Programming Assignment 4

When you develop ML models in the production, you often have more resources (eg powerful servers) and more constraints (eg business requirements). In this assignment, you will learn some techniques that have become increasingly popular in the last few years for effectively handling the resources and constraints.

Task 1 : AutoML with AutoGluon (20 points)

In this task, you will explore the emerging topic of AutoML. In the class, we have talked about some automated methods for designing good classifiers. For example, we talked about 4 algorithms to select good hyper parameters. We also talked about various rule-of-thumbs for designing effective classifiers. Similarly, it is also possible to design effective algorithms to search the design space of feature engineering, data preprocessing, hyper-parameters, ensembling, end-to-end ML pipelines etc.

When you have a lot of time and/or resources, you can use AutoML tools. They have sophisticated algorithms (such as based on Bayesian methods, meta learning, genetic programming etc) to give good ML models.

AutoGluon is a popular approach for doing AutoML. There are other methods (e.g. auto-sklearn, TPOT etc) that take different approaches. A big issue in the AutoML field is that these tools do not work well together and have lot of idiosyncracies. For e.g. AutoGluon does not work with Python 3.12 yet (released two years ago even though 3.13 is already released), auto-sklearn requires fairly old sklearn version and so on. Nevertheless, these tools are quite useful to understand how automated techniques can give good ML models beyond what the vast majority of data scientists can generate. In fact, I often challenge my employees to design a classifier that beats the performance of these AutoML models. This trend is likely to be accelerated with the advent of ChatGPT and Claude. So it is essential that you folks find some way to not be replaced by these tools :)

Note 1: AutoGluon installs and uses lots and lots of models to do AutoML. This might cause some problems in Windows or Mac. If you face any issues, please see [here](https://auto.gluon.ai/stable/install.html) for instructions to fix them. If you still face issues, please post in forum. For Windows, it is highly recommended to use WSL instead of direct installation.

Note 2: AutoGluon will train and use LOTs of models in AutoGluon folder. Each run can potentially be 100MB+ . So please periodically clean this folder. Of course, when submitting, remove this folder.

Note 3: As mentioned above, you need to use Python 3.11 when installing AutoGluon. So, you need to setup your pyproject.toml and Rye for this. Specifically, look at how to use .python-version file to control your Python environment.

Some good resources for learning AutoGluon are:

1. [GitHub Repo](https://github.com/autogluon/autogluon)

2. [Quick Start](https://auto.gluon.ai/stable/tutorials/tabular/tabular-quick-start.html)

3. [More Details](https://auto.gluon.ai/stable/tutorials/tabular/tabular-essentials.html)

4. [Even More Details](https://auto.gluon.ai/stable/tutorials/tabular/tabular-indepth.html)

5. [Some low level details](https://auto.gluon.ai/stable/tutorials/tabular/how-it-works.html)

This is a simple task. You just need to write a class named AutoMLWithAutoGluon with three functions. You can use any dataset for this.

- Function `train_automl_model` will instantiate TabularPredictor and fit it with appropriate preset and time limit (eg medium with 2 minutes).
- Function `test_automl_model` so that it returns the evaluation of various accuracy metrics of how the AutoML did on the test set.
- Function `print_leaderboard` prints the leaderboard of the models.
- Function `print_feature_importance` prints the importance of each feature for the test data.

Task 2: Mixture of Experts with Gating Network (20 points)

Ensembling typically produces the state-of-the-art results when it comes to non-DL / traditional ML models. In the class, we talked about various ensembling techniques. Most of these techniques (bagging, boosting, voting, stacking, blending and others) are implemented in libraries like scikit-learn.

In this task, you will implement an idea that is becoming increasingly popular in the deep learning (specifically large language models) and apply it to traditional scikit-learn.

1. Choose some arbitrary dataset (e.g. use the make_classification function from scikit-learn)
2. Choose 3 diverse ML models as the "experts" (eg decision tree, SVM, KNN)
3. Train each of the model on the training data.

4. In this task, we will use a very simple MoE approach. Ensemble methods will combine the predictions of all the three model to produce final output. In MoE, we will use a routing logic using a gating network that tells us which of the three models we should use for making the prediction. Then we will just produce the output of that model. In other words, the gating network identifies the right "expert" for this input and we will use this.
5. The implementation should work something like this: After you trained the model from Step 3, you will train a gating network. The input to this model is the training data. Instead of using the original labels, you will train the model that is the expert for this tuple. The gating network takes the tuple/features (ie X) as input and will produce the index of the best classifier as output (e.g. 0 / 1 / 2 for first / second / third classifier). We will use a simple logic to find the best classifier. For each tuple in the training data, we will choose the classifier that produced the highest probability for the correct label. For example, suppose that the label for tuple i is 1. Then we will get the predicted probability (using predict_proba) for class 1 for all three classifiers. Then we will use that as the label for that tuple. You have to use the same logic if the class is 0. It is imperative to get this right otherwise your model will behave weirdly. Once a dataset of (X, best expert) is collected, you can use any ML model (e.g. Logistic Regression or decision tree) to implement the gating network.
6. During testing time, we will first pass the input tuple to the gating network that will output the best "expert". Then we will pass the tuple to the corresponding ML model and will return the final prediction.
7. You should also implement a function that prints the accuracy score of the individual models and the ensemble using MoE on the test dataset. (e.g. Accuracy of model 1 is X; Accuracy of model 2 is Y. Accuracy of model 3 is Z. Accuracy of MoE is ABC).

Task 3: Bias-Variance Decomposition (20 points)

In your ML class, you learned about the bias-variance tradeoff. However, a big issue is that the insight is not very actionable. In this task, you will learn how to make it more actionable.

Suppose you are training a model for some task. Typically, it will not produce good results. The first step is to do hyper parameter tuning. If that still does not work, you can try out some of the basic ML debugging tricks that we talked in the class. If that also does not work, a good next step is to try another model. However, it does not make sense to randomly try other models. It has to be guided by the bias-variance decomposition of your current model. Once you get a decomposition, it can be used in two ways. First, if your model has high bias, then you can try other lower bias models (e.g. decision tree). Second, you can choose an appropriate ensemble model. Specifically, if your model has high variance, you will choose a variance reduction tactic such as bagging. If it has high bias, you will try bias reduction technique such as boosting.

You will use the bias_variance_decomp function from mlxtend library. The documentation can be seen at https://rasbt.github.io/mlxtend/user_guide/evaluate/bias_variance_decomp/

In this task, you will generate some figures such as those in Slides 36, 47 and 52 of 14_model_ensembles.pdf from Moodle.

Concretely, generate the impact of bias-variance when vary

1. max_depth of decision tree
2. n_estimators of RandomForest
3. n_estimators of GradientBoostingClassifier
4. n_estimators of AdaBoostClassifier

Your submission should produce both the code and the matplotlib plot for these scenarios.

Task 4: Data Hardness Analysis (20 points)

In our course, we mostly focused on designing increasingly powerful models and various tricks to improve them. Another emerging topic is to shift the focus on data instead of models. Even though the research on DCAI (data centric AI) is still in infancy, there has been lot of interesting tools. In this task, you will try a popular package for analyzing one aspect of data complexity : data hardness.

You will use the Datagnosis library from https://github.com/vanderschaarlab/Datagnosis .

In this task, you can choose 2 thematically diverse models (e.g. DecisionTree/SVM or DecisionTree/RandomForest). Next, you will choose any 3 hardness metrics from https://datagnosis.readthedocs.io/en/latest/methods.html .

Then you will do three things for this combination:

(a) display the hardness scores of the dataset as a plot
(b) identify the top-10 hardest examples
(c) write a short para if you find any patterns/ rationale about why these examples are hard.

Task 5: Fine Tuning a Traditional Classifier (20 points)

Finetuning has become a common technique for deep learning models where you tweak a generic model for your purpose. It is possible to do the same for traditional models. You can train a general model and then tweak it so that it can be applied in different settings. For example, I designed a classifier for predicting X but then released it to three teams that had different requirements (one was sensitive to false positives, another to false negatives and the third team was a legal team that wanted to explore the liability cost of different models).

Tuning of a classifier is a fairly advanced technique even a few months ago. However, scikit-learn has come up with a TunedThresholdClassifier in May 2024 that makes this easy. In this assignment, you will explore threshold tuning.

Specifically, I will list 5 good resources for threshold tuning. It is possible that there are other good resources too. You can mix and match ideas from these list to tune the classifier. 10 points will be for whether your code works and 10 points will be awarded based on the thoughtfulness of the tuning.

(a) https://towardsdatascience.com/tune-in-decision-threshold-optimization-with-scikit-learns-tunedthresholdclassifiercv-7de558a2cf58
(b) https://towardsdatascience.com/achieve-better-classification-results-with-classificationthresholdtuner-39c5d454637e
(c) https://www.geeksforgeeks.org/how-to-use-scikit-learns-tunedthresholdclassifiercv-for-threshold-optimization/
(d) https://machinelearningmastery.com/threshold-moving-for-imbalanced-classification/
(e) https://towardsdatascience.com/fine-tuning-a-classifier-in-scikit-learn-66e048c21e65

Submission Instructions

When submitting, create a folder DS5612_PA4 folder. You have to give it as a pyproject (either rye or uv) so that I can install everything with rye sync or uv sync. For each task, there should be a single script (Python or Shell) that I can run to generate the output and see the results. Any increase in complexity will see a proportional decrease in your score :)

Please submit your solution in the format id1_id2_id3_PA4.zip where id1, id2, id3 are the IIT-PK ids of the students. All students in the team have to use the same ordering. For example, student 2 should NOT name his/her file as id2_id1_id3_PA4.zip.

- Delete the .git, .pdm-build, .venv, .ruff_cache, .pytest_cache folders from project root
- Delete the .gitignore, .python-version from project root.
- Delete the __pycache__ file from src/ds5612_pa2 and src/ds5612_pa2/code.
- Delete the autogluon folder.