100, 500, 500,
500, 500]\n", "y1=
[10, 50, 100, 500,
10, 50, 100, 500, 10,
50, 100, 500]\n",
"z1=
train_results\n",
"z2= test_results"]
}, { "cell_type":
"code",
"execution_count":
40, "metadata": {
"colab": {},
"colab_type": "code",

"id": "G2v7iPsipdU4"
}, "outputs": [{
"data": {
"application/vnd.plotly.v1+json":
{ "config": {
"linkText": "Export
to plot.ly",
"plotlyServerURL":
"https://plot.ly",
"responsive": true,
"showLink": false },
"data": [{ "name":
"train", "type":
"scatter3d", "uid":
"04d3d7b7-4dd1-4b05-
89db-3bb08b557c64",
"x": [10, 10, 10,
10, 100, 100, 100,
100, 500, 500, 500,
500], "y": [10, 50,
100, 500, 10, 50,
100, 500, 10, 50,
100, 500], "z": [
0.7417557900728293,
0.993180828655031,
0.9993455017999106,
0.9994975344065997,
0.7935303105935989,
0.999931965993197,
0.9999580497269009,
0.9999717336638296,
0.8028513196135809,
0.9999664721755539,
0.9999764721755539,
0.9999964721755539]
}, { "name": "test
validation", "type":
"scatter3d", "uid":
"ec11524e-b1bb-4ab4-
b2d4-f47466184c0f",
"x": [10, 10, 10,
10, 100, 100, 100,
100, 500, 500, 500,
500], "y": [10, 50,
100, 500, 10, 50,
100, 500, 10, 50,
100, 500], "z": [
0.6474052580743127,
0.6265276984756352,
0.5968250007108975,

0.5948936526905276,
0.6854534469824943,
0.688632373048712,
0.6846656635202172,
0.6849438012509007,
0.6920002041588785,
0.7001789030791831,
0.7124532041588785,
0.7254832041588785]
}], "layout": {
"scene": { "xaxis": {
"title": { "text":
"n_estimators" } },
"yaxis": { "title": {
"text": "max_depth" }
}, "zaxis": {
"title": { "text":
"AUC" } } } } },
"text/html": ["
\n", " \n", " \n", "
\n", " \n", "
"] }, "metadata":
{}, "output_type":
"display_data" }],
"source": ["#
https://plot.ly/python/3d-
axes/\n", "trace1 =
go.Scatter3d(x=x1,y=y1,z=z1,
name = 'train')\n",
"trace2 =
go.Scatter3d(x=x1,y=y1,z=z2,
name = 'test
validation')\n",
"data = [trace1,
trace2]\n", "\n",
"layout =
go.Layout(scene =
dict(\n", " xaxis =
dict(title='n_estimators'),\n",
" yaxis =
dict(title='max_depth'),\n",
" zaxis =
dict(title='AUC'),))\n",
"\n", "fig =
go.Figure(data=data,
layout=layout)\n",
"offline.iplot(fig,
filename='set2')"]
}, { "cell_type":
"code",

"execution_count":
41, "metadata": {
"colab": {},
"colab_type": "code",
"id": "G2v7iPsipdU4"
}, "outputs": [{
"data": {
"text/html": [" \n",
" "] }, "metadata":
{}, "output_type":
"display_data" }, {
"data": {
"image/png":
"iVBORw0KGgoAAAANSUhEUgAAAaYAAAEYCAYAAAAXsV
"text/plain": ["

"] },
"metadata":
{},
"output_type":
"display_data"
}],
"source": [
"#hyperprameter
cross
checking
again for max
depth\n",
"from
sklearn.ensemble
import
RandomForestClassifier\n",
"import
matplotlib.pyplot
as plt\n",
"from
sklearn.metrics
import
roc_auc_score\n",
"import
plotly.offline
as
offline\n",
"import
plotly.graph_objs
as go\n",
"offline.init_notebook_mode()\n",
"import numpy
as np\n",
"from

```
sklearn.metrics
import
roc_curve,
auc\n", "\n",
"max_depth_param=
[2,3,4,5,6,7,8,9,10]\n",
"\n",
"train_results1=
[]\n",
"test_results1=
[]\n", "\n",
"\n", "\n",
"for i in
max_depth_param:\n",
" classifier
=
RandomForestClassifier(n_estimators
=
500,max_depth=
i,class_weight='balanced',n_jobs=-1)\n'
"
classifier.fit(X_tr,
y_train)\n",
" \n", "\n",
"
y_train_pred
=
classifier.predict_proba(X_tr)
[:,1]\n", "
false_positive_rate,
true_positive_rate,
thresholds =
roc_curve(y_train,
y_train_pred)\n",
" roc_auc =
auc(false_positive_rate,
true_positive_rate)\n",
" # Add auc
score to
previous
train
results\n", "
train_results1.append(roc_auc)
\n", " \n", "
y_test_pred =
classifier.predict_proba(X_te)
[:,1]\n", "
false_positive_rate,
true_positive_rate,
thresholds =
```

roc_curve(y_test,
y_test_pred)\n",
" roc_auc =
auc(false_positive_rate,
true_positive_rate)\n",
" # Add auc
score to
previous test
results\n", "
test_results1.append(roc_auc)\n",
" \n", "\n",
"from
matplotlib.legend_handler
import
HandlerLine2D\n",
"line1, =
plt.plot(max_depth_param,
train_results1,
'b',
label=\"Train
AUC\")\n",
"line2, =
plt.plot(max_depth_param,
test_results1,
'r',
label=\"Test
AUC\")\n",
"\n",
"plt.legend(handler_map=
{line1:
HandlerLine2D(numpoints=2)})\n",
"plt.ylabel(\"AUC
score\")\n",
"plt.xlabel(\"max_depth\")\n",
"plt.show() "
] }, {
"cell_type":
"markdown",
"metadata": {
"colab": {},
"colab_type":
"code", "id":
"G2v7iPsipdU4"
}, "source":
["#as dept
increase s we
could see
model is
overfitting
.therefore

considering 4
as the max
depth"] }, {
"cell_type":
"code",
"execution_count":
42,
"metadata": {
"colab": {},
"colab_type":
"code", "id":
"G2v7iPsipdU4"
}, "outputs":
[{ "data": {
"text/html":
[" \n", " "
] },
"metadata":
{},
"output_type":
"display_data"
}, { "data":
{
"image/png":
"iVBORw0KGgoAAAANSUhEUgAAAZ0AAAEYCAYAAA
"text/plain":
["

"]
},
"metadata":
{},
"output_type":
"display_data"
}],
"source":
[
"#hyperprameter
cross
checking
again
for n
estimators\n",
"from
sklearn.ensemble
import
RandomForestClassifier\n",
"import
matplotlib.pyplot
as

```
plt\n",
"from
sklearn.metrics
import
roc_auc_score\n",
"import
plotly.offline
as
offline\n",
"import
plotly.graph_objs
as
go\n",
"offline.init_notebook_mode()\n",
"import
numpy
as
np\n",
"\n",
"\n",
"n_estimators_param=
[5,10,50,100,200,500,1000]\n",
"\n",
"train_results2=
[]\n",
"test_results2=
[]\n",
"\n",
"\n",
"\n",
"for
i in
n_estimators_param:\n",
"
classifier
=
RandomForestClassifier(n_estimators
=
i,max_depth=
4,class_weight='balanced',n_jobs=-1
"
classifier.fit(X_tr,
y_train)\n",
"
\n",
"\n",
"
y_train_pred
=
classifier.predict_proba(X_tr)
```

```
[:,1]\n",
"
false_positive_rate,
true_positive_rate,
thresholds
=
roc_curve(y_train,
y_train_pred)\n",
"
roc_auc
=
auc(false_positive_rate,
true_positive_rate)\n",
" #
Add
auc
score
to
previous
train
results\n",
"
train_results2.append(roc_auc)
\n",
"
\n",
"
y_test_pred
=
classifier.predict_proba(X_te)
[:,1]\n",
"
false_positive_rate,
true_positive_rate,
thresholds
=
roc_curve(y_test,
y_test_pred)\n",
"
roc_auc
=
auc(false_positive_rate,
true_positive_rate)\n",
" #
Add
auc
score
to
previous
test
```

results\n",
"
test_results2.append(roc_auc)\n",
"
\n",
"from
matplotlib.legend_handler
import
HandlerLine2D\n",
"line1,
=
plt.plot(n_estimators_param,
train_results2,
'b',
label=\"Train
AUC\")\n",
"line2,
=
plt.plot(n_estimators_param,
test_results2,
'r',
label=\"Test
AUC\")\n",
"\n",
"plt.legend(handler_map=
{line1:
HandlerLine2D(numpoints=2)})\n",
"plt.ylabel(\"AUC
score\")\n",
"plt.xlabel(\"n
estimators\")\n",
"plt.show()
"]
}, {
"cell_type":
"markdown",
"metadata":
{
"colab":
{},
"colab_type":
"code",
"id":
"G2v7iPsipdU4"
},
"source":
[
"#we
could
see

that
hyperparameters
are
n_estimators=200
,max_depth=4"
] },
{
"cell_type":
"code",
"execution_count":
43,
"metadata":
{
"colab":
{},
"colab_type":
"code",
"id":
"G2v7iPsipdU4"
},
"outputs":
[{
"data":
{
"image/png":
"iVBORw0KGgoAAAANSUhEUgAAAZUAAAEkCA
"text/plain":
["

"
]
},
"metadata":
{},
"output_type":
"display_data"
}
],
"source":
[
"#
finding
the
train
and
test
AU\n",
"from
sklearn.ensemble
import

```
RandomForestClassifier\n",
"from
sklearn.metrics
import
roc_curve,
auc\n",
"model
=
RandomForestClassifier(n_estima
=
200,
max_depth=
4,
class_weight='balanced',
n_jobs=-1)\n",
"model.fit(X_tr,
y_train)\n",
"\n",
"#
roc_auc_score(y_true,
y_score)
the
2nd
parameter
should
be
probability
estimates
of
the
positive
class\n",
"#
not
the
predicted
outputs\n",
"\n",
"y_train_pred
=
model.predict_proba(X_tr)
[:,1]
\n",
"y_test_pred
=
model.predict_proba(X_te)
[:,1]\n",
"\n",
"\n",
"train_fpr,
```

train_tpr,
tr_thresholds
=
roc_curve(y_train,
y_train_pred)\n",
"test_fpr,
test_tpr,
te_thresholds
=
roc_curve(y_test,
y_test_pred)\n",
"\n",
"plt.plot(train_fpr,
train_tpr,
label=\"Train
AUC
=\"+str(auc(train_fpr,
train_tpr)))\n",
"plt.plot(test_fpr,
test_tpr,
label=\"Test
AUC
=\"+str(auc(test_fpr,
test_tpr)))\n",
"plt.legend()\n",
"plt.xlabel(\"True
Positive
Rate(TPR)\")\n",
"plt.ylabel(\"False
Positive
Rate(FPR)\")\n",
"plt.title(\"set2
AUC\")\n",
"plt.grid()\n",
"plt.show()"
]
},
{
"cell_type":
"code",
"execution_count":
44,
"metadata":
{
"colab":
{},
"colab_type":
"code",
"id":
"G2v7iPsipdU4"

},
"outputs":
[],
"source":
[
"def
predict(proba,
threshould,
fpr,
tpr):\n",
"
\n",
"
t
=
threshould[np.argmax(fpr*
(1-
tpr))]\n",
"
\n",
"
#
(tpr*
(1-
fpr))
will
be
maximum
if
your
fpr
is
very
low
and
tpr
is
very
high\n",
"
\n",
"
print(\"the
maximum
value
of
tpr*
(1-
fpr)\",
max(tpr*

(1-
fpr)),
\"for
threshold\",
np.round(t,3))\n",
"
predictions
=
[]\n",
"
for
i
in
proba:\n",
"
if
i>=t:\n",
"
predictions.append(1)\n",
"
else:\n",
"
predictions.append(0)\n",
"
return
predictions"
]
},
{
"cell_type":
"code",
"execution_count":
45,
"metadata":
{
"colab":
{},
"colab_type":
"code",
"id":
"G2v7iPsipdU4"
},
"outputs":
[
{
"name":
"stdout",
"output_type":
"stream",
"text":

[
"Train
confusion
matrix\n",
"the
maximum
value
of
tpr*
(1-
fpr)
0.2499999979647145
for
threshold
0.494\n",
"
[[
5542
5541]\n",
"
[14141
47972]]\n"
]
}
],
"source":
[
"from
sklearn.metrics
import
confusion_matrix\n",
"print(\"Train
confusion
matrix\")\n",
"print(confusion_matrix(y_trai
predict(y_train_pred,
tr_thresholds,
train_fpr,
train_fpr)))"
]
},
{
"cell_type":
"code",
"execution_count":
46,
"metadata":
{
"colab":
{},

"colab_type":
"code",
"id":
"G2v7iPsipdU4"
},
"outputs":
[
{
"name":
"stdout",
"output_type":
"stream",
"text":
[
"the
maximum
value
of
tpr*
(1-
fpr)
0.2499999979647145
for
threshold
0.494\n"
]
}
],
"source":
[
"conf_matr_df_train_2
=
pd.DataFrame(confusion_matrix(y
predict(y_train_pred,
tr_thresholds,
train_fpr,
train_fpr)),
range(2),range(2))"
]
},
{
"cell_type":
"code",
"execution_count":
47,
"metadata":
{
"colab":
{},
"colab_type":

"code",
"id":
"G2v7iPsipdU4"
},
"outputs":
[
{
"data":
{
"text/plain":
[
"Text(0.5,
1.0,
'Confusion
matrix
-
Train
data
set
1')"
]
},
"execution_count":
47,
"metadata":
{},
"output_type":
"execute_result"
},
{
"data":
{
"image/png":
"iVBORw0KGgoAAAANSUhEUgAAAZQAA
"text/plain":
[
"

"
]
},
"metadata":
{},
"output_type":
"display_data"
}
],
"source":
[
"sns.set(font_scale=1.4)#f

label
size\n",
"sns.heatmap(conf_matr_df_
annot=True,annot_kws=
{\"size\":
16},
fmt='g')\n",
"\n",
"plt.xlabel(\"Actual
class\")\n",
"plt.ylabel(\"Predicted
class\")\n",
"plt.title(\"Confusion
matrix
-
Train
data
set
1\")"
]
},
{
"cell_type":
"code",
"execution_count":
48,
"metadata":
{
"colab":
{},
"colab_type":
"code",
"id":
"G2v7iPsipdU4"
},
"outputs":
[
{
"name":
"stdout",
"output_type":
"stream",
"text":
[
"Test
confusion
matrix\n",
"the
maximum
value

of
tpr*
(1-
fpr)
0.24999999161092998
for
threshold
0.506\n",
"
[[
4135
1324]\n",
"
[15518
15075]]\n"
]
}
],
"source":
[
"from
sklearn.metrics
import
confusion_matrix\n",
"print(\"Test
confusion
matrix\")\n",
"print(confusion_matrix(y_
predict(y_test_pred,
tr_thresholds,
test_fpr,
test_fpr)))"
]
},
{
"cell_type":
"code",
"execution_count":
49,
"metadata":
{
"colab":
{},
"colab_type":
"code",
"id":
"G2v7iPsipdU4"
},
"outputs":
[

{
"name":
"stdout",
"output_type":
"stream",
"text":
[
"the
maximum
value
of
tpr*
(1-
fpr)
0.24999999161092998
for
threshold
0.506\n"
]
}
],
"source":
[
"conf_matr_df_test_2
=
pd.DataFrame(confusion_mat
predict(y_test_pred,
tr_thresholds,
test_fpr,
test_fpr)),
range(2),range(2))"
]
},
{
"cell_type":
"code",
"execution_count":
50,
"metadata":
{
"colab":
{},
"colab_type":
"code",
"id":
"G2v7iPsipdU4"
},
"outputs":
[
{

"data":
{
"text/plain":
[
"Text(0.5,
1.0,
'Confusion
matrix
-
Test
data-
set
1')"
]
},
"execution_count":
50,
"metadata":
{},
"output_type":
"execute_result"
},
{
"data":
{
"image/png":
"iVBORw0KGgoAAAANSUhEUgAAA
"text/plain":
[
"

"
]
},
"metadata":
{},
"output_type":
"display_data"
}
],
"source":
[
"sns.set(font_scale=1.
label
size\n",
"sns.heatmap(conf_matr
annot=True,annot_kws=
{\"size\":
16},
fmt='g')\n",

"\n",
"plt.xlabel(\"Actual
class\")\n",
"plt.ylabel(\"Predicte
class\")\n",
"plt.title(\"Confusion
matrix
-
Test
data-
set
1\")"
]
},
{
"cell_type":
"markdown",
"metadata":
{
"colab_type":
"text",
"id":
"53C-
64FJpdU5"
},
"source":
[
"###
2.4.3
Applying
Random
Forests
on
AVG
W2V,
SET
3"
]
},
{
"cell_type":
"code",
"execution_count":
51,
"metadata":
{
"colab":
{},
"colab_type":
"code",

"id":
"TBUKDKS1pdU6"
},
"outputs":
[],
"source":
[
"X_tr=set3_train\n",
"X_te=set3_test"
]
},
{
"cell_type":
"code",
"execution_count":
202,
"metadata":
{
"colab":
{},
"colab_type":
"code",
"id":
"TBUKDKS1pdU6"
},
"outputs":
[
{
"data":
{
"text/html":
[
"
\n",
"
"
]
},
"metadata":
{},
"output_type":
"display_data"
}
],
"source":
[
"from
sklearn.ensemble
import
RandomForestClassifier

```
"import
matplotlib.pyplot
as
plt\n",
"from
sklearn.metrics
import
roc_auc_score\n",
"import
plotly.offline
as
offline\n",
"import
plotly.graph_objs
as
go\n",
"offline.init_notebook
"import
numpy
as
np\n",
"\n",
"\n",
"n_estimators_param
=
[10,
100,
500]\n",
"max_depth_param
=
[10,
50,
100,
500]\n",
"\n",
"train_results=
[]\n",
"test_results=
[]\n",
"\n",
"\n",
"\n",
"for
i
in
n_estimators_param:\n"
"
for
j
in
```

```
max_depth_param:\n",
"\n",
"
classifier
=
RandomForestClassifier
=
i,max_depth=
j,class_weight='balanc
"
classifier.fit(X_tr,
y_train)\n",
"
\n",
"\n",
"
y_train_pred
=
classifier.predict_pro
[:,1]\n",
"
false_positive_rate,
true_positive_rate,
thresholds
=
roc_curve(y_train,
y_train_pred)\n",
"
roc_auc
=
auc(false_positive_rat
true_positive_rate)\n"
"
#
Add
auc
score
to
previous
train
results\n",
"
train_results.append(r
\n",
"
\n",
"
y_test_pred
=
classifier.predict_pro
```

[:,1]\n",
"
false_positive_rate,
true_positive_rate,
thresholds
=
roc_curve(y_test,
y_test_pred)\n",
"
roc_auc
=
auc(false_positive_rat
true_positive_rate)\n"
"
#
Add
auc
score
to
previous
test
results\n",
"
test_results.append(ro
"
\n",
"
"
]
},
{
"cell_type":
"code",
"execution_count":
54,
"metadata":
{},
"outputs":
[
{
"data":
{
"text/plain":
[
"
[0.8188635987476665,\n
"
0.9995080864360953,\n"
"
0.999600989763516,\n",

"
0.9995368728199752,\n"
"
0.851583466613596,\n",
"
0.999961379920681,\n",
"
0.999965092887007,\n",
"
0.9999617387245475,\n"
"
0.8536239043061432,\n"
"
0.8543919973408305,\n"
"
0.9999950174199931,\n"
"
0.9999998728199752]"
]
},
"execution_count":
54,
"metadata":
{},
"output_type":
"execute_result"
}
],
"source":
[
"train_results"
]
},
{
"cell_type":
"code",
"execution_count":
55,
"metadata":
{},
"outputs":
[
{
"data":
{
"text/plain":
[
"
[0.6434556795450965,\n
"

0.5825652281658993,\n"
"
0.5892915554586282,\n"
"
0.5930536630139156,\n"
"
0.6672870940578144,\n"
"
0.6641017251550976,\n"
"
0.6641815869876306,\n"
"
0.6608373027682934,\n"
"
0.6690831664627703,\n"
"
0.6702492779547267,\n"
"
0.6833247302105627,\n"
"
0.6852447341475225]"
]
},
"execution_count":
55,
"metadata":
{},
"output_type":
"execute_result"
}
],
"source":
[
"test_results"
]
},
{
"cell_type":
"code",
"execution_count":
56,
"metadata":
{
"colab":
{},
"colab_type":
"code",
"id":
"TBUKDKS1pdU6"
},

"outputs":
[
{
"data":
{
"text/html":
[
"
\n",
"
"
]
},
"metadata":
{},
"output_type":
"display_data"
}
],
"source":
[
"#plotting\n",
"import
plotly.offline
as
offline\n",
"import
plotly.graph_objs
as
go\n",
"offline.init_notebook
"import
numpy
as
np\n",
"\n",
"x1=
[10,
10,
10,
10,
100,
100,
100,
100,
500,
500,
500,
500]\n",
"y1=

[10,
50,
100,
500,
10,
50,
100,
500,
10,
50,
100,
500]\n",
"z1=
train_results\n",
"z2=
test_results"
]
},
{
"cell_type":
"code",
"execution_count":
57,
"metadata":
{
"colab":
{},
"colab_type":
"code",
"id":
"TBUKDKS1pdU6"
},
"outputs":
[
{
"data":
{
"application/vnd.plotl
{
"config":
{
"linkText":
"Export
to
plot.ly",
"plotlyServerURL":
"https://plot.ly",
"responsive":
true,
"showLink":

false
},
"data":
[
{
"name":
"train",
"type":
"scatter3d",
"uid":
"3e5fade9-
90b0-
4ed0-
8779-
a69f67efa1af",
"x":
[
10,
10,
10,
10,
100,
100,
100,
100,
500,
500,
500,
500
],
"y":
[
10,
50,
100,
500,
10,
50,
100,
500,
10,
50,
100,
500
],
"z":
[
0.8188635987476665,
0.9995080864360953,
0.999600989763516,

0.9995368728199752,
0.851583466613596,
0.999961379920681,
0.999965092887007,
0.9999617387245475,
0.8536239043061432,
0.8543919973408305,
0.9999950174199931,
0.9999998728199752
]
},
{
"name":
"test
validation",
"type":
"scatter3d",
"uid":
"0ccae50c-
812a-
4fbc-
8573-
2379ec2cb606",
"x":
[
10,
10,
10,
10,
100,
100,
100,
100,
500,
500,
500,
500
],
"y":
[
10,
50,
100,
500,
10,
50,
100,
500,
10,
50,

100,
500
],
"z":
[
0.6434556795450965,
0.5825652281658993,
0.5892915554586282,
0.5930536630139156,
0.6672870940578144,
0.6641017251550976,
0.6641815869876306,
0.6608373027682934,
0.6690831664627703,
0.6702492779547267,
0.6833247302105627,
0.6852447341475225
]
}
],
"layout":
{
"scene":
{
"xaxis":
{
"title":
{
"text":
"n_estimators"
}
},
"yaxis":
{
"title":
{
"text":
"max_depth"
}
},
"zaxis":
{
"title":
{
"text":
"AUC"
}
}
}
}

},
"text/html":
[
"
\n",
"
\n",
"
\n",
"
\n",
"
\n",
"
"
]
},
"metadata":
{},
"output_type":
"display_data"
}
],
"source":
[
"#
https://plot.ly/python
axes/\n",
"trace1
=
go.Scatter3d(x=x1,y=y1
name
=
'train')\n",
"trace2
=
go.Scatter3d(x=x1,y=y1
name
=
'test
validation')\n",
"data
=
[trace1,
trace2]\n",
"\n",
"layout
=
go.Layout(scene
=

dict(\n",
"
xaxis
=
dict(title='n_estimato
"
yaxis
=
dict(title='max_depth'
"
zaxis
=
dict(title='AUC'),))\n
"\n",
"fig
=
go.Figure(data=data,
layout=layout)\n",
"offline.iplot(fig,
filename='set3')"
]
},
{
"cell_type":
"markdown",
"metadata":
{
"colab":
{},
"colab_type":
"code",
"id":
"TBUKDKS1pdU6"
},
"source":
[
"#we
could
see
that
hyperparameters
are
n_estimators
as
100
,max_depth
as
10
but
to

cross
check
we
will
again
do
hyperparameter
tuning
for
depth
and
estimators\n"
]
},
{
"cell_type":
"code",
"execution_count":
58,
"metadata":
{},
"outputs":
[
{
"data":
{
"text/html":
[
"
\n",
"
"
]
},
"metadata":
{},
"output_type":
"display_data"
},
{
"data":
{
"image/png":
"iVBORw0KGgoAAAANSUhEU
"text/plain":
[
"
"
]

},
"metadata":
{},
"output_type":
"display_data"
}
],
"source":
[
"#hyperprameter
cross
checking
again
for
max
depth\n",
"from
sklearn.ensemble
import
RandomForestClassi
"import
matplotlib.pyplot
as
plt\n",
"from
sklearn.metrics
import
roc_auc_score\n",
"import
plotly.offline
as
offline\n",
"import
plotly.graph_objs
as
go\n",
"offline.init_note
"import
numpy
as
np\n",
"from
sklearn.metrics
import
roc_curve,
auc\n",
"\n",
"max_depth_param=
[2,3,4,5,6,7,8,9,1
"\n",

```
"train_results1=
[]\n",
"test_results1=
[]\n",
"\n",
"\n",
"\n",
"for
i
in
max_depth_param:\n
"
classifier
=
RandomForestClassi
=
100,max_depth=
i,class_weight='ba
"
classifier.fit(X_t
y_train)\n",
"
\n",
"\n",
"
y_train_pred
=
classifier.predict
[:,1]\n",
"
false_positive_rat
true_positive_rate
thresholds
=
roc_curve(y_train,
y_train_pred)\n",
"
roc_auc
=
auc(false_positive
true_positive_rate
"
#
Add
auc
score
to
previous
train
results\n",
```

```
"
train_results1.app
\n",
"
\n",
"
y_test_pred
=
classifier.predict
[:,1]\n",
"
false_positive_rat
true_positive_rate
thresholds
=
roc_curve(y_test,
y_test_pred)\n",
"
roc_auc
=
auc(false_positive
true_positive_rate
"
#
Add
auc
score
to
previous
test
results\n",
"
test_results1.appe
"
\n",
"\n",
"from
matplotlib.legend_
import
HandlerLine2D\n",
"line1,
=
plt.plot(max_depth
train_results1,
'b',
label=\"Train
AUC\")\n",
"line2,
=
plt.plot(max_depth
```

test_results1,
'r',
label=\"Test
AUC\")\n",
"\n",
"plt.legend(handle
{line1:
HandlerLine2D(nump
"plt.ylabel(\"AUC
score\")\n",
"plt.xlabel(\"max_
"plt.show()
"
]
},
{
"cell_type":
"markdown",
"metadata":
{},
"source":
[
"#as
dept
increase
s
we
could
see
model
is
overfitting
.therefore
considering
6
as
the
max
depth"
]
},
{
"cell_type":
"code",
"execution_count":
59,
"metadata":
{},
"outputs":
[

{
"data":
{
"text/html":
[
"
\n",
"
"
]
},
"metadata":
{},
"output_type":
"display_data"
},
{
"data":
{
"image/png":
"iVBORw0KGgoAAAANS
"text/plain":
[
"

"
]
},
"metadata":
{},
"output_type":
"display_data'
}
],
"source":
[
"#hyperpramete
cross
checking
again
for
n
estimators\n",
"from
sklearn.ensemb
import
RandomForestCl
"import
matplotlib.pyp
as

```
plt\n",
"from
sklearn.metric
import
roc_auc_score\
"import
plotly.offline
as
offline\n",
"import
plotly.graph_c
as
go\n",
"offline.init_
"import
numpy
as
np\n",
"\n",
"\n",
"n_estimators_
[5,10,50,100,2
"\n",
"train_results
[]\n",
"test_results2
[]\n",
"\n",
"\n",
"\n",
"for
i
in
n_estimators_p
"
classifier
=
RandomForestCl
=
i,max_depth=
6,class_weight
"
classifier.fit
y_train)\n",
"
\n",
"\n",
"
y_train_pred
=
```

```
classifier.pre
[:,1]\n",
"
false_positive
true_positive_
thresholds
=
roc_curve(y_tr
y_train_pred)\
"
roc_auc
=
auc(false_posi
true_positive_
"
#
Add
auc
score
to
previous
train
results\n",
"
train_results2
\n",
"
\n",
"
y_test_pred
=
classifier.pre
[:,1]\n",
"
false_positive
true_positive_
thresholds
=
roc_curve(y_te
y_test_pred)\r
"
roc_auc
=
auc(false_posi
true_positive_
"
#
Add
auc
score
```

to
previous
test
results\n",
"
test_results2.
"
\n",
"from
matplotlib.leg
import
HandlerLine2D\
"line1,
=
plt.plot(n_est
train_results2
'b',
label=\"Train
AUC\")\n",
"line2,
=
plt.plot(n_est
test_results2,
'r',
label=\"Test
AUC\")\n",
"\n",
"plt.legend(ha
{line1:
HandlerLine2D(
"plt.ylabel(\'
score\")\n",
"plt.xlabel(\'
estimators\")\
"plt.show()
"
]
},
{
"cell_type":
"markdown",
"metadata":
{},
"source":
[
"#we
could
see
that
hyperparameter

are
n_estimators=1
,max_depth=6"
]
},
{
"cell_type":
"code",
"execution_cou
60,
"metadata":
{
"colab":
{},
"colab_type":
"code",
"id":
"TBUKDKS1pdU6'
},
"outputs":
[
{
"data":
{
"image/png":
"iVBORw0KGgoA/
"text/plain":
[
"

"
]
},
"metadata'
{},
"output_ty
"display_(
}
],
"source":
[
"#
finding
the
train
and
test
AU\n",
"from
sklearn.er

```
import
RandomFore
"from
sklearn.me
import
roc_curve,
auc\n",
"model
=
RandomFore
=
150,
max_depth=
class_weig
n_jobs=-1)
"model.fit
y_train)\n
"\n",
"#
roc_auc_sc
y_score)
the
2nd
parameter
should
be
probabilit
estimates
of
the
positive
class\n",
"#
not
the
predicted
outputs\n"
"\n",
"y_train_p
=
model.pred
[:,1]
\n",
"y_test_pr
=
model.pred
[:,1]\n",
"\n",
"\n",
"train_fpr
```

train_tpr,
tr_thresho
=
roc_curve(
y_train_pr
"test_fpr,
test_tpr,
te_thresho
=
roc_curve(
y_test_pre
"\n",
"plt.plot(
train_tpr,
label=\"Tr
AUC
=\"+str(au
train_tpr)
"plt.plot(
test_tpr,
label=\"Te
AUC
=\"+str(au
test_tpr))
"plt.legen
"plt.xlabe
Positive
Rate(TPR)\
"plt.ylabe
Positive
Rate(FPR)\
"plt.title
AUC\")\n",
"plt.grid(
"plt.show(
]
},
{
"cell_type
"code",
"execution
61,
"metadata"
{
"colab":
{},
"colab_typ
"code",
"id":
"TBUKDKS1p

},
"outputs"
[],
"source":
[
"def
predict(pr
threshoulc
fpr,
tpr):\n",
"
\n",
"
t
=
threshoulc
(1-
tpr))]\n",
"
\n",
"
#
(tpr*
(1-
fpr))
will
be
maximum
if
your
fpr
is
very
low
and
tpr
is
very
high\n",
"
\n",
"
print(\"th
maximum
value
of
tpr*
(1-
fpr)\",
max(tpr*

(1-
fpr)),
\"for
threshold\
np.round(t
"
predictior
=
[]\n",
"
for
i
in
proba:\n",
"
if
i>=t:\n",
"
predictior
"
else:\n",
"
predictior
"
return
predictior
]
},
{
"cell_type
"code",
"executior
62,
"metadata"
{
"colab":
{},
"colab_typ
"code",
"id":
"TBUKDKS1p
},
"outputs"
[
{
"name":
"stdout",
"output_ty
"stream",
"text":

```
[
"Train
confusion
matrix\n",
"the
maximum
value
of
tpr*
(1-
fpr)
0.2499999
for
threshold
0.477\n",
"
[[
5543
5540]\n",
"
[13600
48513]]\n"
]
}
],
"source":
[
"from
sklearn.me
import
confusion_
"print(\"
confusion
matrix\")
"print(con
predict(y_
tr_thresho
train_fpr,
train_fpr)
]
},
{
"cell_type
"code",
"executior
63,
"metadata"
{
"colab":
{},
```

"colab_typ
"code",
"id":
"TBUKDKS1p
},
"outputs"
[
{
"name":
"stdout",
"output_ty
"stream",
"text":
[
"the
maximum
value
of
tpr*
(1-
fpr)
0.2499999
for
threshold
0.477\n"
]
}
],
"source":
[
"conf_matr
=
pd.DataFra
predict(y_
tr_thresho
train_fpr,
train_fpr)
range(2),r
]
},
{
"cell_type
"code",
"executior
64,
"metadata"
{
"colab":
{},
"colab_typ

"code",
"id":
"TBUKDKS1p
},
"outputs"
[
{
"data":
{
"text/pla
[
"Text(0.5,
1.0,
'Confusion
matrix
-
Train
data
set
3')"
]
},
"execution
64,
"metadata"
{},
"output_ty
"execute_r
},
{
"data":
{
"image/png
"iVBORw0KG
"text/pla
[
"
"
]
},
"meta
{},
"outp
"disp
}
],
"sour
[
"sns.

label
size\r
"sns.l
annot=
{\"si
16},
fmt='
"\n",
"plt.x
class
"plt.y
class
"plt.t
matri
-
Train
data
set
3\")"
]
},
{
"cell_
"code
"execu
65,
"meta
{
"colab
{},
"colab
"code
"id":
"TBUK
},
"outpu
[
{
"name
"stdo
"outpu
"strea
"text
[
"Test
confu
matri
"the
maxim
value

of
tpr*
(1-
fpr)
0.249
for
thresh
0.53\n
"
[[
3956
1503]]
"
[1549
15102
]
}
],
"sourc
[
"from
sklea
impor
confu
"prin
confu
matri
"prin
predi
tr_th
test_
test_
]
},
{
"cell_
"code
"execu
66,
"meta
{
"cola
{},
"cola
"code
"id":
"TBUK
},
"outp
[

{
"name
"stdou
"outpu
"strea
"text
[
"the
maximu
value
of
tpr*
(1-
fpr)
0.249
for
thresh
0.53\n
]
}
],
"sourc
[
"conf_
=
pd.Dat
predic
tr_thr
test_
test_
range
]
},
{
"cell_
"code
"execu
67,
"metad
{
"colab
{},
"colab
"code
"id":
"TBUKI
},
"outpu
[
{

"data
{
"text/
[
"Text
1.0,
'Conf
matri
-
Test
data-
set
3')"
]
},
"exec
67,
"meta
{},
"outp
"exec
},
{
"data
{
"imag
"iVBO
"text/
[
"
"
]
}
"
{
"
"
}
]
"
[
"
l
s
"
a
{
1
f

"
"
c
"
c
"
m
-
T
d
s
3
]
}
{
"
"
"
{
"
"
"
"
}
"
[
"
2
A
R
F
o
T
W
S
4
]
}
{
"
"
"
6
"
{
"
{
"
"
"

"
}
"
[
"
[
"
"
]
}
{
"
"
"
2
"
{
"
{
"
"
"
"
}
"
[
{
"
{
"
[
"
\
"
"
]
}
"
{
"
"
}
]
"
[
"
s
i
R
"

m
a
p
"
s
i
r
"
p
a
o
"
p
a
g
"
"
n
a
n
"
"
=
[
1
5
"
=
[
5
1
5
"
"
[
"
[
"
"
i
i
n
"
f
j
i
m
"
"
c

=
R
=
i
j
"
c
y
"
\
"
"
y
=
c
[
"
f
t
t
=
r
y
"
r
=
a
t
"
#
A
a
s
t
p
t
r
"
t
\
"
\
"
y
=
c
[
"
f
t

t
=
r
y
"
r
=
a
t
"
#
A
a
s
t
p
t
r
"
t
"
\
"
"
]
}
{
"
"
"
7
"
{
"
{
"
"
"
"
}
"
[
{
"
{
"
[
"
[
"

0
"
0
"
0
"
0
"
0
"
0
"
0
"
0
"
0
"
0
"
0
]
}
"
7
"
{
"
"
}
]
"
[
"
]
}
{
"
"
"
7
"
{
"
{
"
"
"
"
}

"
[
{
"
{
"
[
"
[
"
0
"
0
"
0
"
0
"
0
"
0
"
0
"
0
"
0
"
0
"
0
]
}
"
7
"
{
"
"
}
]
"
[
"
]
}
{
"
"
"

7
"
{
"
{
"
"
"
"
}
"
[
{
"
{
"
[
"
\
"
"
]
}
"
{
"
"
}
]
"
[
"
"
p
a
o
"
p
a
g
"
"
n
a
n
"
"
[
1
1

1
1
1
1
1
5
5
5
5
"
[
5
1
5
1
5
1
5
1
5
1
5
"
t
"
t
]
}
{
"
"
"
7
"
{
"
{
"
"
"
"
}
"
[
{
"
{
"
{
"

{
"
"
t
p
"
"
"
t
"
f
}
"
[
{
"
"
"
"
"
"
0
4
a
b
"
[
1
1
1
1
1
1
1
1
5
5
5
5
]
"
[
1
5
1
5
1
5
1
5

1
5
1
5
]
"
[
0
0
0
0
0
0
0
0
0
0
0
0
]
}
{
"
"
v
"
"
"
"
9
4
a
b
"
[
1
1
1
1
1
1
1
1
5
5
5
5
]
"
[

1
5
1
5
1
5
1
5
1
5
1
5
]
"
[
0
0
0
0
0
0
0
0
0
0
0
0
]
}
]
"
{
"
{
"
{
"
{
"
"
}
}
"
{
"
{
"
"
}
}

"
{
"
{
"
"
}
}
}
}
}
"
[
"
\
"
\
"
\
"
\
"
\
"
"
]
}
"
{
"
"
}
]
"
[
"
h
a
"
=
g
n
=
'
"
=
g
n
=
'

v
"
=
[
t
"
"
=
g
=
d
"
x
=
d
"
y
=
d
"
z
=
d
"
"
=
g
l
"
f
]
}
{
"
"
"
7
"
{
"
[
{
"
{
"
[
"
\
"
"

]
}
"
{
"
"
}
{
"
{
"
"
"
[
"