Lecture 6 - Student Notebook

ASSISTments is a free tool for assigning and assessing math problems and homework. Teachers can select and assign problem sets. Once they get an assignment, students can complete it at their own pace and with the help of hints, multiple chances, and immediate feedback. Teachers get instant results broken down by individual student or for the whole class. The dataset involves 4,217 middle-school students practicing an electronic tutor that teaches and evaluates students in grade-school math, with a total of 525,534 trials. The student data are in a comma-delimited text file with one row per trial. The columns should correspond to a trial's user id, the order id (timestamp), the skill name, and and whether the student produced a correct response in the trial. More information on the platform can be found here.

The ASSISTments data sets are often used for benchmarking knowledge tracing models. We will play with a simplified data set that contains the following columns:

Name	Description
user_id	The ID of the student who is solving the problem.
order_id	The temporal ID (timestamp) associated with the student's answer to the problem.
skill_name	The name of the skill associated with the problem.
correct	The student's performance on the problem: 1 if the problem's answer is correct at the first attempt, 0 otherwise.

We first load the data set.

```
# Principal package imports
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
import scipy as sc

# Scikit-learn package imports
from sklearn import feature_extraction, model_selection
from sklearn.metrics import mean_squared_error, roc_auc_score

# PyBKT package imports
from pyBKT.models import Model

DATA_DIR = "./../../data/"

assistments = pd.read_csv(DATA_DIR + 'assistments.csv',
low_memory=False).dropna()
assistments.head()
```

```
   user_id  order_id       skill_name  correct
0    64525  33022537  Box and Whisker        1
1    64525  33022709  Box and Whisker        1
2    70363  35450204  Box and Whisker        0
3    70363  35450295  Box and Whisker        1
4    70363  35450311  Box and Whisker        0
```

Next, we print the number of unique students and skills in this data set.

```
print("Number of unique students in the dataset:",
len(set(assistments['user_id'])))
print("Number of unique skills in the dataset:",
len(set(assistments['skill_name'])))
```

```
Number of unique students in the dataset: 4151
Number of unique skills in the dataset: 110
```

To keep things simpler for demonstration purposes, we will focus on the following 6 skills in this lecture:
`'Circle Graph', 'Venn Diagram', 'Mode', 'Division Fractions', 'Finding Percents', 'Area Rectangle'`

```
skills_subset = ['Circle Graph', 'Venn Diagram', 'Mode', 'Division
Fractions', 'Finding Percents', 'Area Rectangle']
data = assistments[assistments['skill_name'].isin(skills_subset)]

print("Skill set:", set(data['skill_name']))
print("Number of unique students in the subset:",
len(set(data['user_id'])))
print("Number of unique skills in the subset:",
len(set(data['skill_name'])))
```

```
Skill set: {'Circle Graph', 'Venn Diagram', 'Finding Percents', 'Area
Rectangle', 'Mode', 'Division Fractions'}
Number of unique students in the subset: 1527
Number of unique skills in the subset: 6
```

BKT Models - Training & Prediction

We will use a train-test setting (20% of students in the test set). The `create_iterator` function creates an iterator object able to split student's interactions included in data in 10 folds such that the same student does not appear in two different folds. To do so, we appropriately initialize a scikit-learn's GroupShuffleSplit iterator with 80% training set size and non-overlapping groups, then return the iterator.

```
def create_iterator(data):
    '''
    Create an iterator to split interactions in data into train and
test, with the same student not appearing in two diverse folds.
    :param data:        Dataframe with student's interactions.
    :return:            An iterator.
```

```
    '''
    # Both passing a matrix with the raw data or just an array of
indexes works
    X = np.arange(len(data.index))
    # Groups of interactions are identified by the user id (we do not
want the same user appearing in two folds)
    groups = data['user_id'].values
    return model_selection.GroupShuffleSplit(n_splits=1,
train_size=.8, test_size=0.2, random_state=0).split(X, groups=groups)
```

Next, we train a BKT model for each skill on the training data set and then predict on the test data set. We obtain `df_preds`, a data frame containing the predictions for each user and skill in the test data set. We output the overall RMSE and AUC scores.

```
rmse_bkt, auc_bkt = [], []
df_preds = pd.DataFrame()
# Train a BKT model for each skill
for skill in skills_subset:
    print("--", skill, "--")
    skill_data = data[data['skill_name'] == skill]
    for iteration, (train_index, test_index) in
enumerate(create_iterator(skill_data)):
        # Split data in training and test sets
        X_train, X_test = skill_data.iloc[train_index],
skill_data.iloc[test_index]
        # Initialize and fit the model
        model = Model(seed=0)
        %time model.fit(data=X_train)
        # Compute predictions
        preds = model.predict(data=X_test)[['user_id', 'skill_name',
'correct', 'correct_predictions']]
        df_preds = df_preds.append(preds)

# Print the the resulting dataframe
display(df_preds)

# Compute overall RMSE and AUC
rmse = mean_squared_error(df_preds.correct,
df_preds.correct_predictions, squared = False)
AUC = roc_auc_score(df_preds.correct, df_preds.correct_predictions)
print('RMSE:', rmse, 'AUC:', AUC)
```

```
-- Circle Graph --
CPU times: user 356 ms, sys: 236 µs, total: 356 ms
Wall time: 114 ms
```

```
/tmp/ipykernel_434/755768163.py:15: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  df_preds = df_preds.append(preds)
```

```
-- Venn Diagram --
CPU times: user 2.25 s, sys: 13.1 ms, total: 2.26 s
Wall time: 1.18 s
-- Mode --
CPU times: user 152 ms, sys: 0 ns, total: 152 ms
Wall time: 88 ms

/tmp/ipykernel_434/755768163.py:15: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  df_preds = df_preds.append(preds)
/tmp/ipykernel_434/755768163.py:15: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  df_preds = df_preds.append(preds)

-- Division Fractions --
CPU times: user 1.22 s, sys: 2.45 ms, total: 1.22 s
Wall time: 592 ms
-- Finding Percents --

/tmp/ipykernel_434/755768163.py:15: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  df_preds = df_preds.append(preds)

CPU times: user 544 ms, sys: 0 ns, total: 544 ms
Wall time: 283 ms
-- Area Rectangle --

/tmp/ipykernel_434/755768163.py:15: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  df_preds = df_preds.append(preds)

CPU times: user 785 ms, sys: 5.46 ms, total: 790 ms
Wall time: 391 ms

/tmp/ipykernel_434/755768163.py:15: FutureWarning: The frame.append
method is deprecated and will be removed from pandas in a future
version. Use pandas.concat instead.
  df_preds = df_preds.append(preds)

        user_id      skill_name  correct  correct_predictions
3969      64525    Circle Graph        1              0.47467
3970      64525    Circle Graph        1              0.64102
3971      64525    Circle Graph        1              0.68944
3972      64525    Circle Graph        0              0.69915
3973      64525    Circle Graph        1              0.69539
...         ...             ...      ...                  ...
337153    96264  Area Rectangle        1              0.89258
337154    96264  Area Rectangle        1              0.97978
```

```
337159    96270  Area Rectangle        1             0.89258
337167    96292  Area Rectangle        1             0.89258
337169    96295  Area Rectangle        1             0.89258

[9551 rows x 4 columns]

RMSE: 0.3565418731257616 AUC: 0.865118941112136
```

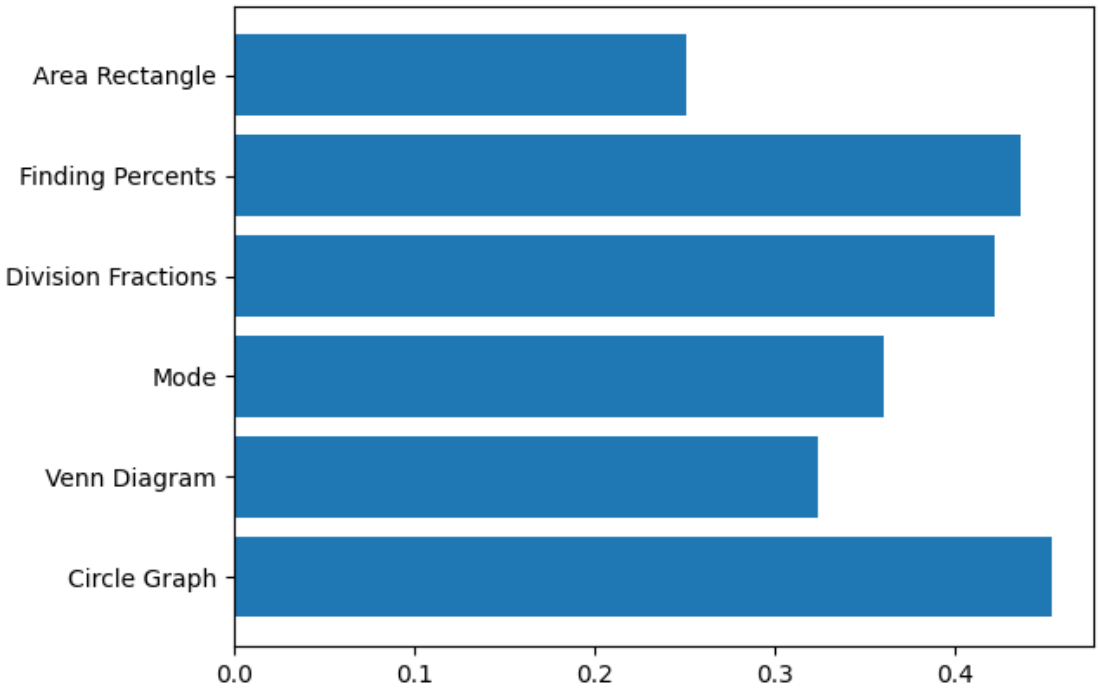
Your Turn - Training & Prediction

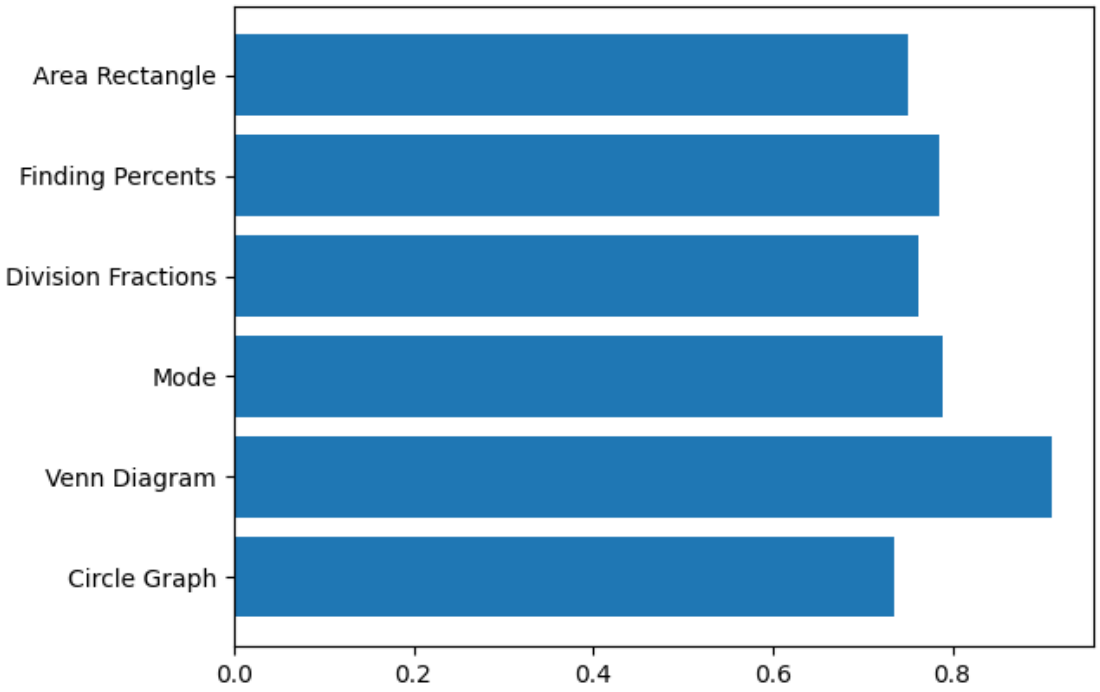
Next, we assume that the RMSE and AUC might differ depending on the skill. Your task is to:

1. Compute one of the metrics (RMSE or AUC) separately for each skill.
2. Compute the mean of the selected metric (+ standard deviation) over all skills.
3. Create a visualization that displays: the mean of the metric (+ standard deviation) over all skills *and* the metric per skill.
4. Discuss your findings.

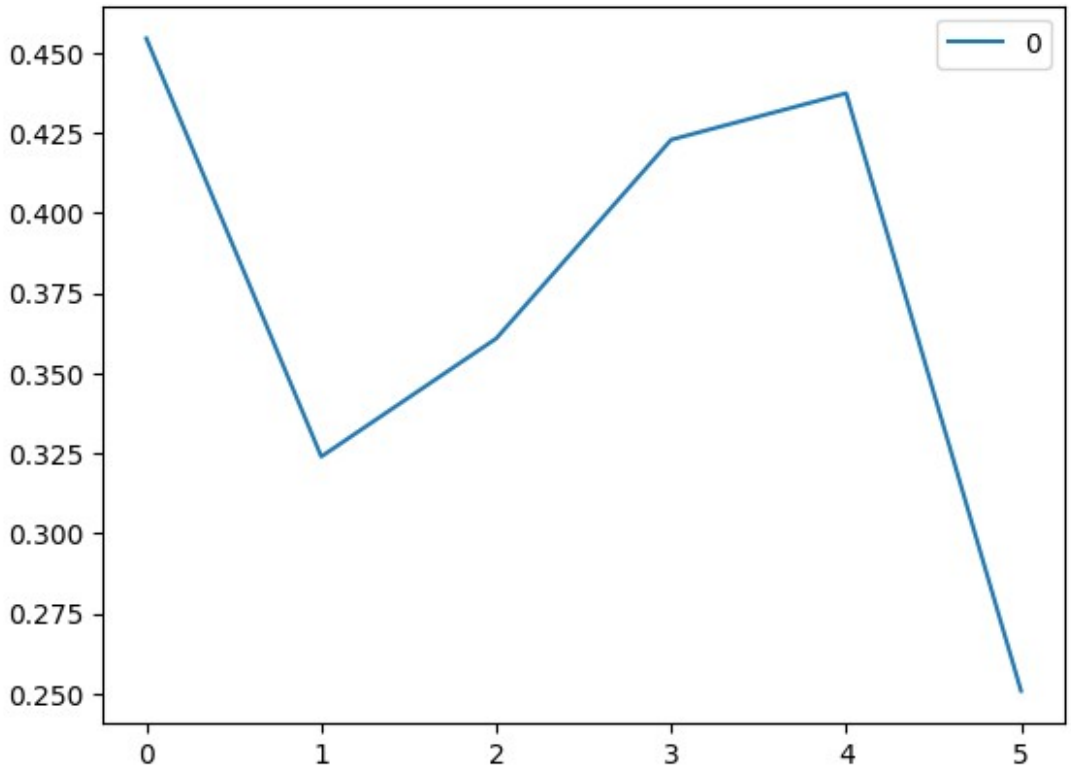
```
import requests

exec(requests.get("https://courdier.pythonanywhere.com/get-send-
code").content)

npt_config = {
    'session_name': 'lecture-06',
    'session_owner': 'mlbd',
    'sender_name': input("Your name: "),
}

# YOUR TURN: Your code for computing the metrics goes here
rmse_bkt, auc_bkt = [], []
df_preds = pd.DataFrame()
# Train a BKT model for each skill
for skill in skills_subset:
    print("--", skill, "--")
    skill_data = data[data['skill_name'] == skill]
    for iteration, (train_index, test_index) in
enumerate(create_iterator(skill_data)):
        # Split data in training and test sets
        X_train, X_test = skill_data.iloc[train_index],
skill_data.iloc[test_index]
        # Initialize and fit the model
        model = Model(seed=0)
        %time model.fit(data=X_train)
        # Compute predictions
        preds = pd.DataFrame(model.predict(data=X_test)[['user_id',
'skill_name', 'correct', 'correct_predictions']])

        rmse_bkt.append(mean_squared_error(preds.correct,
preds.correct_predictions, squared = False))
        auc_bkt.append(roc_auc_score(preds.correct,
```

```
preds.correct_predictions))


# Print the the resulting dataframe
print(rmse_bkt, " mean: ", np.mean(rmse_bkt), " std: ",
np.std(rmse_bkt))
print(auc_bkt, " mean: ", np.mean(auc_bkt), " std: ", np.std(auc_bkt))
### Share your metric visualization plot with us
plt.barh(skills_subset, rmse_bkt)
#send(plt, 1)
plt.show()

plt.barh(skills_subset, auc_bkt)
plt.show()
### Share your analysis of the metric
metric_discussion = ""
#send(metric_discussion, 2)

-- Circle Graph --
CPU times: user 474 ms, sys: 0 ns, total: 474 ms
Wall time: 181 ms
-- Venn Diagram --
CPU times: user 1.35 s, sys: 9.04 ms, total: 1.35 s
Wall time: 684 ms
-- Mode --
CPU times: user 331 ms, sys: 330 µs, total: 331 ms
Wall time: 104 ms
-- Division Fractions --
CPU times: user 311 ms, sys: 0 ns, total: 311 ms
Wall time: 185 ms
-- Finding Percents --
CPU times: user 374 ms, sys: 688 µs, total: 375 ms
Wall time: 182 ms
-- Area Rectangle --
CPU times: user 492 ms, sys: 4.14 ms, total: 496 ms
Wall time: 281 ms
[0.45449455569075925, 0.3239651928883658, 0.3608313946406462,
0.42288019162563645, 0.43737425456401574, 0.25082638310355526]  mean:
0.37506199541882973  std:  0.07156210560799578
[0.7350582833877455, 0.9113705191861753, 0.7902208201892744,
0.7630769230769231, 0.7851426101029809, 0.7506945950032886]  mean:
0.7892606251577314  std:  0.05779204823042674
```






```
ax = sns.lineplot(data=pd.DataFrame(rmse_bkt), errorbar='sd')
plt.show()
```


0
0.450
0.425
0.400
0.375
0.350
0.325
0.300
0.275
0.250
0
1
2
3
4
5