Predicting Professor Ratings With Machine Learning

Daniel Son

Context

Students at the University of Maryland use the website PlanetTerp to leave reviews and star ratings for professors and courses they enrolled in.

Dylan Carpenter (ENGL391 - Advanced Composition)

“Such a nice person, and he really cares about his students. He is a very lenient grader, and the assignments aren't that hard…”
– Anonymous

Ting Jiang (CMSC351 - Algorithms)

“A fantastic instructor. She is extremely organized and always willing to help her students succeed. Her slides are very informative and are posted on Canvas…”
– Anonymous

The Goal

How well do students' reviews reflect the star rating they award to a professor?

With the power of machine learning, we can uncover the relationship between what students write and the score they ultimately choose

We must build a tool that takes in a review of a professor and predicts how many stars that review gave

The Data

Ting Jiang

Total reviews: 26

Average rating: 3.42

Courses taught:

- CMSC351
- CMSC417

Larry Herman

Total reviews: 336

Average rating: 3.14

Courses taught:

- CMSC106
- CMSC131
- CMSC132
- CMSC216
- CMSC250
- CMSC330

Maksym Morawski

Total reviews: 247

Average rating: 2.80

Courses taught:

- CMSC106
- CMSC250
- CMSC320
- CMSC351
- CMSC421
- DATA602

Ilchul Yoon

Total reviews: 175

Average rating: 2.26

Courses taught:

- CMSC106
- CMSC122
- CMSC131
- CMSC132
- CMSC216
- CMSC389N
- CMSC411

Allan Yashinski

Total reviews: 93

Average rating: 4.60

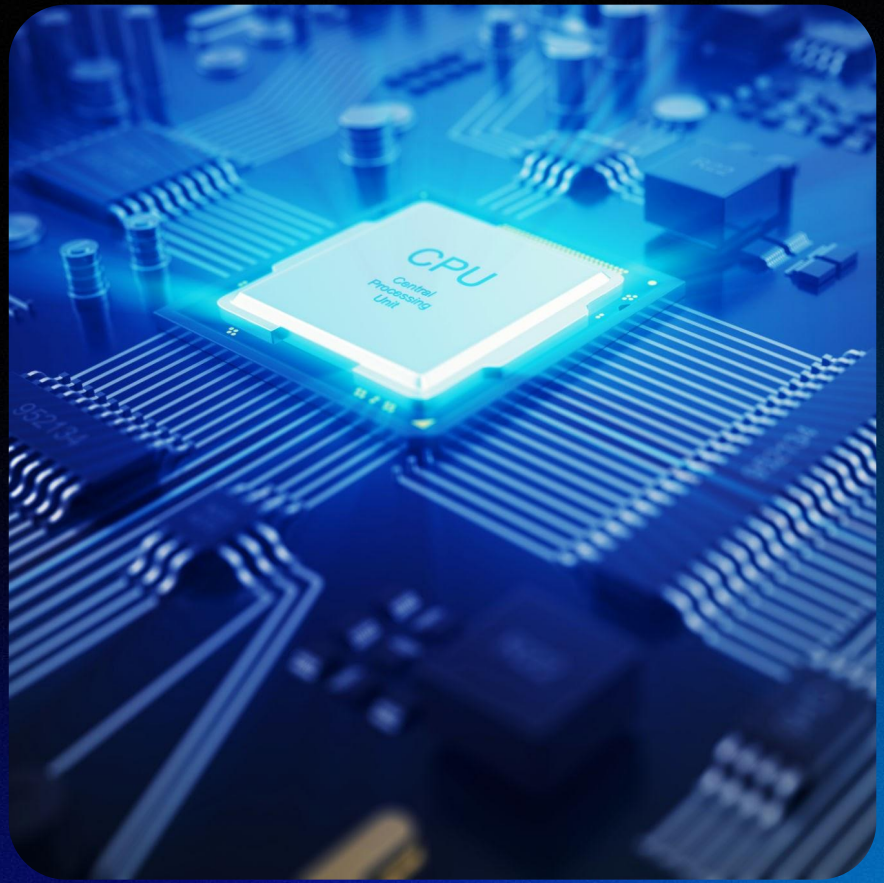
Courses taught:

- MATH240
- MATH241H
- MATH401
- MATH402
- MATH403
- MATH405
- MATH456
- MATH461
- MATH463

The ML Model

Model: PyTorch regressor

- Pre-trained DistilBert
- AdamW optimizer
- Predicts a continuous value 1-5 stars
- 5 epochs
- 80/20 test split



The Process

01 **Extract and Clean Data**

02 **Run PyTorch Model**

03 **Results and Analysis**

1. Extract and Clean Data

1. Obtained professor reviews from PlanetTerp, extracting key data such as
   - Paragraph review
   - Star rating (0-5)
   - Expected letter grade

2. Cleaned data by removing rows with missing data

2. Run PyTorch Model

1. Tokenized review text data and generated attention mask tensors
2. Converted token sequences into contextual embeddings representing the overall meaning of the review
3. Passed embedding and corresponding one-hot encoded expected grade into a neural network to predict star rating

3. Results and Analysis

Epoch	Training Loss
1	3.64607
2	1.11485
3	0.40607
4	0.25080
5	0.17374

Stat	Measurement
Mean Absolute Error	.59 stars
Test MSE	.70 stars
Test RMSE	.83 stars
Accuracy (within ±0.25)	35%
Accuracy (within ±0.5)	55%

3. Results and Analysis

- Strong grasp of overall sentiment
- Central tendency toward high ratings
- Clear sentiment cues dominate predictions
    - Reviews with explicit sentiment words (“worst,” “fantastic,” “excellent”) tend to have very small error, sometimes within ±0.1 stars

Overall:

Most predictions are reasonably close to the true ratings, and the model clearly learns meaningful sentiment patterns from the reviews

Summary

- Collected PlanetTerp reviews for 5 UMD professors
- Fine-tuned a DistilBERT transformer in PyTorch
- Achieved statistically significant accuracy

Conclusion:
Students' reviews can be used to predict how highly they rate their professors

Follow up question:
Is there a correlation between expected grade and star rating?

55%

accuracy within ±0.5 stars

0.59

stars away from true rating, on average

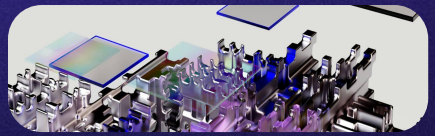
877

reviews used to train model

0.17

training loss after five epochs

Limitations



Small Dataset

With only 5 professors and 877 reviews, the model limits generalization

Lack of Stratification

Single train/test split with no stratification means metrics may be optimistic if similar reviews land in both sets

Weak Feature Handling

One-hot can mishandle missing/rare grades and learn factitious correlations

Overfitting Risk

DistilBERT is heavy for the dataset size, so results have high variance

Thank you!

Disclaimer: the code attached may have different results

hw4

December 11, 2025

[23]:
```python
import requests

BASE_URL = "https://planetterp.com/api/v1"

professors = [
    "Ting Jiang",
    "Larry Herman",
    "Maksym Morawski",
    "Ilchul Yoon",
    "Allan Yashinski",
]

def get_course(name: str, reviews: bool = False):
    params = {
        "name": name,
        "reviews": str(reviews).lower()
    }
    r = requests.get(f"{BASE_URL}/professor", params=params)
    r.raise_for_status()
    return r.json()

for prof in professors:
    data = get_course(prof)

    # remove duplicates by converting to a set, then back to a list for printing
    unique_courses = sorted(set(data.get("courses", [])))

    print(prof, "teaches:")
    print(unique_courses)
    print()
```

```
Ting Jiang teaches:
['CMSC351', 'CMSC417']

Larry Herman teaches:
['CMSC102', 'CMSC132', 'CMSC132H', 'CMSC216', 'CMSC330']

Maksym Morawski teaches:
```

```
['BIOI602', 'CMSC106', 'CMSC250', 'CMSC320', 'CMSC351', 'CMSC398P', 'CMSC421',
'DATA320', 'DATA602', 'MSML602']

Ilchul Yoon teaches:
['BMGT302', 'BMGT402', 'CMSC106', 'CMSC122', 'CMSC131', 'CMSC132', 'CMSC216',
'CMSC335', 'CMSC389N', 'CMSC411']

Allan Yashinski teaches:
['CMSC456', 'ENEE456', 'HLTH710', 'MATH141H', 'MATH240', 'MATH241H', 'MATH401',
'MATH402', 'MATH403', 'MATH405', 'MATH406', 'MATH423', 'MATH436', 'MATH456',
'MATH461', 'MATH462', 'MATH463']
```

[24]:
```python
professor_reviews = {}

for prof in professors:
    data = get_course(prof, reviews=True)
    professor_reviews[prof] = data.get("reviews", [])
    print(f"{prof}: retrieved {len(professor_reviews[prof])} reviews")
```

```
Ting Jiang: retrieved 26 reviews
Larry Herman: retrieved 336 reviews
Maksym Morawski: retrieved 247 reviews
Ilchul Yoon: retrieved 175 reviews
Allan Yashinski: retrieved 93 reviews
```

[25]:
```python
import pandas as pd

# move fetched data into a dataframe
rows = []
for prof, reviews in professor_reviews.items():
    for r in reviews:
        rows.append({
            "professor": prof,
            "course": r.get("course"),
            "review": r.get("review"),
            "rating": r.get("rating"),
            "expected_grade": r.get("expected_grade"),
        })

df = pd.DataFrame(rows)
print(df)
```

```
          professor   course  \
0         Ting Jiang  CMSC351
1         Ting Jiang     None
2         Ting Jiang  CMSC351
3         Ting Jiang  CMSC351
4         Ting Jiang  CMSC351
```

```
..              …      …
872  Allan Yashinski  MATH405
873  Allan Yashinski  MATH461
874  Allan Yashinski  CMSC456
875  Allan Yashinski  MATH461
876  Allan Yashinski  MATH401

                                                 review  rating expected_grade
0    Ting is an ok lecturer but i always found myse…       4              A-
1    bad lecturer decent exams saved by justin's no…       3              A-
2    Ting is an okay lecturer and a nice person. Sh…       4              B+
3    She's a pretty boring and unhelpful lecturer, …       3               B
4    Setting aside the co-teaching with Justin, Tin…       2              A-
..                                                   …     …              …
872  Allan for MATH405 is not my most recommended p…       4              B+
873  He's a great lecturer; he really explains conc…       5              B+
874  Allan is an awesome professor. Great lectures,…       4               B
875  Allan is the goat. I find his lectures to be e…       5               A
876  This is a very engaging and well-structured cl…       4

[877 rows x 5 columns]
```

[26]:
```
# replace empty strings with NaN
df.replace("", pd.NA, inplace = True)

# drop rows with missing values
df_clean = df.dropna(subset = ["review", "rating"])

print(df_clean)
```

```
          professor   course  \
0         Ting Jiang  CMSC351
1         Ting Jiang     None
2         Ting Jiang  CMSC351
3         Ting Jiang  CMSC351
4         Ting Jiang  CMSC351
..              …      …
872  Allan Yashinski  MATH405
873  Allan Yashinski  MATH461
874  Allan Yashinski  CMSC456
875  Allan Yashinski  MATH461
876  Allan Yashinski  MATH401

                                                 review  rating expected_grade
0    Ting is an ok lecturer but i always found myse…       4              A-
1    bad lecturer decent exams saved by justin's no…       3              A-
2    Ting is an okay lecturer and a nice person. Sh…       4              B+
3    She's a pretty boring and unhelpful lecturer, …       3               B
```

```
4    Setting aside the co-teaching with Justin, Tin…       2             A-
..                                                  …     …           …
872  Allan for MATH405 is not my most recommended p…       4            B+
873  He's a great lecturer; he really explains conc…       5            B+
874  Allan is an awesome professor. Great lectures,…       4             B
875  Allan is the goat. I find his lectures to be e…       5             A
876  This is a very engaging and well-structured cl…       4          <NA>

[876 rows x 5 columns]
```

[27]:
```
import torch
from torch.utils.data import Dataset

class ReviewDataset(Dataset):
    def __init__(self, df, tokenizer, grade_to_index, max_len = 256):
        self.df = df
        self.tokenizer = tokenizer
        self.grade_to_index = grade_to_index
        self.max_len = max_len

    def __len__(self):
        return len(self.df)

    def __getitem__(self, index):

        # maps letter grade to numerical value
        row = self.df.iloc[index]
        text = row["review"]
        grade = row["expected_grade"] if pd.notna(row["expected_grade"]) else␣
 ↪"Unknown"
        grade_index = self.grade_to_index[grade]

        # converts numerical value to tensor
        rating = torch.tensor(float(row["rating"]), dtype = torch.float)

        # tokenizes review text
        tokens = self.tokenizer(
            text,
            padding="max_length",
            truncation = True,
            max_length = self.max_len,
            return_tensors = "pt",
        )

        # one hot encode expected grade
        grade_onehot = torch.nn.functional.one_hot(
            torch.tensor(grade_index),
```

```
            num_classes = len(self.grade_to_index)
        ).float()

        return {
            "input_ids": tokens["input_ids"].squeeze(),
            "attention_mask": tokens["attention_mask"].squeeze(),
            "grade": grade_onehot,
            "rating": rating,
        }
```

[28]:
```
from transformers import DistilBertModel

class RatingRegressor(torch.nn.Module):
    def __init__(self, num_grades):
        super().__init__()

        # encoder
        self.bert = DistilBertModel.from_pretrained("distilbert-base-uncased")

        # 768 dimensional embedding
        hidden_size = 768

        # regression to predict rating
        self.regressor = torch.nn.Sequential(
            torch.nn.Linear(hidden_size + num_grades, 256),
            torch.nn.ReLU(),
            torch.nn.Linear(256, 1)
        )

    def forward(self, input_ids, attention_mask, grade):

        # run text through bert
        bert_out = self.bert(input_ids = input_ids, attention_mask =␣
 ↪attention_mask)

        # embedding representing meaning of sentence
        cls = bert_out.last_hidden_state[:, 0]

        # return as a 1D tensor
        x = torch.cat([cls, grade], dim = 1)
        out = self.regressor(x)
        return out.squeeze(-1)
```

[29]:
```
from torch.utils.data import DataLoader
from transformers import DistilBertTokenizerFast
from sklearn.model_selection import train_test_split
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

tokenizer = DistilBertTokenizerFast.from_pretrained("distilbert-base-uncased")

# convert letter grades to numerical values
all_grades = sorted(df_clean["expected_grade"].fillna("Unknown").unique().
 ↪tolist())
grade_to_index = {g: i for i, g in enumerate(all_grades)}
num_grades = len(all_grades)

# split data 80/20
train_df, test_df = train_test_split(df_clean, test_size=0.2)
train_ds = ReviewDataset(train_df, tokenizer, grade_to_index=grade_to_index)
test_ds  = ReviewDataset(test_df, tokenizer, grade_to_index=grade_to_index)

# create batches
train_loader = DataLoader(train_ds, batch_size=8, shuffle=True)
test_loader  = DataLoader(test_ds, batch_size=8)

# build model
model = RatingRegressor(num_grades=num_grades).to(device)

# updates weights based on gradient
optimizer = torch.optim.Adam(model.parameters(), lr=2e-5)

# MSE loss
criterion = torch.nn.MSELoss()

# five training loops
for epoch in range(5):
    model.train()
    total_loss = 0

    for batch in train_loader:
        optimizer.zero_grad()

        pred = model(
            batch["input_ids"].to(device),
            batch["attention_mask"].to(device),
            batch["grade"].to(device)
        )

        loss = criterion(pred, batch["rating"].to(device))
        loss.backward()
        optimizer.step()

        total_loss += loss.item()
```

```
    print(f"Epoch {epoch+1} : Training loss: {total_loss / len(train_loader):.
↪5f}")
```

```
Epoch 1 : Training loss: 3.52424
Epoch 2 : Training loss: 1.61950
Epoch 3 : Training loss: 0.65728
Epoch 4 : Training loss: 0.31405
Epoch 5 : Training loss: 0.18644
```

[30]:
```
import numpy as np

model.eval()
predictions = []
true = []

# collects predicted and true ratings
with torch.no_grad():
    for batch in test_loader:
        prediction = model(
            batch["input_ids"].to(device),
            batch["attention_mask"].to(device),
            batch["grade"].to(device),
        )

        predictions.extend(prediction.cpu().numpy())
        true.extend(batch["rating"].cpu().numpy())

mse = np.mean((np.array(predictions) - np.array(true))**2)
print("Test MSE:", mse)
print("Test RMSE:", np.sqrt(mse))
```

```
Test MSE: 0.5999168
Test RMSE: 0.774543
```

[31]:
```
from sklearn.metrics import r2_score

prediction_df = pd.DataFrame({
    "review": test_df["review"].values,
    "predicted_rating": predictions,
    "true_rating": true
})

# how far off each prediction is
prediction_df["off_by"] = (
    prediction_df["predicted_rating"] - prediction_df["true_rating"]
).abs()
```

```python
# average off by (MAE)
avg_off_by = prediction_df["off_by"].mean()

# accuracy thresholds
acc_025 = (prediction_df["off_by"] <= 0.25).mean()
acc_05  = (prediction_df["off_by"] <= 0.5).mean()

# R^2
r2 = r2_score(prediction_df["true_rating"], prediction_df["predicted_rating"])

print(prediction_df)
print("Average off by (MAE):", avg_off_by)
print("Accuracy (within ±0.25):", acc_025)
print("Accuracy (within ±0.5):", acc_05)
print("R²:", r2)
```

```
                                                review  predicted_rating  \
0    This was genuinely one of my favorite courses …          5.043595
1    Maksym did a fantastic job with this course an…          4.960024
2    Max is goated idc what the other reviews say. …          4.524414
3    Yoon's class is really keeping me on my toes. …          2.356693
4    bro pick anybody, not him. When he explains th…          1.455905
..                                                 …                ⋯
171  This professor has been THE LEAST accomodating…          1.293905
172  Take it with someone else if you actually want…          2.182126
173  Great instructor. Explains difficult concepts …          5.036975
174  Definitely try to avoid him. \r\n\r\nHis lectu…          1.927685
175  Really good professor. He's great at teaching,…          4.628873

     true_rating    off_by
0            5.0  0.043595
1            5.0  0.039976
2            5.0  0.475586
3            5.0  2.643307
4            1.0  0.455905
..           …       ⋯
171          1.0  0.293905
172          1.0  1.182126
173          5.0  0.036975
174          2.0  0.072315
175          5.0  0.371127

[176 rows x 4 columns]
Average off by (MAE): 0.563574
Accuracy (within ±0.25): 0.3693181818181818
Accuracy (within ±0.5): 0.5738636363636364
R²: 0.7552772760391235
```