DonorsChoose

DonorsChoose.org receives hundreds of thousands of project proposals each year for classroom projects in need of donations. Page visits, donations, and essays written by students are recorded in textually-rich event submission tables for organizers to prepare for the DonorsChoose.org website.

Here your *Donorsch* Data.org mission is to narrow down to 500,000 project proposals, for a start, there are three main problems they would like to solve:

- How to use current textual processed and resources to screen 500,000 projects so that they can be posted as quickly and as accurately as possible?
- How to develop the competitive scoring system and effective calculation to measure the impactiveness of donations
- How to ensure that the text features generated from the essays are robust and meaningful over the 500,000 projects

The goal of the competition is to provide competitive and robust machine learning models for a project recommendation system to ensure that more students receive the materials they need to succeed in the classroom and school. DonorsChoose.org hopes that the insights from the competition proposals would be used to improve their existing recommendation system.

About the DonorsChoose Data Set

The `projects.csv` data set provided by DonorsChoose contains the following features:

Feature	Description
`projectid`	A unique identifier for the proposed project. Example: `ca2cd086ada9bd6fb8b6bb267f0f891e`
`teacher_acctid`	The id for a teacher proposing a project. Example: `c90749f5d961ff158a67a3dd1fb6f605`
`schoolid`	School identifier for the school associated with the project. Example: `a9bf2a4a5ed7c0a0ad0f687f68f07450`
`school_ncesid`	Good source to extend the school data with additional features. Example: `000003000000`
`school_latitude`	School latitude. Example: `34.052835`
`school_longitude`	School longitude. Example: `-118.243685`
`school_city`	Text description of the school city. Example: `Los Angeles`
`school_state`	Two-letter USPS postal abbreviation for the school state. Example: `CA`
`school_zip`	Primary five-digit zipcode of the school. Example: `90033`
`school_metro`	Text description of the school metropolitan area. Example: `urban` or `suburban` or `rural` or `unknown`
`school_district`	Text description of the school district. Example: `Los Angeles Unified School District`
`teacher_prefix`	Teacher's title. Example: `Ms.` or `Mr.` or `Mrs.` or `Teacher`
`primary_focus_subject`	Text description of a subject area for which the project is most focused. Example: `Literature & Writing, Social Sciences`
`primary_focus_area`	Text description of an area for which the project is most focused. Example: `Literacy & Language`
`secondary_focus_subject`	Text description of a secondary subject area for which the project is focused. Example: `Mathematics`
`secondary_focus_area`	Text description of a secondary area for which the project is focused. Example: `History & Civics`
`resource_type`	Text description of the primary resource requested for purchase. Example: `Technology`
`poverty_level`	Text description of the poverty level of the school associated with the project. Example: `high poverty`
`grade_level`	Text description of the grade level associated with the project. Example: `Grades PreK-2`
`vendor_zipcode`	Primary five-digit zipcode associated with the vendor. Example: `60657`
`vendor_name`	Text description of the vendor associated with the project. Example: `Lakeshore Learning Materials`
`number_of_previously_posted_projects`	Number of school applicants previously submitted by the same teacher. Example: `2`

Take the section **Notes on the Essay Data** for more details about these features.

Additionally, the following files contain other information about the resources requested for each project. Each item in this file represents a resource requested for a project:

File	Description
`resources.csv`	A project id, an description, quantity, and price for each resource requested.

Each row represents a resource requested for a project, identified by a `projectid`.

Label	Description
`teacher_number_of_previously_posted_projects`	Number of previously posted projects by the same teacher. Example: `0`

```
# For every project in resources.csv, summarize quantity and price
# Merge resources.csv summaries into projects.csv
# Fill missing values with 0
# Print the head of the new projects.csv
!head -n 1 /kaggle/input/donorschoose-application-screening/projects.csv > /kaggle/working/projects.csv
!tail -n +2 /kaggle/input/donorschoose-application-screening/projects.csv | \
awk -F,'{key=$1; q[key]+=$7; p[key]+=$8; print}' \
/kaggle/input/donorschoose-application-screening/resources.csv - | \
awk -F,'{key=$1; if (q[key]>0 || p[key]>0) {q[key]=0; p[key]=0}; print}' \
>> /kaggle/working/projects.csv
```

```
projectid,teacher_acctid,schoolid,school_ncesid,school_latitude,school_longitude,school_city,school_state,school_zip,school_metro,school_district,teacher_prefix,primary_focus_subject,primary_focus_area,secondary_focus_subject,secondary_focus_area,resource_type,poverty_level,grade_level,vendor_zipcode,vendor_name,number_of_previously_posted_projects,total_price_excluding_optional_support,total_price_including_optional_support,students_reached
```

1.1 Reading Data

```
# Load the data
import pandas as pd, numpy as np, matplotlib.pyplot as plt, seaborn as sns
projects = pd.read_csv('/kaggle/working/projects.csv')
print("Projects shape:", projects.shape)
projects.sample(3)
```

```
Projects shape: (1110022, 20)
```

	projectid	teacher_acctid	schoolid	school_ncesid	school_latitude	school_longitude	school_city	school_state	school_zip	school_metro	school_district	teacher_prefix	primary_focus_subject	primary_focus_area	secondary_focus_subject	secondary_focus_area	resource_type	poverty_level	grade_level	vendor_zipcode	vendor_name	number_of_previously_posted_projects	total_price_excluding_optional_support	total_price_including_optional_support	students_reached
1110019	ffffb2ef6e0d0f0f0f0f0f0f0f0f0f0f	0000000000000000000000000000000000	0000000000000000000000000000000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Mrs.	NaN	NaN	NaN	NaN	NaN	NaN	Grades PreK-2	NaN	NaN	0	0	0	NaN
1110020	ffffb2ef6e0d0f0f0f0f0f0f0f0f0f0f	0000000000000000000000000000000000	0000000000000000000000000000000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Mrs.	NaN	NaN	NaN	NaN	NaN	NaN	Grades PreK-2	NaN	NaN	0	0	0	NaN
1110021	ffffb2ef6e0d0f0f0f0f0f0f0f0f0f0f	0000000000000000000000000000000000	0000000000000000000000000000000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Mrs.	NaN	NaN	NaN	NaN	NaN	NaN	Grades PreK-2	NaN	NaN	0	0	0	NaN

```
# Load resources.csv
resources = pd.read_csv('/kaggle/input/donorschoose-application-screening/resources.csv')
print("Resources shape:", resources.shape)
resources.sample(3)
```

```
Resources shape: (1317488, 4)
```

	projectid	description	quantity	price
1317486	ffffb2ef6e0d0f0f0f0f0f0f0f0f0f0f	Wooden Boxes for Classroom Supplies	4	14.85
1317487	ffffb2ef6e0d0f0f0f0f0f0f0f0f0f0f	Wooden Boxes for Classroom Supplies	4	14.85
1317488	ffffb2ef6e0d0f0f0f0f0f0f0f0f0f0f	Wooden Boxes for Classroom Supplies	4	14.85

1.2 preprocessing of project_subject_categories

```
# Preprocessing of project_subject_categories
# https://www.kaggle.com/shtechgirl/donorschoose-eda-for-beginners#Text-Preprocessing
# https://www.kaggle.com/quickdraw2018/donorschoose-eda-a-beginner-s-perspective#Text-Preprocessing
# Text preprocessing
catogories = list(projects['project_subject_categories'])
# remove special character from list
for i, value in enumerate(catogories):
    catogories[i] = value.replace("The ", "").replace("&", "").replace(",", "").replace(" ", "")
cat_list = []
for i in catogories:
    temp = ""
    # consider we have text like "Math & Science, History & Civics, Health & Sports"
    j = i.split(',')
    for k in j:
        if 'Warmth' in k:
        # print("Warmth")
        pass
        else:
        if 'The' in k:
        temp+=k.replace("The","")+" "
        else:
        temp+=k+" "
    cat_list.append(temp.strip())
projects['clean_categories'] = cat_list
projects.drop(['project_subject_categories'], axis=1, inplace=True)
# Count of all the words in corpus python: https://stackoverflow.com/a/515956/4084039
from collections import Counter
my_counter = Counter()
for word in projects['clean_categories'].values:
    for word in str(word).split():
        my_counter.update(word.replace(',',''))
cat_dictionary = dict(my_counter)
sorted_cat_dictionary = {k: v for k, v in sorted(cat_dictionary.items(), key=lambda item: -item[1])}
```

1.3 Text preprocessing

```
# Text preprocessing
stopwords = set('for', 'this', 'also', 'with', 'the', 'a', 'in', 'from', 'for', 'to', 'and', 'the', 'of', 'a', 'an', 'in', 'for', 'on', 'by', 'with', 'is', 'that', 'this', 'these', 'those', 'you', 'your', 'we', 'our', 'us', 'i', 'me', 'my', 'mine', 'he', 'she', 'it', 'its', 'they', 'them', 'their', 'theirs', 'what', 'which', 'who', 'whom', 'whose', 'be', 'am', 'is', 'are', 'was', 'were', 'being', 'been', 'have', 'has', 'had', 'do', 'does', 'did', 'will', 'would', 'shall', 'should', 'may', 'might', 'must', 'can', 'could', 'not', 'no', 'nor', 'but', 'if', 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', 'should', 'now')
```

```
# Combining all the above stundents
from tqdm import tqdm
preprocessed_essays = []
# tqdm for for loop
for sentance in tqdm(projects['essay_1']):
    sentance = sentance.replace('\\r', ' ')
    sentance = sentance.replace('\\"', ' ')
    sentance = sentance.replace('\\n', ' ')
    sentance = re.sub('[^A-Za-z0-9]+', ' ', sentance)
    # https://gist.github.com/sebleier/554280
    sentance = ' '.join(e for e in sentance.split() if e not in stopwords)
    preprocessed_essays.append(sentance.lower())
```

```
100%|□□□
```

```
In [2]: df.groupby(df.order_status).count().order_id
Out[2]: order_status
        Canceled     141
        Delivered   9820
        Invoiced     314
        Processing   301
        Shipped     1107
        Unavailable    6
        Unknown        9
        Name: order_id, dtype: int64

In [3]: df.groupby(df.order_status).count().order_id.plot.bar();
```

```
In [4]: df[df.order_status == 'Unknown']
Out[4]:                 order_id  customer_id order_status order_purchase_timestamp order_approved_at  \
        2605  5a8dd3ad9de8d483a...  f3c38ab652836d21d...      Unknown                NaN               NaT
        6728  e481f51cbdc54678b...  7c396fd4830fd04220...      Unknown                NaN               NaT
        6829  dd7ddc04e1b6c2c1a...  eb7f5f2d8d68a07cb...      Unknown                NaN               NaT
        8742  5ef8c82a62f5a4038...  3d7a9d0f94d3f0ed0...      Unknown                NaN               NaT
        9271  8a076d0f2a354d7ad...  ec05a6d2f4f2f9e2e...      Unknown                NaN               NaT
        9670  2c67e779ed9f0b9f1...  ad21c59310ef5f9d8...      Unknown                NaN               NaT
        9848  0f5ac8d7f9d94cb8d...  a8ec6a3d5d0d0f2f2...      Unknown                NaN               NaT
        9849  0008288aa423d2a3f...  7d235b4d60b49c2a2...      Unknown                NaN               NaT
        9850  bb8ef542450f37147...  8dd2d8f2ce7bb0f58...      Unknown                NaN               NaT

              order_delivered_carrier_date  order_delivered_customer_date  \
        2605                           NaT                            NaT
        6728                           NaT                            NaT
        6829                           NaT                            NaT
        8742                           NaT                            NaT
        9271                           NaT                            NaT
        9670                           NaT                            NaT
        9848                           NaT                            NaT
        9849                           NaT                            NaT
        9850                           NaT                            NaT

              order_estimated_delivery_date  order_status_code  \
        2605                            NaT                  0
        6728                            NaT                  0
        6829                            NaT                  0
        8742                            NaT                  0
        9271                            NaT                  0
        9670                            NaT                  0
        9848                            NaT                  0
        9849                            NaT                  0
        9850                            NaT                  0

        [9 rows x 10 columns]
```

Assignment 3: Apply KNN

Step 1: Apply features from exercise three and two

- Tip1: categorical_numerical_features = project_df[df[df.columns].select_dtype(object)]
- Tip2: categorical_numerical_features = project_df[df[df.columns].select_dtype(int64)]
- Tip3: categorical_numerical_features = project_df[df[df.columns].select_dtype(float64)]
- Tip4: categorical_numerical_features = project_df[df[df.columns].select_dtype(object|int64|float64)]

Step 2: Apply KNN function from exercise two

- Tip1 Use the function provided in the exercise two (Assignment 2.1) (slide)
- Tip2 Find the best hyper-parameter using train-cross validation (on whole cross validation data)
- Tip3 Use the best hyper-parameter and evaluate the model on the test data (hold-out data)

Step 3: Recommendation system

- You need to pick the performance of model both on train data and cross validation data for each hyper-parameter, as shown in the figure.
- Find the best hyper-parameter using the cross validation data (as shown in the figure)
- Find the best hyper-parameter using train-cross validation (on whole cross validation data)
- Find the optimal decision boundary (threshold) on cross validation data and apply the prediction (discussed in class, not in the pdf)

Step 4: Extra (2)

- Select top 2000 features from this notebook (select_top_features_2000) and then apply KNN on top of these features
- Tip1 Instead of using all features, use top 2000 most frequent words (TF) (select_top_features_2000)
- Tip2 Instead of using all features, use top 2000 most informative features by using "sklearn.feature_selection" and then apply KNN on top of these features
- Tip3 Instead of using all features, use top 2000 most informative features by using "TfIdfVectorizer" and then apply KNN on top of these features

Step 5: Extra (3)

- Instead of using cross validation, use 5-fold cross validation for finding the best hyper-parameter, shown in the figure
- After finding the best hyper-parameter, combine train and cross validation data (as shown in the figure) and then evaluate the model performance on the test data
- Discuss why 5-fold cross validation is better than 2-fold cross validation or using hold-out data

Step 6: Extra (4)

- Instead of using 5-fold cross validation, use 10-fold cross validation
- Find the best hyper-parameter using train-cross validation (as shown in the figure), and then evaluate the model performance on the test data
- Discuss why 10-fold cross validation is better than 5-fold cross validation or 2-fold cross validation, or why they are the same

Step 7: Extra (5)

- Instead of using 5-fold cross validation, use 10-fold cross validation
- Find the best hyper-parameter using train-cross validation (as shown in the figure), and then evaluate the model performance on the test data
- Discuss why 10-fold cross validation is better than 5-fold cross validation or 2-fold cross validation, or why they are the same

Step 8: Extra (6)

- Instead of using cross validation, use leave-one-out cross validation
- Find the best hyper-parameter using train-cross validation (as shown in the figure), and then evaluate the model performance on the test data
- Discuss why leave-one-out cross validation is better than 10-fold cross validation or 5-fold cross validation, or why they are the same

2. K Nearest Neighbor

2.1 Splitting data into Train and cross validation(or test): Stratified Sampling

```
In [5]: # Stratified sampling, 70% data should be train data, and 30% data should be cross validation
        # this is time and memory consuming operation, so be patient, it will take 30 seconds on a 2.5GHz machine
        # first figure out what is the sum that about 30% of data is cross validation
        # then randomly select that many points from the data and make them as cross validation
        # after we split the data into train and cross validation, we save them in disk
        # so that we can use them in future, so that we don't do this expensive step in the future
        # n_samples, n_features
        # X_train.shape, X_cv.shape
        # y_train.shape, y_cv.shape

        y_project_data[y_project_data.shape[0] - 10]
        # we are using random state to make results reproducible
        # stratify parameter is used to make sure that the proportion of class labels in the train and cross validation data is same as that in the original data
```

In [24]: # train model and save and plot acc/loss/auc/prec/recall
from keras.models import Sequential, load_model, TrainHistory[cr]

```
# fix random seed for reproducibility
numpy.random.seed(7)

print('X_train shape: %s, y_train shape:
%s'%(X_train.shape, y_train.shape))
print('X_test shape: %s, y_test shape:
%s'%(X_test.shape, y_test.shape))

print("Training")
# initialize the keras model
model = Sequential(cnn_layers)
metrics=['acc','fmeasure','precision','recall']

# we use the "binary_crossentropy" loss and the adam optimizer
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=metrics)
# fit the model, store the training history in variable "history"
history = model.fit(X_train, y_train, nb_epoch=nb_epoch, batch_size=batch_size,
verbose=1, validation_data=(X_test, y_test))

print("Evaluating")
# evaluate the model against the test set
scores = model.evaluate(X_test, y_test, batch_size=batch_size, verbose=1)
print("%s: %.2f%%"%(model.metrics_names[1], scores[1]*100))
print("%s: %.2f%%"%(model.metrics_names[2], scores[2]*100))
print("%s: %.2f%%"%(model.metrics_names[3], scores[3]*100))
print("%s: %.2f%%"%(model.metrics_names[4], scores[4]*100))

# save model
model.save('models/cnn.h5')

# plot acc/loss/auc/prec/recall vs epoch
import matplotlib.pyplot as plt
plt.rcParams['figure.figure_size'] = (15, 5)
plt.rcParams['axes.grid'] = True
plt.rcParams['axes.spines.top'] = False
plt.rcParams['axes.spines.right'] = False
plt.rcParams['axes.spines.bottom'] = False
plt.rcParams['axes.spines.left'] = False
plt.rcParams['axes.ymargin'] = 0.1
plt.rcParams['axes.xmargin'] = 0.1
plt.rcParams['axes.color_cycle'] = ('1f77b4', 'ff7f0e', '2ca02c', 'd62728',
'9467bd', '8c564b', 'e377c2', '7f7f7f', 'bcbd22', '17becf')
plt.rcParams['lines.linewidth'] = 2.0
plt.rcParams['font.family'] = 'monospace'
plt.rcParams['font.size'] = 10.0
plt.rcParams['axes.labelsize'] = 'medium'
plt.rcParams['axes.titlesize'] = 'medium'
plt.rcParams['legend.fontsize'] = 'medium'
plt.rcParams['legend.frameon'] = False
plt.rcParams['legend.numpoints'] = 1
plt.rcParams['legend.scatterpoints'] = 1
plt.rcParams['legend.fancybox'] = True
plt.rcParams['legend.shadow'] = True
plt.rcParams['legend.markerscale'] = 1.0
plt.rcParams['legend.borderaxespad'] = 0.5
plt.rcParams['legend.borderpad'] = 0.4
plt.rcParams['legend.labelsep'] = 0.02
plt.rcParams['legend.handlelength'] = 2.0
plt.rcParams['legend.handleheight'] = 0.7
plt.rcParams['legend.handletextpad'] = 0.8
plt.rcParams['legend.ncol'] = 1
plt.rcParams['legend.loc'] = 'best'
plt.rcParams['legend.mode'] = None
plt.rcParams['legend.bbox_to_anchor'] = None
plt.rcParams['legend.bbox_transform'] = None
plt.rcParams['legend.bbox_clip_on'] = False
plt.rcParams['legend.bbox_clip_path'] = None
plt.rcParams['legend.bbox_inches'] = None
plt.rcParams['legend.bbox_pad_inches'] = None
plt.rcParams['legend.bbox_extra_artists'] = None
plt.rcParams['legend.bbox_extra_text'] = None
plt.rcParams['legend.bbox_text_clip_on'] = False
plt.rcParams['legend.bbox_text_clip_path'] = None
plt.rcParams['legend.bbox_text_inches'] = None
plt.rcParams['legend.bbox_text_pad_inches'] = None
plt.rcParams['legend.bbox_text_extra_artists'] = None
plt.rcParams['legend.bbox_text
```

```
# Average Weighted
# Average Weighted Ensemble for Deep Learning
# [0.9, 0.1] Average Weighted Ensemble for best ensemble/stacking as stated in this list
# [0.5, 0.5] Average Weighted Ensemble for good ensemble and of ens single
# [0.25, 0.75] Average Weighted Ensemble for of ens single
# [0.1, 0.9] Average Weighted Ensemble for worst ensemble/stacking
# [0.0, 1.0] Average Weighted Ensemble for worst single
# [0.0, 0.0, 1.0] Average Weighted Ensemble for worst ens and single
# [0.0, 0.0, 0.0, 1.0] Average Weighted Ensemble for worst ens and two of single
# Note we are evaluating for accuracy(correct/wrong) and the of single/ensemble choice
# [0.25,0.25,0.25,0.25] Average Weighted Ensemble
# [0.0,0.0,0.0,0.0] Average Weighted Ensemble (uniform) (average) (mean) (equal) (equally) (equally the)
# [0.0,0.0,0.0,0.0] Average Weighted Ensemble
```