UCLA Dropout Prediction Documentation

0. **Summary of Pipeline**
1. **Creating Unique Key Table**
    a. Input: cleaned_student_data.csv
    b. Script: create_uniqe_key.py
    c. Result: uniq_key_student_data.csv
2. **Creating Feature Table**
    a. Input: uniq_key_student_data.csv + create_feature_table.yml
    b. Script: create_feature_table.py + feature_computation.py
    c. Result: feature_table.csv
3. **Creating Labeled Feature Table**
    a. Input: feature_table.csv + extra_data.csv
    b. Script: attach_labels.py
    c. Result: attach_labels.s
4. **Create Model Outputs**
    a. Input: feature_table.csv + machine_learning.yml
    b. Script: generate_predictions.py
    c. Result: pickle (.p) files in ../model_output/ + generate_predictions.s
5. **Evaluate Models**
    a. Input: (.p) files in ../model_output/ + evaluation.yml
    b. Script: evaluation.py
    c. Result: model_scores.csv + Image (.png) files in ../model_plots/

0. Summary of Pipeline

The pipeline is structured as follows:

a. Start with Cleaned Student Data
b. Attach unique labels to cleaned student data
c. Create various features using cleaned student data and some extra data
d. Attach labels we want to predict to the feature table
e. Perform various machine learning algorithms on the feature table and store results
f. Calculate various evaluation metrics and generate visuals for the model results
g. Change parameters as needed and repeat

1. Creating Unique Key Table

One issue with the original cleaned student data is that there is no unique identifier for each row. We simply calculate one by using a hash of the student id, term, and course, which is a (theoretically) unique combination. We add this hash as a column to each row.

a. We use as input the **cleaned_student_data.csv** file
b. The script that performs this task is called **create_uniqe_key.py**
c. We know this task is done if its output **uniq_key_student_data.csv** exists in the current working directory. If it exists, do not run this task. If it does not exist, run this task and all following it.

2. Creating Feature Table

This task creates a csv which has as each column a feature regarding each (student, term) combination in our dataset. A feature is loosely defined as any predictor of whether or not a student drops out of their major.

a. We use as input the **uniq_key_stuent_data.csv** file and the **create_feature_table.yml** configuration file. The various options of the latter file are described below
b. The script that performs this task is called **create_feature_table.py**
c. We know this task is done if the feature table, **feature_table.csv** is younger than **create_feature_table.yml**. If this is the case, then the user modified the feature table parameters and then already created a feature table based on these parameters and so we do not need to run this task. If, on the other hand, **create_feature_table.yml** is younger, the use wants to modify the feature table so we need to run this task.

create_feature_table.yml options:

This file contains the information about each feature we want to generate. For each feature we want to add, we need to do two things. First, we need to add information about the feature to **create_feature_table.yml** and then we need to add a function to **feature_computation.py** specifying the code to generate the feature.

An entry for a feature in **create_feature_table.yml** looks like this:

```
---
name: running_gpa
deps: ID, alph_term, grade
incl: True
coll: True
type: cts
---
```

name: the name of your feature in the final feature_table. Note also that when you add a function to **feature_computation.py** to compute your feature, the function must be called "**name_**feature"

deps: the features that we need already computed in order to compute this feature. These can be other features in the yml file or columns native to the **uniq_key_stuent_data.csv** file. The code will then figure out the correct order to create the features so that all dependencies are met. Be careful not to create circular dependencies!

incl: whether or not to compute this feature. We might want to turn some features off in certain runs. Note that setting this to "True" *does not* mean that this feature will be a column in the final feature table. It only means that we will compute this feature and can use it as a dependency for other features

coll: whether or not to include this feature in the final feature table, **feature_table.csv.** If this is True, then incl must also be True. However, we can have cases where incl is True but this is False because we only want to compute this feature in order to help us compute another feature. Warning: Make sure that any features with coll set to True are at the (student, term) level. That is, make sure that it makes sense to assign that feature to a particular student in a particular term. As an example, "received_A_plus" would not be such a feature since this is at the (student, term, course) level. But, "number_A_pluses_so_far" is acceptable. So, we might still want to set incl for "received_A_plus" to true but set its coll to false since we only want to use it to help us compute "number_A_pluses_so_far".

type: this can be either ‘cts’ or ‘cat’ which stand for continuous and categorical respectively. A continuous feature is one that comes from a continuous probability distribution, such as “current_student_gpa”. A categorical feature comes from a discrete probability distribution, such as “student_major”. It is important to mark this field correctly because it dictates how this feature is treated at the machine learning step.

3. **Creating Labeled Feature Table**

Now that we have a feature table, we want to attach our labels that we are trying to predict. A “label” is simply the variable that we are trying to predict. In this case it is something like “drops_out_in_next_year” or “drops_out_in_next_quarter”.

a. We use as input the **feature_table.csv** and the **extra_data.csv** which contains important information about which student graduate from the mathematics major and which do not.
b. The script that performs this task is **attach_labels.py**
c. We know this task is done if the file **attach_labels.s** exists in the current directory. If it does, no need to run this task. If it does not, run this task and all the come after.

4. **Create Model Outputs**

We are now read to perform the machine learning phase of the project where we actually make predictions about the students in our feature table and store these results somewhere for later evaluation.

a. We use as input the **feature_table.csv** and **machine_learning.yml**, which will be described below.
b. The script that performs this task is **generate_predictions.py**.
c. We store the results of each model in a pickle (.p) file. A pickle file is a frozen Python object, such as a dictionary, list, or dataframe, which can be later revived. In our case, each pickle file contains a dataframe of predictions and a list of most important features for this model. This files live in the **../model_ouptut/** directory.

   We know we need to rerun this task if a success file called **generate_models.s** does not exist. If it does not exist, we check if **machine_learning.yml** is younger than the success file. It the yml file is younger, we need to rerun this step, otherwise, we do not. We go one step further and say that even when we need to rerun this step, we only run *new* models. That is, if there are already pickle files for some model, we do not need to rerun it.

   The parameters in the **machine_learning.yml** configuration file are discussed below:

train_start_year: which year to start the training set

num_train_years: how many years should the training set span

num_test_years: how many years should the testing set span (note the testing set begins where the training set ends)

models_to_run: space separated list of all the model types we wish to run. We use short names here (eg. LR = logistic regression). You can find out all the short names by inspecting the **generate_predictions.py** file.

feat_tbl_name: name of the feature table

date_col: which column in the feature table represents the numeric term

prediction_var: which column of the feature table we want to predict

feats_to_use: comma separated list of all the features from the feature table we wish to use in our models. We do not have to use all of them. Note also that if you have two features "gpa_last_quarter" and "gpa_so_far", and you put only "gpa" as the feature to use in this list, the code will pick up both "gpa_last_quarter" and "gpa_so_far" as features to use. In general, if you include a feature to use, it will pick up all features from the feature table that have that search text contained in their names.

train_tbl_name: name of the training table (this is generated and used by the code automatically so don't worry too much about what you name it, something like train_table.csv is fine)

test_tbl_name: name of the training table (this is generated and used by the code automatically so don't worry too much about what you name it, something like test_table.csv is fine)

5. Evaluate Models

Now that we have our model outputs, we want to evaluate our models to see which are best under different metrics that we care about.

a. We use as input the pickle (.p) files in the directory **../model_output/** and the **evaluation.yml** file, whose fields will be discussed below
b. The script that runs this task is called **evaluation.py**
c. We know this task is done and does not need to be run if the **model_scores.csv** table exists in the current working directory and it is younger than the **evaluation.yml** configuration file. The final output of this pipeline is the **model_scores.csv** file which contains several fields about each model we run including precision and recall at some specified threshold, weighted score (discussed below) at that threshold, time the model was evaluated, and the top three features of the model (if applicable).

   Note also that we do not want to create new rows in our **model_scores.csv** file if a model has previously been evaluated and already has a row in the csv file. We thus use a file called **tracker.s** in the **../model_output/** directory whose timestamp is updated each time a new batch of models is run. Thus, we only add new rows for the pickle files newer than the **tracker.s** file.

The parameters in the **evaluation.yml** configuration file are discussed below:

<u>true_pos_reward</u>: The number of points we give for each true positive (predicted true and actually true) in the dataset. This should be greater than 0

<u>true_neg_reward:</u> The number of points we give for each true negative (predicted false and actually false) in the dataset. This should be greater than 0

<u>false_pos_penalty</u>: The number of points we take away for each false positive (predicted true and actually false) in the dataset. This should be less than 0

<u>false_neg_penalty</u>: The number of points we take away for each false negative (predicted false and actually true) in the dataset. This should be less than 0

<u>k</u>: The threshold at which to cutoff giving “true” predictions. For example, if k=25, we assign a prediction of “true” to the top 25 percent of scores in each model output.

<u>pics</u>: A True or False field of whether we want precision-recall plots for each of our models or not. If we pick “True”, make sure to have created the directory **../model_plots/** as this is where the plots will be generated