Have 100 students’ records with three sets.

Take 70 students for training

15 for validation

15 for test

Structure data: so use a random forest model of five trees for the five potential scores

Train so the inputs are the quizzle scores… and outputs (target variables) are the predicted AP scores.

Use a small sample to start --- like five students to train the model.

Then, we want to tune the model by using the training data.

Then we do a model comparison on the test data when a new student completes the quizzle. So, the results will not underfit or overfit compared to the training set and stay in the Goldilocks zone, meaning my model is good.

If it overfits, it means the model has data leakage where the training data leaks out into the test data. Ensure the splits are correct so the model doesn’t see the training data during the testing phase. Also, data mismatch if there are different features between the training data and test data, which can cause underfitting but should not occur since the test data and training data have the same features. But, a way to reduce overfitting and underfitting is to try a more advanced model, increase the model’s hyperparameters, reduce the number of features, or train the model for longer. This will help with underfitting. We can collect more data and try a less advanced model for overfitting.

Before we get into the experimentation side of things, it's worth having a little reminder of overfitting and underfitting are.

All experiments should be conducted on different portions of your data.

* **Training data set** — Use this set for model training, 70–80% of your data is the standard.
* **Validation/development data set** — Use this set for model hyperparameter tuning and experimentation evaluation, 10–15% of your data is the standard.
* **Test data set** — Use this set for model testing and comparison, 10–15% of your data is the standard.

These amounts can fluctuate slightly, depending on your problem and the data you have.

Poor performance on training data means the model hasn’t learned properly and is **underfitting**. Try a different model, improve the existing one through hyperparameter or collect more data.

Great performance on the training data but poor performance on test data means your model doesn’t generalize well. Your model may be **overfitting** the training data. Try using a simpler model or making sure your the test data is of the same style your model is training on.

Another form of **overfitting** can come in the form of better performance on test data than training data. This may mean your testing data is leaking into your training data (incorrect data splits) or you've spent too much time optimizing your model for the test set data. Ensure your training and test datasets are kept separate at all times and avoid optimizing a models performance on the test set (use the training and validation sets for model improvement).

Poor performance once deployed (in the real world) means there’s a difference in what you trained and tested your model on and what is actually happening. Ensure the data you're using during experimentation matches up with the data you're using in production.

There a couple of ways to do this:

1. Share your entire project folder (including the environment folder containing all of your Conda packages).

2. Share a .yml (pronounced YAM-L) file of your Conda environment.

The benefit of 1 is it's a very simple setup, share the folder, activate the environment, run the code. However, an environment folder can be quite a large file to share.

That's where 2 comes in. A .yml is basically a text file with instructions to tell Conda how to set up an environment.

For example, to export the environment we created earlier at /Users/daniel/Desktop/project\_1/env as a YAML file called environment.yml we can use the command:

conda env export --prefix /Users/daniel/Desktop/project\_1/env > environment.yml

After running the export command, we can see our new .yml file stored as environment.yml.

A sample .yml file might look like the following:

1. name: my\_ml\_env
2. dependencies:
3. - numpy
4. - pandas
5. - scikit-learn
6. - jupyter
7. - matplotlib

Of course, your actual file will depend on the packages you've installed in your environment.

For more on sharing an environment, check out the [Conda documentation on sharing environments](https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html" \l "sharing-an-environment" \t "_blank).

Finally, to create an environment called env\_from\_file from a .yml file called environment.yml, you can run the command:

conda env create --file environment.yml --name env\_from\_file

Top of Form

Bottom of Form

Top of Form

Bottom of Form