Assignment 1

Working Code is available in *github*<*https://github.com/rajs1006/DataScience*> repository and also attached as *ZIP*.

Code is written using **Python** Orchestrated using **Dagster** and deployed using **Docker** and **GitHub Actions.**

—----------

How would you assess if a code is ready for production? What do you think about the given code in model.ipynb?

Few things I noticed in model.ipynb:

1. Historical Data is split into *Train*(training set) and *Test*(holdout set), but they are transformed using the **fit_transform** method of ***StandardScaler().***
    a. Ideally only *Train* data should be *fit_transform*
    b. *Test* data should just be *transformed*

    It gives us more realistic results if *Transformation* is applied after *Split*. Also, *shuffle* split could give better representation of model performance.

    ```
    X_train, X_test, y_train, y_test = train_test_split(
       X, y, test_size=0.3, random_state=99
    )
    transformModel =  StandardScaler()
    X_train = transformModel.fit_transform(X_train)
    X_test = transformModel.transform(X_test)
    ```

2. I would use the Sklearn's **pipeline** for the classifier rather than using the classifier directly. That applies the transformation on *inference* data as well.
3. Inference data is ***not*** *transformed* using ***StandardScaler(),*** but the model is trained on *transformed* data which will result in *Inconsistent representations* and *Performance degradation.*

In general, code does not seem ready for the Production,

- **Description of data/features** are not provided and so it may be difficult for ML Engineers to include the data source and apply the necessary transformations.
- The **Technical** as well as **Business KPI's** are not mentioned and so it's difficult to know if Model is fulfilling its purpose(Only model accuracy is not enough to go to production).

Would you change anything in the code? How and why?

- ***Hyperparameter tuning*** could improve the results.

- Some other Parametric models like Tree, Bagging based models could help understand the feature importance.
- I would spend little time finding the ***explanation*** of the model results.

How would you rearrange code for the production system?
I would rearrange the code a little bit.(functional code is provided)
- A separate transformation function
- Separate the action of Training, Evaluation and Inference.

Discuss training and inference data provided
- Training data seem non-linearly separable.
- Inference data seem to be from the same distribution as of training data.

Data Sources:
- ❖ Data can be delivered from multiple sources and so in this case I would go for a feature-store structure. Which can collect data from multiple sources and transform them specific for modeling and keep them organized in one source(cache or db or files).
- ❖ Having a feature store is beneficial for reproducible results and models could detach them themselves from the source of data.
- ❖ Using proper storage for feature-store can enrich the model triggering, which can be constant pooling or event triggered based on use case.

How would you organize the codebase management. E.g. how do you deliver code to the production environment?

- Code should be able to be containerized(using docker or any other tool).
- Code should be able to be orchestrated, separate action flows for Training, Evaluation and Inference to enable **Continuous Evaluation** and **Continuous Training**.
- There are 2 ways to separate the model, as per below:
    - Have one repository per use case but different models but have an *Orchestration* tool to control the interdependent runs and Schedules.
        - Single repository per use case will help to bring down the **COST** and **Maintenance Effort.**
        - There could be multiple models per use case, they can stick together and orchestrated together.(*Output* of one model as *Input* of another model)
        - The schedules and triggers can be managed effectively in a *cohesive* model environment.
    - Have a separate repository for every model.
        - Separate **CI/CD** flow maintaining **microservice** architecture
        - If orchestration is needed, a universal *Orchestration* tool can be utilized to directly use the Containers for orchestrating.
- Save the final result in a **Permanent Storage** for better **monitoring** and **Visualization.**

Optional part

Would you consider any alternatives to the used algorithm and if so, why?

Current GaussianProcessClassifier algorithm is a non parametric algorithm which looks like a good choice here as data is not linearly separable. However, I would also try out *Tree-based* algorithms as that would give me as they are *faster* and more *explainable*.

Discuss pros and cons of your chosen algorithms as well as evaluation metrics.

GaussianProcessClassifier is a bit of a slow algorithm but I think that should not be a problem in this particular use case.
Accuracy might not be a very useful evaluation matrix in this particular case as here it is more important to identify the classes which require the maintenance rather than the one which does not require maintenance.

- The system that requires maintenance should be identified correctly as nearly 100% accuracy.
- Looking at **sensitivity/Recall** would make more sense as we do not want to miss the systems that need maintenance.

Discuss Big O notation of the algorithms, its algorithmic and memory complexity.

For GaussianProcessClassifier **O(Nd)** will be the complexity given **d** as dimension and **N** as number of dataset.

List libraries in various programming languages that you know or have used that implement those algorithms.

GaussianProcessClassifier : SKLearn : Python
KDependentBayesClassifier : Java-ML : Java

Would your setup change if these were Databricks notebooks with Spark compute clusters behind them, written in PySpark?

I will probably use **MLLib** which directly works on distributed data.