Building, training, deploying, and monitoring ML models on the Cloud: An introduction to Amazon SageMaker

Meher Satya Swaroop Malina
Boston University, swaroopm@bu.edu

1. Abstract: Amazon SageMaker is a comprehensive machine learning (ML) cloud service offered by Amazon Web Services (AWS) that is designed to streamline and democratize the process of building, training, deploying, and managing machine learning models with fully managed infrastructure, tools, and workflows. It plays a pivotal role in the field of machine learning and artificial intelligence (AI), offering a robust platform for business analysts, data scientists and ML engineers to develop and deploy machine learning solutions. Key features of Amazon SageMaker include a wide array of pre-built algorithms, support for popular frameworks like TensorFlow and PyTorch, an integrated development environment, and AutoML. With Amazon SageMaker, organizations can unlock the full potential of machine learning, accelerating the development cycle, reducing operational costs, and driving innovation across a wide range of industries, from healthcare to finance and beyond.

2. Role in Machine Learning and Artificial Intelligence

Amazon SageMaker serves a critical role in advancing the capabilities of machine learning and artificial intelligence. Its core functions can be summarized as follows:

Data Preparation and Feature Engineering: SageMaker provides tools for data preparation and feature engineering, which are essential steps in building machine learning models. This includes data labeling, data cleaning, standardization, one-hot encoding of categorical variables, feature selection and any data preparation step that can be coded in Jupyter notebooks. SageMaker also offers a powerful data preparation and feature engineering solution called SageMaker Data Wrangler [1], which simplifies the process of getting data ready for machine learning.

Model Building and Training: SageMaker provides tools for building and developing machine learning models, making it easier for data scientists and developers to create predictive models without the need to write complex code from scratch and without worrying about the infrastructure. It also provides pre-trained models to start with in SageMaker JumpStart.

<u>SageMaker Studio Notebooks:</u> With SageMaker Studio, you can create Jupyter notebooks to build and train machine learning models. These notebooks are fully integrated into SageMaker, providing easy access to datasets stored in Amazon S3, managed training jobs, and real-time monitoring. You can leverage popular libraries like TensorFlow [2], PyTorch [3], and Scikit-learn within your notebooks. SageMaker Studio Notebooks are excellent for data scientists and developers who prefer a more hands-on approach to model building.

<u>No-Code Solutions</u>: For users who are not well-versed in traditional coding and want to build models without writing code, SageMaker offers no-code solutions like SageMaker Autopilot [4] and SageMaker Canvas [5].

<u>SageMaker Autopilot</u>: It automatically explores and builds the best model for your data. You provide the dataset, and SageMaker Autopilot takes care of data preprocessing, feature engineering, model selection, and hyperparameter tuning, all without any coding required. It is an excellent choice for users who want to quickly develop machine learning models without extensive coding knowledge.

Model Deployment: SageMaker enables users to deploy trained models as endpoints for real-time inference, facilitating the integration of machine learning predictions into applications, websites, and services.

SageMaker Endpoints: SageMaker allows for real-time model deployment through SageMaker Endpoints. Developers can easily deploy models as RESTful APIs and integrate them into applications.

SageMaker Batch Transform: For batch processing, SageMaker offers Batch Transform, enabling you to process large volumes of data with your trained models efficiently.

SageMaker Multi-Model Endpoints: This service optimizes resource usage by hosting multiple models on a single endpoint, reducing costs, and improving scalability.

SageMaker Hosting Containers: Developers can use SageMaker Hosting Containers to deploy custom models and custom inference logic in Docker containers.

CI/CD Pipelines: SageMaker integrates with CI/CD tools like SageMaker Pipelines, making it straightforward to build end-to-end ML pipelines. This simplifies development, testing, and deployment, enabling a seamless, automated process from code to production.

Model Monitoring: The platform includes features for continuous model monitoring, which is crucial for ensuring that deployed models remain accurate and reliable over time.

SageMaker Model Monitor: Ensuring model quality post-deployment is crucial. SageMaker Model Monitor automatically detects data and concept drift, making it easier for MLOps engineers to maintain model performance.

Amazon CloudWatch Integration: SageMaker Model Monitor seamlessly integrates with Amazon CloudWatch, enabling you to set up custom alarms and notifications based on monitoring results. This ensures timely responses to deviations in model behavior.

Custom Metrics and Dashboards: You can set up custom metrics and dashboards to gain deeper insights into model performance and share those insights with stakeholders.

3. Significance in Simplifying the End-to-End Machine Learning Process

Amazon SageMaker significantly streamlines the end-to-end machine learning process in several ways:

Scalability: SageMaker offers the capability to scale resources on-demand, allowing users to train and deploy models with ease, regardless of the dataset's size or complexity.

Integration: It seamlessly integrates with other AWS services, such as S3, Lambda, and API Gateway, to create a complete ecosystem for developing and deploying machine learning solutions.

Automation: SageMaker provides AutoML capabilities through SageMaker Autopilot and SageMaker Canvas, enabling a no-code approach to model development, automating data preprocessing, feature engineering, model selection, and hyperparameter tuning.

Customization: For users who require fine-grained control and customization, SageMaker allows code-based model development using popular machine learning frameworks like TensorFlow, PyTorch, and XGBoost.

4. Using Amazon SageMaker for Model Building, Training, Deployment, and Monitoring

4.1 Dataset Overview

In this section, we provide a comprehensive overview of the dataset used to showcase the capabilities of Amazon SageMaker. The dataset, titled "On-time-delivery-data.csv" [6], was sourced from the Amazon SageMaker datasets. This dataset serves as the foundation for building and training a predictive model using Amazon SageMaker to determine the likelihood of delivery being on time or delayed.

- The dataset comprises a total of 45,000 rows, indicating a substantial volume of data for analysis.
- It is structured with 16 columns, where each column represents a specific feature or attribute.

The dataset contains a mix of numerical and categorical features, which are pivotal for our predictive modeling task. Specifically, there are 14 numeric columns and 2 categorical columns. Below is a breakdown of the feature types:

Numeric Columns (14): total_items, precipitation_rate, water_runoff, snow_depth, temperature, temperature_at_1500m, min_temperature, max_temperature, pressure, wind_gust_speed, total_cloud_cover, dew_point_temperature, relative_humidity, wind_speed

Categorical Columns (2): zipcode, classification_ontime (Target Column)

4.2 Code Approach: Amazon SageMaker Studio Notebooks

All the below sections were executed in the Jupyter notebook provided by the Amazon SageMaker Studio with ml.t3.medium instance.

4.2.1 Data Import

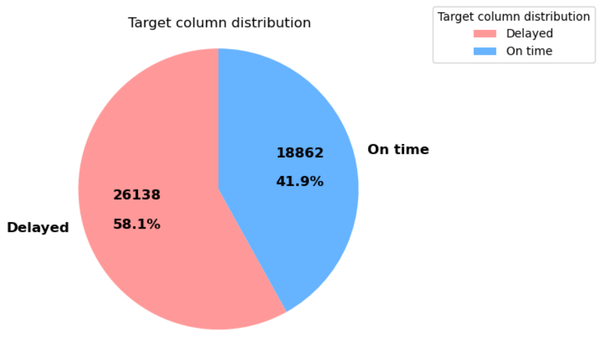
We initiated the data import process by retrieving the dataset from our S3 bucket and transferring it to the SageMaker Studio notebook environment by a simple API call. This facilitated easy access and manipulation of the data.

4.2.2 Exploratory Data Analysis (EDA)

To gain a comprehensive understanding of the dataset, we conducted Exploratory Data Analysis, which included the following steps:

Feature Exploration: We examined the dataset's features to understand their characteristics, types, and distributions.

Label Distribution: A pie chart was generated to visualize the distribution of the target labels, helping us assess class balance or imbalance.



Null Value Check: We checked for missing or null values within the dataset, an essential step in data quality assessment.

4.2.3 Data Preprocessing:

In preparation for model building, we applied several data preprocessing techniques to enhance the dataset's suitability for machine learning. These steps included:

Cardinality Reduction: We aggregated zip codes based on their first digits, effectively reducing the cardinality of the feature, which can help improve model efficiency.

Feature Selection: To identify the most relevant features for our model, we employed a feature importance metric derived from a Random Forests model. This technique helped us focus on the most influential variables while eliminating noise.

Standardization: We standardized the numeric columns to ensure that all features had a mean of 0 and a standard deviation of 1. Standardization is crucial for models that are sensitive to feature scaling.

Data Split: To facilitate training, validation, and testing of our model, we divided the preprocessed dataset into three distinct portions: a 70% training set, a 20% validation set, and a 10% test set. This division allowed us to assess our model's performance under various conditions and prevent overfitting.

- Train dataset count: 32400 samples
- Validation dataset count: 8100 samples
- Test dataset count: 4500 samples

Preprocessing Function: We encapsulated the entire data preprocessing pipeline into a dedicated function, which was called individually for each of the training, validation, and test datasets. This approach ensured consistency and reproducibility across different dataset splits.

4.2.4 Model Building

The decision to employ XGBoost as the primary model for this project is grounded in its exceptional performance and versatility, making it a preferred choice among data scientists and machine learning practitioners. XGBoost, short for Extreme Gradient Boosting, is renowned for the following characteristics:

High Predictive Accuracy: XGBoost consistently delivers competitive results in a wide range of predictive modeling tasks, including classification and regression. Its ability to handle complex, nonlinear relationships in the data is a substantial advantage.

Ensemble Learning: XGBoost is an ensemble learning algorithm that combines the predictions of multiple weak learners (decision trees) to create a strong, robust model. This ensemble approach often leads to improved model performance.

Regularization Techniques: It incorporates L1 (Lasso) and L2 (Ridge) regularization techniques to prevent overfitting, making it resilient to noise and outliers in the data.

Parallel and Distributed Computing: XGBoost is designed for efficiency, with the ability to utilize parallel and distributed computing resources. This speeds up model training, especially for large datasets.

4.2.5 Hyperparameter Tuning

To maximize the performance of our XGBoost model, we conducted hyperparameter tuning using grid search and cross-validation. We explored a range of hyperparameter combinations to identify the optimal configuration. The hyperparameters we tuned included:

min_child_weight: Controls the minimum sum of instance weight needed in a child. We tested values of 1, 5, and 10.

gamma: Governs the regularization term on the tree. We examined values of 0.5, 1, 1.5, 2, and 5.

subsample: Determines the fraction of samples used for tree building. We explored values of 0.6, 0.8, and 1.0.

colsample_bytree: Defines the fraction of features used for tree building. We tested values of 0.6, 0.8, and 1.0.

max_depth: Specifies the maximum depth of the tree. We considered values of 3, 4, and 5.

4.2.6 Model Training

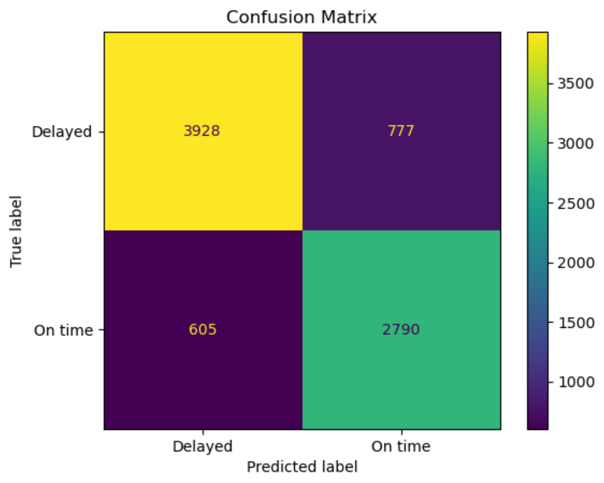
For the model training, we used a stratified k-fold cross-validation strategy with 3 splits to ensure robust evaluation. The best hyperparameters were identified through this process, and the final XGBoost classifier was created using these optimal settings.

4.2.7 Model Evaluation

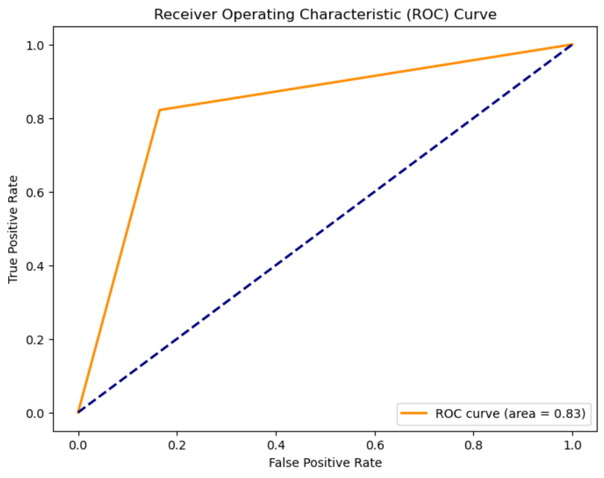
The performance of our XGBoost model was rigorously assessed on the validation dataset, yielding the following key metrics:

On Validation data set		
XGBoost	**Accuracy**	0.83
	Precision	0.78
	Recall	0.82
	F1 Score	0.8
	AUC	0.83

Evaluation metrics of the model with the best hyperparameter obtained by the grid search



Confusion matrix using the predicted probabilities given by the best model on the validation data set



ROC Curve using the predicted probabilities given by the best model on the validation data set

4.2.8 Model Deployment

Model deployment is a crucial step in the machine learning workflow, where a trained model is made accessible for making predictions on new data. In this section, we discuss the process of deploying our best-performing XGBoost model using Amazon SageMaker [7].

5.1 Model Deployment Process

The model deployment process involved the following steps:

Model Selection and Artifact Location: After training and hyperparameter tuning, we selected the best-performing XGBoost model.

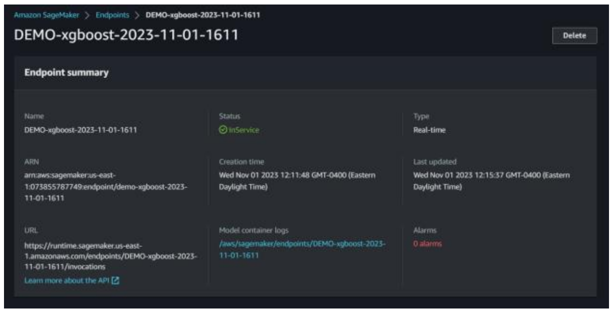
The trained model was stored in an S3 location (model_path) for easy accessibility and deployment.

Choice of SageMaker Container: To deploy the XGBoost model, we utilized the SageMaker XGBoost container (container). This container is designed to host XGBoost models and allows for straightforward deployment.

Initializing the XGBoost Estimator: We created an XGBoost Model object with the necessary configuration, including the model data location, container image, and the IAM role.

Data Capture Configuration: For monitoring and capturing data during endpoint invocations, a DataCaptureConfig object was configured. This allowed us to capture both request and response data [8].

Deploying the Model: The XGBoost model was deployed using the deploy method. The model was hosted on a SageMaker endpoint with an initial instance count of 1 and an instance type of "ml.m4.xlarge."



Amazon SageMaker > Endpoints > DEMO-xgboost-2023-11-01-1611

DEMO-xgboost-2023-11-01-1611

Endpoint summary

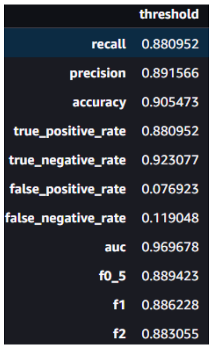
Name DEMO-xgboost-2023-11-01-1611	Status InService	Type Real-time
ARN arn:aws:sagemaker:us-east-1:073855787749:endpoint/demo-xgboost-2023-11-01-1611	Creation time Wed Nov 01 2023 12:11:48 GMT-0400 (Eastern Daylight Time)	Last updated Wed Nov 01 2023 12:15:37 GMT-0400 (Eastern Daylight Time)
URL https://runtime.sagemaker.us-east-1.amazonaws.com/endpoints/DEMO-xgboost-2023-11-01-1611/invocations Learn more about the API	Model container logs /aws/sagemaker/endpoints/DEMO-xgboost-2023-11-01-1611	Alarms 0 alarms

Screenshot of the deployed endpoint in service

4.2.9 Model Monitoring

Model monitoring is a critical component of maintaining machine learning models and ensuring their continued accuracy and reliability. In this section, we delve into the process of implementing model monitoring using Amazon SageMaker [9].

We initiated a ModelQualityMonitor using SageMaker, specifying parameters such as the IAM role, instance count, instance type, volume size, and maximum runtime. This ensured that the monitoring process was appropriately configured. To monitor the model's performance, we created a baseline using a test dataset. The dataset, stored in S3, was used as a reference to measure the model's predictions and to detect potential drift. The SageMaker ModelQualityMonitor was utilized to suggest the baseline. This step involved configuring various settings, including the problem type (in our case, 'BinaryClassification') and specifying the columns for inference, probabilities, and ground truth labels.



	threshold
recall	0.880952
precision	0.891566
accuracy	0.905473
true_positive_rate	0.880952
true_negative_rate	0.923077
false_positive_rate	0.076923
false_negative_rate	0.119048
auc	0.969678
f0_5	0.889423
f1	0.886228
f2	0.883055

Threshold values generated by the baseline job

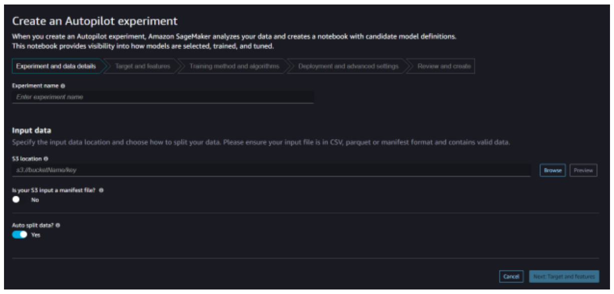
4.3 No-Code Approach: Amazon SageMaker Autopilot

Amazon SageMaker offers a no-code approach to machine learning through SageMaker Autopilot, a powerful tool that automates the end-to-end machine learning process. In this section, we will explore how SageMaker Autopilot was utilized for feature engineering, data preprocessing, model selection, evaluation, and deployment using the provided dataset.

The entire Autopilot experiment was run on ml.r16.large instance which was allocated automatically by the Amazon SageMaker Autopilot.

4.3.1 Data Import

Data import in Autopilot is as simple as clicking a button. It supports various data formats such as CSV, parquet, and manifest format, making it flexible for different types of datasets. Users can upload their data to Amazon S3 and specify the S3 URI. Users can split the data by specifying the split ratio manually or can leave the auto split toggle on. We have imported our dataset "On-time-delivery-data.csv" from an S3 bucket.



Create an Autopilot experiment

When you create an Autopilot experiment, Amazon SageMaker analyzes your data and creates a notebook with candidate model definitions. This notebook provides visibility into how models are selected, trained, and tuned.

Experiment name: Enter experiment name

Input data

Specify the input data location and choose how to split your data. Please ensure your input file is in CSV, parquet or manifest format and contains valid data.

S3 location: s3://bucketName/key

Is your S3 input a manifest file? No

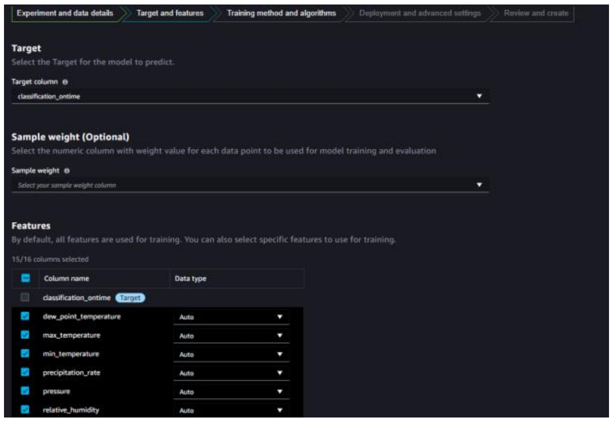
Auto split data? Yes

Experiment and data details
Target and features
Training method and algorithms
Deployment and advanced settings
Review and create
Browse
Preview
Cancel
Next: Target and features

Screenshot of the Amazon SageMaker Autopilot Interface

4.3.2 Target and features

Next, we select the target column and either select the features and their datatype manually or leave it to the autopilot. In our approach, we identify the 'classification_ontime' as the target column and opt for the inclusion of all available features, with a specific modification to 'zipcode,' which is reclassified as a categorical variable during the implementation process.



Experiment and data details > Target and features > Training method and algorithms > Deployment and advanced settings > Review and create

Target
Select the Target for the model to predict.

Target column: classification_ontime

Sample weight (Optional)
Select the numeric column with weight value for each data point to be used for model training and evaluation

Sample weight: Select your sample weight column

Features
By default, all features are used for training. You can also select specific features to use for training.

15/16 columns selected

	Column name	Data type
☐	classification_ontime (Target)	
✓	dew_point_temperature	Auto
✓	max_temperature	Auto
✓	min_temperature	Auto
✓	precipitation_rate	Auto
✓	pressure	Auto
✓	relative_humidity	Auto

Screenshot of the Amazon SageMaker Autopilot Interface

4.3.3 Model selection

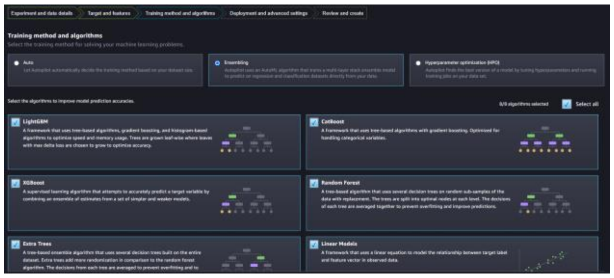
We then select the training method and algorithms. Amazon SageMaker Autopilot provides three training methods.

<u>Auto:</u> This method allows Autopilot to automatically select an appropriate training method based on the problem type (regression or classification) and the dataset characteristics. It provides a quick and efficient way to build a machine learning model without the need for manual algorithm selection.

<u>Ensembling:</u> The ensembling method, our chosen approach for this problem, leverages the power of combining multiple algorithms to enhance model performance. Ensembles are known for their robustness and ability to reduce bias and variance, often resulting in more accurate predictions. However, it is worth noting that ensembling typically requires more computational resources compared to other methods.

<u>Hyperparameter Optimization (HPO):</u> HPO involves fine-tuning the hyperparameters of individual algorithms to optimize their performance. It can be a computationally intensive process as it runs a multitude of experiments with various hyperparameter combinations, potentially utilizing significant resources, such as the ml.r16.large instance.

For our specific problem, we opted for the ensembling method, recognizing that HPO could be costly in terms of time and resources. Within the ensembling approach, we harnessed the potential of several available algorithms, including LightGBM, CatBoost, XGBoost, Random Forest, Extra Trees, Linear Models, and neural networks. By combining these diverse algorithms in Autopilot, we aimed to maximize the model's predictive capabilities and overall performance. This strategy provides a well-rounded and robust solution for our machine learning problem, taking advantage of the strengths of various algorithms while mitigating their individual weaknesses.



Experiment and data details > Target and features > Training method and algorithms > Deployment and advanced settings > Review and create

Training method and algorithms
Select the training method for solving your machine learning problems.

Auto — Let Autopilot automatically decide the training method based on your dataset size.
[✓ Ensembling] — Autopilot uses an AutoML algorithm that trains a multi-layer stack ensemble model to predict on regression and classification datasets directly from your data.
Hyperparameter optimization (HPO) — Autopilot finds the best version of a model by tuning hyperparameters and running training jobs on your data set.

Select the algorithms to improve model prediction accuracies. 8/8 algorithms selected ✓ Select all

✓ **LightGBM** A framework that uses tree-based algorithms, gradient boosting, and histogram-based algorithms to optimize speed and memory usage. Trees are grown leaf-wise where leaves with max delta loss are chosen to grow to optimize accuracy.

✓ **CatBoost** A framework that uses tree-based algorithms with gradient boosting. Optimized for handling categorical variables.

✓ **XGBoost** A supervised learning algorithm that attempts to accurately predict a target variable by combining an ensemble of estimates from a set of simpler and weaker models.

✓ **Random Forest** A tree-based algorithm that uses several decision trees on random sub-samples of the data with replacement. The trees are split into optimal nodes at each level. The decisions of each tree are averaged together to prevent overfitting and improve predictions.

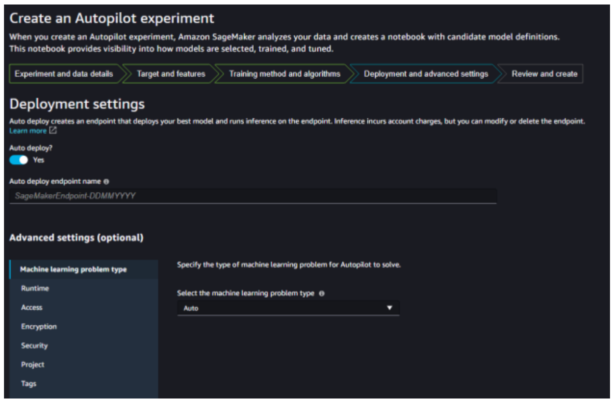
✓ **Extra Trees** A tree-based ensemble algorithm that uses several decision trees built on the entire dataset. Extra trees add more randomization in comparison to the random forest algorithm. The decisions from each tree are averaged to prevent overfitting and to

✓ **Linear Models** A framework that uses a linear equation to model the relationship between target label and feature vector in observed data.

Screenshot of the Amazon SageMaker Autopilot Interface

4.3.4 Model Deployment

Amazon SageMaker Autopilot allows us to deploy the model in just one click. Users can either select auto-deploy before running the experiment or deploy once the best model is trained. In our implementation, we have not deployed the model using Autopilot because of limited resources and budget. More on how to deploy using SageMaker Autopilot can be found here [10].



Create an Autopilot experiment

When you create an Autopilot experiment, Amazon SageMaker analyzes your data and creates a notebook with candidate model definitions. This notebook provides visibility into how models are selected, trained, and tuned.

Experiment and data details > Target and features > Training method and algorithms > Deployment and advanced settings > Review and create

Deployment settings

Auto deploy creates an endpoint that deploys your best model and runs inference on the endpoint. Inference incurs account charges, but you can modify or delete the endpoint. Learn more

Auto deploy? Yes

Auto deploy endpoint name: SageMakerEndpoint-DDMMYYYY

Advanced settings (optional)

Machine learning problem type | Runtime | Access | Encryption | Security | Project | Tags

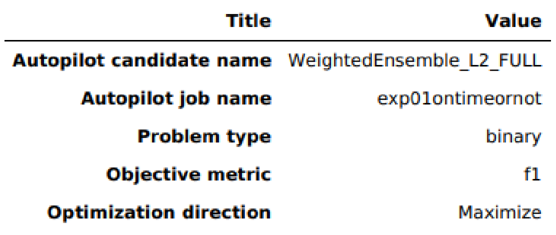
Specify the type of machine learning problem for Autopilot to solve.

Select the machine learning problem type: Auto

Screenshot of the Amazon SageMaker Autopilot Interface

4.3.5 Results and Reports

Upon the completion of the Amazon SageMaker Autopilot experiment, a comprehensive set of results and reports became available, offering valuable insights into the model building process and its outcomes. The experiment generated key metrics, including F1 score, AUC, precision, recall, and logloss, for each of the models trained during various trials. These metrics provided a holistic view of the models' performance and their suitability for the given machine learning task.



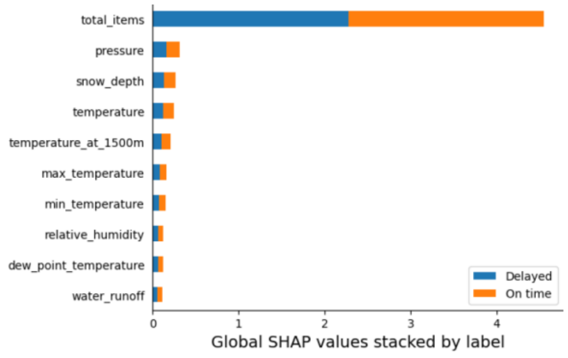
Title	Value
Autopilot candidate name	WeightedEnsemble_L2_FULL
Autopilot job name	exp01ontimeornot
Problem type	binary
Objective metric	f1
Optimization direction	Maximize

Amazon SageMaker Autopilot job details

Furthermore, Autopilot produced two essential types of reports: the Explainability Report and the Performance Report.

Explainability Report: The Explainability Report delved into the inner workings of each ensemble model. It offered crucial insights into model interpretability, featuring the following components:

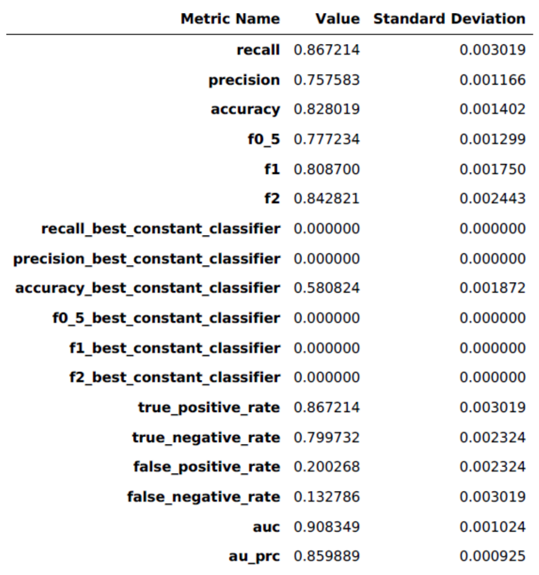
- Feature Importance: This section provided an assessment of the importance of individual features in the model's decision-making process, shedding light on which variables had the most significant impact on predictions.



- Model Parameters: The report detailed all the parameters employed in the model construction, allowing for a comprehensive understanding of the model's configuration.
- Model Types: It also outlined the specific machine learning algorithms used in the ensemble model, elucidating the combination of techniques that contributed to the final model's performance.

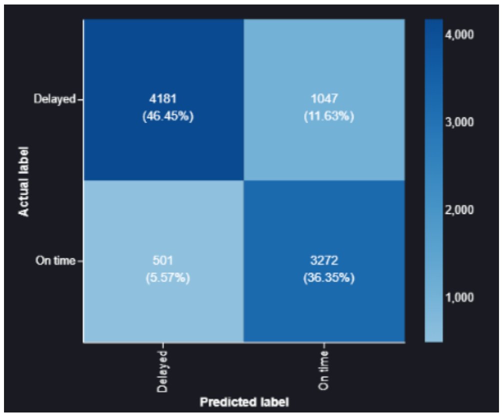
Performance Report: The Performance Report served as a comprehensive evaluation of the model's predictive capabilities, offering a range of metrics and visualizations, including:

- Performance Metrics: This section encompassed a variety of metrics, including accuracy, precision, recall, and F1 score, enabling a robust assessment of the model's overall performance.

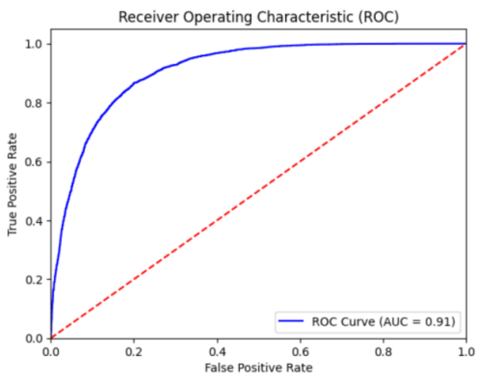


Metric Name	Value	Standard Deviation
recall	0.867214	0.003019
precision	0.757583	0.001166
accuracy	0.828019	0.001402
f0_5	0.777234	0.001299
f1	0.808700	0.001750
f2	0.842821	0.002443
recall_best_constant_classifier	0.000000	0.000000
precision_best_constant_classifier	0.000000	0.000000
accuracy_best_constant_classifier	0.580824	0.001872
f0_5_best_constant_classifier	0.000000	0.000000
f1_best_constant_classifier	0.000000	0.000000
f2_best_constant_classifier	0.000000	0.000000
true_positive_rate	0.867214	0.003019
true_negative_rate	0.799732	0.002324
false_positive_rate	0.200268	0.002324
false_negative_rate	0.132786	0.003019
auc	0.908349	0.001024
au_prc	0.859889	0.000925

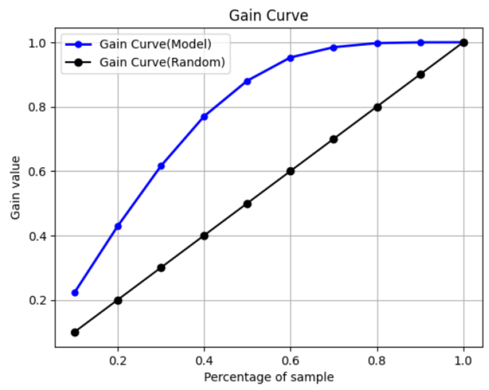
- Confusion Matrix: The confusion matrix provided a breakdown of true positives, true negatives, false positives, and false negatives, giving insights into the model's classification performance.



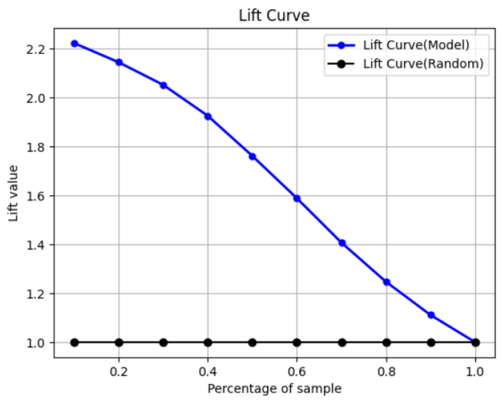
- ROC Curve: The Receiver Operating Characteristic (ROC) curve illustrated the trade-off between true positive rate and false positive rate, helping assess model performance across different thresholds.



- Gain Curve: This curve elucidated how the model's predictions compared to random guessing, providing insights into its effectiveness.



- Lift Curve: The lift curve demonstrated the model's ability to enhance the precision of predictions, offering a practical perspective on its utility for decision-making.



5. Pros and Cons of Amazon SageMaker

5.1 Pros

Scalability and Flexibility: SageMaker enables effortless scaling of computing resources, allowing data scientists to handle projects of varying sizes without the complexities of resource management.

Seamless Integration: SageMaker seamlessly integrates with other AWS services, including Amazon S3, Lambda, and API Gateway, providing a holistic ecosystem for machine learning solutions. This integration simplifies data storage, data processing, and model deployment.

End-to-End Platform: SageMaker offers an end-to-end platform for machine learning, covering data preparation, model development, training, deployment, and monitoring. This streamlining reduces the need for using multiple tools or services.

Efficiency: SageMaker's automation features, including SageMaker Autopilot, reduce the time and effort required for model development and tuning, accelerating project timelines.

Cost-Efficiency: The pay-as-you-go pricing model ensures that users only pay for the resources they use, making machine learning projects cost-effective.

No-Code Approach: SageMaker Autopilot simplifies machine learning for users without extensive coding experience. It automates tasks such as data preprocessing, feature engineering, model selection, and hyperparameter tuning.

5.2 Cons

Data Privacy and Security: For organizations with stringent data privacy and security requirements, handling sensitive data within SageMaker may necessitate additional precautions.

Customization Complexity: While SageMaker provides customization options, complex or highly specialized machine learning models may require more effort to develop and deploy compared to fully custom solutions.

6. Conclusion

In this report, we explored majority of the capabilities and the impact of Amazon SageMaker in the realm of machine learning and artificial intelligence. The study encompassed the use of SageMaker in building, training, deploying, and monitoring machine learning models, with a focus on a binary classification problem using a dataset comprising 45,000 rows and 16 columns.

The key findings and takeaways from this report can be summarized as follows:

Amazon SageMaker Simplifies Machine Learning: SageMaker streamlines the end-to-end machine learning process, offering a comprehensive platform that caters to data preparation, model development, model training, deployment, and monitoring.

No-Code Approach with SageMaker Autopilot: The no-code approach, exemplified by SageMaker Autopilot, demonstrates the potential for automating various tasks, such as feature engineering, data preprocessing, model selection, and evaluation. This approach expedites the machine learning project lifecycle and lowers the barrier to entry for users with varying degrees of machine learning expertise.

7. References

[1] Data Wrangler: https://docs.aws.amazon.com/sagemaker/latest/dg/data-wrangler.html

[2] TensorFlow: https://www.tensorflow.org/

[3] PyTorch: https://pytorch.org/

[4] SageMaker Autopilot: https://docs.aws.amazon.com/sagemaker/latest/dg/autopilot-automate-model-development.html

[5] SageMaker Canvas: https://aws.amazon.com/sagemaker/canvas/

[6] Data set: https://aipackagetracker.awsplayer.com/downloadable/On-time-delivery-data.csv

[7] https://docs.aws.amazon.com/sagemaker/latest/dg/realtime-endpoints-deployment.html

[8] https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor-data-capture-endpoint.html

[9] https://docs.aws.amazon.com/sagemaker/latest/dg/model-monitor-model-quality-baseline.html

[10] https://docs.aws.amazon.com/sagemaker/latest/dg/autopilot-deploy-models.html