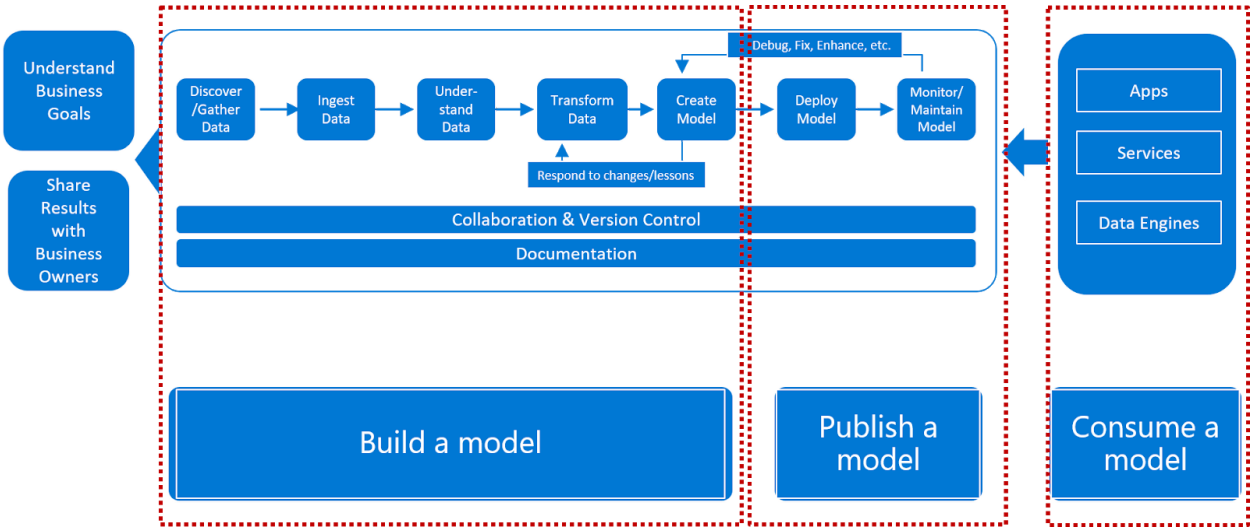
A Quantitative and Qualitative Comparison Across Cloud Platforms on Sentiment Analysis Application

Kefei Wu (kw2669) and Sonal Sharma (ss13449@nyu.edu)

Abstract:
The emergence of cloud computing has made it easier to train and deploy Machine Learning models. Sentiment Analysis is very widely used in industry and is memory intensive. We have used IMDB dataset to test how Machine Learning Services perform, across cloud provider platforms - AWS Sagemaker and Microsoft Azure Machine Learning. We use these services to train, test, and deploy our customized classification model into endpoints, and compare these services along performance, ease of integration and usability, cost, and time.

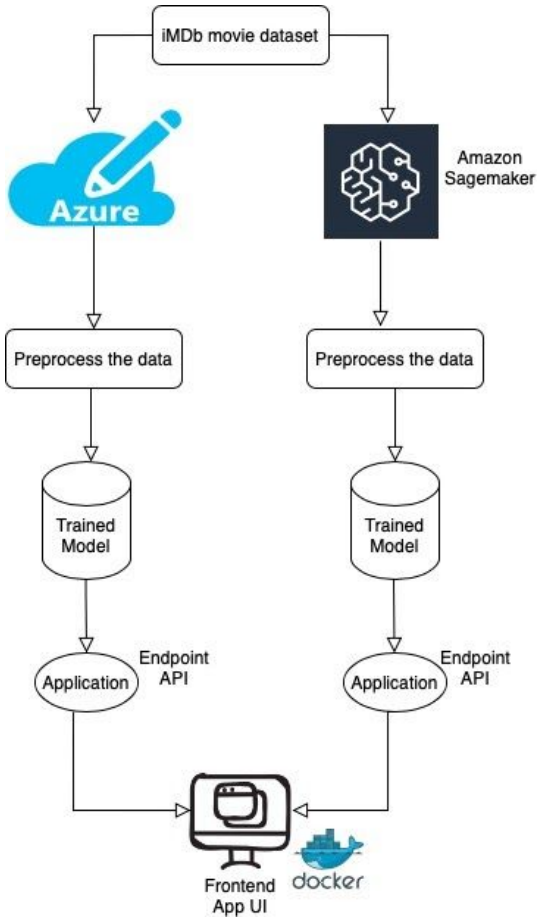
Background
Platform as a service (PaaS) is a complete development and deployment environment in the cloud, with resources for cloud-based to cloud-enabled enterprise applications. Machine learning as a service (MLaaS) covers infrastructure issues such as data pre-processing, model training, and model evaluation, and prediction. Here, deployment can simply means to create/build a machine learning model and integrate it into an existing production environment where it can take in an input and return an output and we can make the predictions from a trained ML model which are also available to others.



Data Preprocessing
In this project, we use the IMDB movie review dataset for binary sentiment classification. Our raw data contains reviews, the text labelled sentiment. The first step is to clean the text by data tokenization, removal of words and performing Stemming. Data Tokenization is used to perform separation in the sentence into words, phrases or other components called tokens to remove words that do not impact the analysi. Stemming was used to alter the form of a word to its root or base form. After Sentiment classification is done on text - it will classify reviews into two classes; positive, negative.

	Review	Sentiment
Count	50000	50000
Unique	49582	2
Top	Loved today's show!!!...	Positive
Freq	5	25000

Project Outline



In our study, we perform the following steps across the two platforms whose specifics of each will be explained in sometime. Project mostly focuses on how to deploy models in production.

1. Download Data
2. Preprocess & Train the model on cloud platforms like MS Azure, and AWS Sagemaker
3. Create web endpoints for each of it which can be accessed from its cloud provided API
4. Build centralised Front-end Flask app to submit text reviews instantly on each cloud web api.

The four major cloud providers AWS, Azure, GCP, IBM nowadays all provide both machine learning PaaS to make the CI/CD process easier. Most cloud providers also provide built-in models in their platform except Azure. However, Azure takes advantage in Machine learning as a Service, it provides more ready to use services and APIs.

Customized Model

Typical models to solve a natural language processing problem include Bag of Words and Word2vec. In our project, we first tried to create BoW using CountVectorizer and classified sentiment with Naive Bayes,and for Word2vec, we use Random Forest as the Classifier. We also tried Tfidf vectorizer and used Logistic regression.

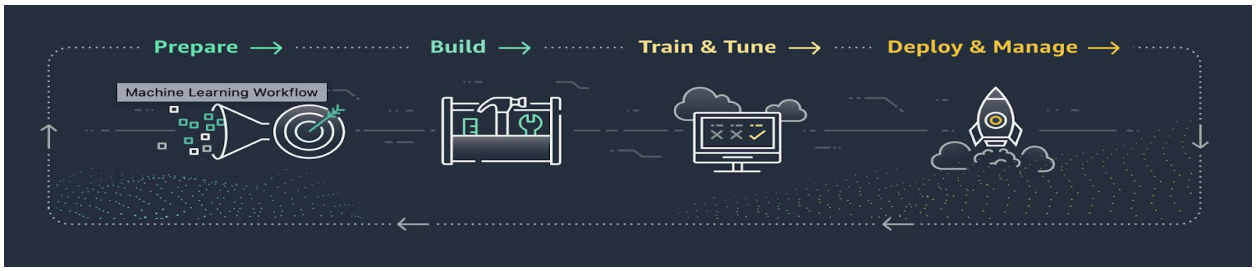
Since LSTM is designed in a way such that they can catch the sequential/time series data, is widely used today for a variety of different tasks like speech recognition, text classification and sentiment analysis, So in our project, we use keras to tokenize the review data and trained a simple LSTM model (embedding layer => LSTM layer => dense layer). Here are all the models we trained in our project in each platform. Since LSTM achieves best accuracy, we decided to deploy LSTM as our web service.

Model	Description	Accuracy
Countvectorizer with Multinomial Naive Bayes	Number of features : 36751 MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)	0.8140
Tfidf Vectorizer with Logistic Regression	Number of features : 10505 LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, l1_ratio=None, max_iter=100, multi_class='auto', n_jobs=None, penalty='l2', random_state=None, solver='lbfgs', tol=0.0001, verbose=0, warm_start=False)	0.8500
Word2Vec with Random Forest Classifier	Number of words in the vocabulary list : 6945 Training set : 4500 feature vectors with 300 dimensions Validation set : 500 feature vectors with 300 dimensions	0.7640
LSTM	embedding layer => LSTM layer => dense layer Compile and fit the model using log loss function and ADAM optimizer.	0.8750

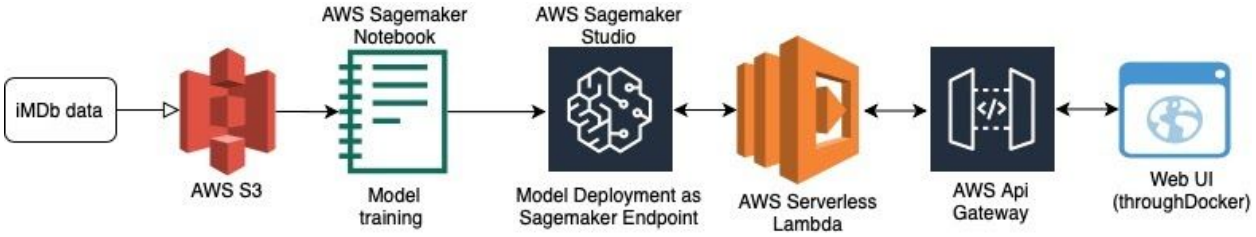
AWS Sagemaker

AWS was launched in 2006 and we have different technical stack in various AI/ML areas. Right now, we'll be focusing on ML. SageMaker is a machine learning environment that is made to simplify the work of engineers by providing tools for model building and deployment. It also provides Jupyter notebook, to simplify data exploration and analysis without server management hassle. It also has built-in algorithms that are optimized for large datasets and computations.

The pipeline for generic model deployment on Sagemaker studio is to label, aggregate, prepare the training data. We can use built-in solutions or bring our own algorithms and tune the models using distributed infrastructure with capabilities to capture, debug, compare performance. Finally deployment using endpoints with capabilities of monitoring and managing models.



Similarly, In our project, we have downloaded the open source iMDb dataset which was put on s3. A jupyter notebook was hosted from sagemaker studio which retrieves data from s3 to train the sentiment analysis models and deploy that model on the studio as sagemaker endpoint. A serverless lambda function was created for seamless interaction between the AWS Api gateway and endpoint. Finally, the API gateway can be accessed through the front-end HTML page to get classification on AWS on reviews.



Here are the performance parameters that we obtained from different algorithms. We finally chose LSTM for model deployment in our project. Even though it took time to train the validation datasets, it gave the highest accuracy among all and we focused more on the testing part since once a model is deployed, we don't need to worry about the training time.

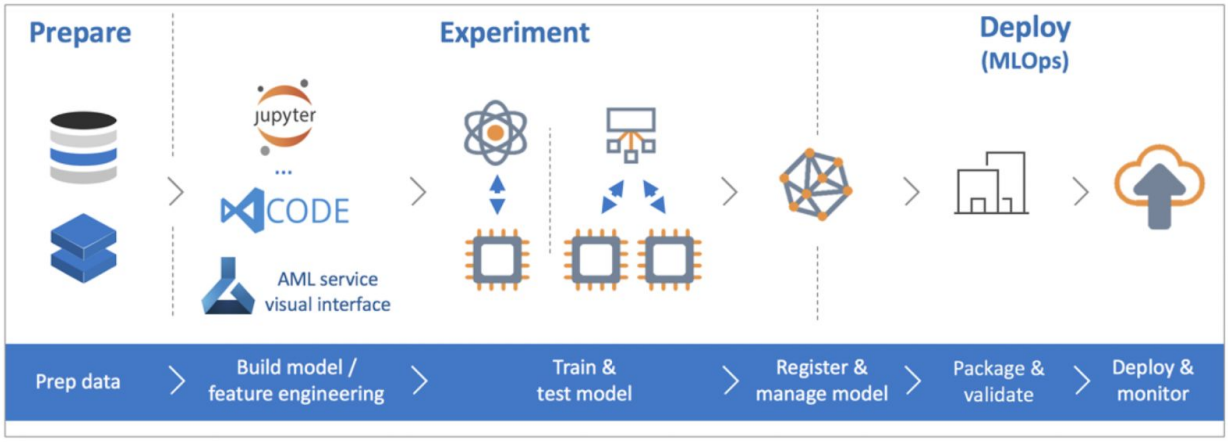
Model	**Wall time**	**CPU Time**	**Time per Epoch**	**Memory**	**Accuracy**	**AUC**
Countvectorizer with Multinomial Naive Bayes	1.15 s	user 1.12 s sys: 10.2 ms total: 1.13 s	-	Peak memory: 1935.25 MiB Increment: 0.23 MiB	0.8140	0.8140
Tfidf Vectorizer with Logistic Regression	1.03 s	user 1.12 s sys: 0 ns total: 1.12 s	-	Peak memory: 1944.39 MiB Increment: 0.00 MiB	0.8500	0.8500
Word2Vec with Random Forest Classifier	55.6 s	user 56.7 s sys: 0 ns total: 56.7 s	-	Peak memory: 2040.82 MiB Increment: 0.00 MiB	0.7640	0.7640
LSTM	2min 19s	user 3min 57s sys: 15.2 s total: 4min 13s	22s	Peak memory: 2772.10 MiB Increment: 1647.09 MiB	0.8750	0.8750

Azure Machine Learning

Azure Machine Learning platform, is aimed at setting a powerful playground for both newcomers and experienced data scientists. For newcomers, ML Studio is a graphical drag-and-drop interface, very easy to use. For experienced developers, Azure ML services also provides python SDK, which doesn't have built-in methods, requires custom model engineering but with a Web interface to deploy models, visualize data, and work on dataset preparation.

Our project is done inside the ML services platform, it offers a powerful toolset to manage ML experiments, use popular frameworks like TensorFlow, scikit-learn, and deploy models into production in a third-party service like Docker. Each machine learning job can be initialized as an experiment, which supports main machine learning frameworks, holds historic configuration and has a workbench dashboard for model management. The model CI / CD process can be realized on Azure, on-premises or even edge devices. Azure also combined with Visual Studio Tools for AI which can be easily integrated into .NET web service.

To train and deploy a custom model as a web service on Azure, we need the following process:



1. Create the workspace on Azure and set environments.
2. Prepare data: Upload the datasets to Azure Blob and register the datasets
3. Create Azure ML Experiment to build model, train model and test model, where we can log metrics to monitor and evaluate the model like running time, model accuracy.

Below are the metrics from the training process:

Model	**Wall time**	**CPU Time**	**Memory**	**Accuracy**	**AUC**
Countvectorizer with Multinomial Naive Bayes	1.02 s	user 1.03 s sys: 0 ms total: 1.03 s	Peak memory: 3365.85 MiB Increment: 0 MiB	0.8140	0.8142
Tfidf Vectorizer with Logistic Regression	3.11 s	user 7.09 s sys: 2.51 s total: 9.6 s	Peak memory: 3365.37 MiB Increment: 0.02 MiB	0.8500	0.8500
Word2Vec with Random Forest Classifier	52s	user 52.3 s sys: 0 ns total: 52.3 s	Peak memory: 2040.82 MiB Increment: 0 MiB	0.7620	0.7620
LSTM	2min 19s	user 3min 57s sys: 15.2 s total: 4min 13s	Peak memory: 2772.10 MiB Increment: 1647.09 MiB	0.8890	0.8890

After we got the desired model LSTM, the next step is to register the model either through the experiment running object or the saved model file on Azure workspace.

Finally, we also deploy the model on Azure as the webservice(RESTful API).

1. Define an entry script
   The entry script receives data submitted to a deployed web service and passes it to the model. It takes the response returned by the model and returns that to the client. The script is specific to the model, which must understand the data that the model expects and returns. The two functions needed to accomplish in the entry script are:
   - Loading model (using a function called init())
   - Running model on input data (using a function called run())
2. Define an inference configuration: define the environment and packages the model needs
3. Choose a computing target. We can first deploy on my local website to test and debug. The deployment of the model in Azure Container Instances is limited to CPU-based workloads that require less than 48 GB of RAM. For serious production, developers can also choose to deploy the model on Azure Kubernetes Service (AKS) for Real-time inference and Azure Machine Learning compute clusters for Batch inference
4. Finally, we are able to deploy our model as a web service by defining the deployment configuration. Since our project works with Azure Container Instances, we can only deploy the model with 1 CPU and 1 memory_gb.

To consume the web service. Azure provide several ways to consume the RESTful API, including calling the service from C#, Java, Go and Python. Unfortunately, Azure Machine Learning SDK does not provide API usage from Javascript compared with the serverless computing method provided by AWS. We still need a backend to run the service.

Azure Machine Learning Featured Products:
Besides the powerful SDK, Azure ML also offers low-code or no-code experience for machine learning beginners.

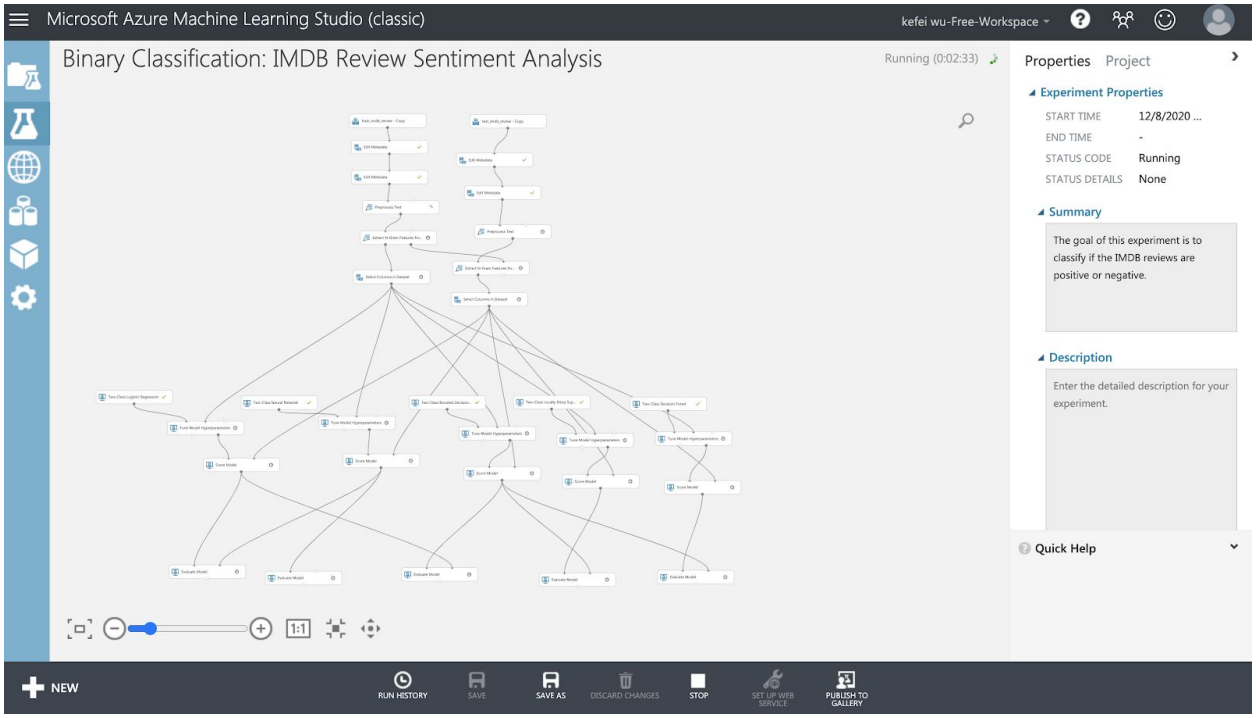
AutoML (low-code experience): Traditional machine learning model development is resource-intensive, requiring significant domain knowledge and time to produce and compare dozens of models. With automated machine learning (AutoML), we'll accelerate the time it takes to get production-ready ML models with great ease and efficiency. It is the process of automating the time consuming, iterative tasks of machine learning model development. It allows data scientists, analysts, and developers to build ML models with high scale, efficiency, and productivity all while sustaining model quality.

We fed in the processed data by word2vec and the autoML trained 44 models for me and 37 of them achieved 99% accuracy. AutoML works best for those who don't have to have significant domain knowledge and time to produce and compare dozens of models. They can accelerate the time with great ease and efficiency.

Algorithm name	Explained	Accuracy ↓	Sampling ⓘ	Run	Created	Duration	Status
StandardScalerWrapper, LightGBM		1.00000	100.00 %	Run 44	2020年12月8日 16:47	1m 11s	Completed
MinMaxScaler, LightGBM		1.00000	100.00 %	Run 43	2020年12月8日 16:46	1m 3s	Completed
SparseNormalizer, LightGBM		1.00000	100.00 %	Run 42	2020年12月8日 16:45	1m 10s	Completed
SparseNormalizer, LightGBM		1.00000	100.00 %	Run 41	2020年12月8日 16:43	1m 14s	Completed
StandardScalerWrapper, LightGBM		1.00000	100.00 %	Run 40	2020年12月8日 16:41	1m 34s	Completed
MinMaxScaler, ExtremeRandomTrees		1.00000	100.00 %	Run 39	2020年12月8日 16:40	1m 5s	Completed
MinMaxScaler, LightGBM		1.00000	100.00 %	Run 38	2020年12月8日 16:39	1m 1s	Completed
StandardScalerWrapper, LightGBM		1.00000	100.00 %	Run 37	2020年12月8日 16:38	1m 4s	Completed
StandardScalerWrapper, RandomForest		1.00000	100.00 %	Run 36	2020年12月8日 16:36	1m 4s	Completed
SparseNormalizer, LightGBM		1.00000	100.00 %	Run 34	2020年12月8日 16:34	1m 2s	Completed
StandardScalerWrapper, LightGBM		1.00000	100.00 %	Run 32	2020年12月8日 16:31	1m 5s	Completed

Algorithm name	Explained	Accuracy ↑	Sampling ⓘ	Run	Created	Duration	Status
TruncatedSVDWrapper, XGBoostClassifier		0.49000	100.00 %	Run 15	2020年12月8日 16:06	2m 38s	Completed
TruncatedSVDWrapper, XGBoostClassifier		0.51000	100.00 %	Run 33	2020年12月8日 16:32	1m 16s	Completed
MinMaxScaler, LightGBM		0.53200	100.00 %	Run 30	2020年12月8日 16:28	1m 4s	Completed
PCA, LightGBM		0.54400	100.00 %	Run 35	2020年12月8日 16:35	1m 2s	Completed
MinMaxScaler, SVM		0.57200	100.00 %	Run 10	2020年12月8日 15:58	1m 56s	Completed
MinMaxScaler, RandomForest		0.89800	100.00 %	Run 8	2020年12月8日 15:55	1m 20s	Completed
MinMaxScaler, RandomForest		0.95800	100.00 %	Run 7	2020年12月8日 15:54	1m 26s	Completed
SparseNormalizer, XGBoostClassifier		0.99600	100.00 %	Run 18	2020年12月8日 16:11	1m 8s	Completed
MinMaxScaler, RandomForest		0.99600	100.00 %	Run 9	2020年12月8日 15:57	1m 4s	Completed
StandardScalerWrapper, LightGBM		1.00000	100.00 %	Run 44	2020年12月8日 16:47	1m 11s	Completed
MinMaxScaler, LightGBM		1.00000	100.00 %	Run 43	2020年12月8日 16:46	1m 3s	Completed

Machine Learning Studio (no-code experience): ML Studio is a drag-and-drop machine learning builder. This includes data exploration, preprocessing, choosing methods, and validating modeling results. The Studio supports around 100 methods that address classification (binary+multiclass), anomaly detection, regression, recommendation, and text analysis. It is a standalone service that only offers a visual experience, which does not interoperate with Azure Machine Learning. There is no SDK for code and limited computing power, only built for a zero code experience. We preprocessed the data with bag-of-words and tried three ML models Logistic Regression, Neural Network, Random Forest, which are all built-in models by azure.



Model (BoW)	Epochs	Hyperparameters for Trained best model
Logistic Regression	3	OptimizationTolerance = 0.000028, L1weight = 0.496711, L2weight = 0.96395, Memorysize = 6
Neural Network	5	Learning rate = 0.3544, LossFunction = SquaredError, Number of iterations = 21
Random Forest	50	Min samples per leaf node = 3, Random splits per node = 98, Max depth = 12, decision trees = 25

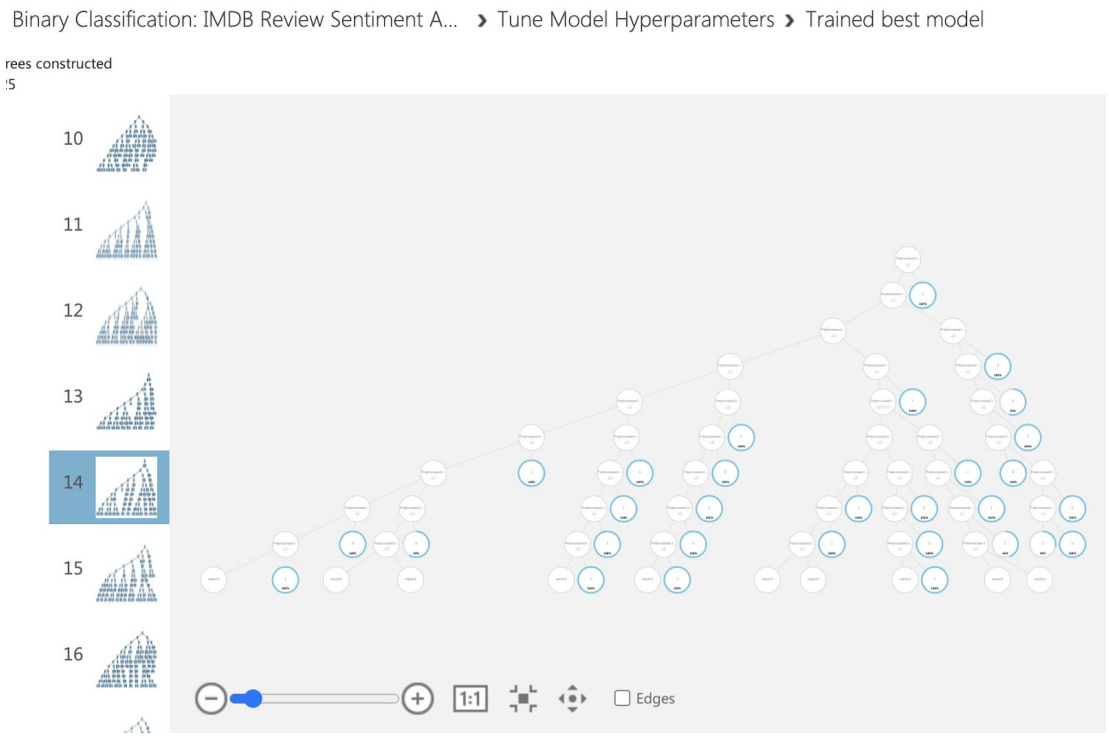
Model	Training Time	Accuracy	Precision	Recall	F-Score	AUC	Avg Log Loss	Training Log Loss
LR(in SDK)	3.11 s	0.8500	0.8500	0.8500	0.8500	0.8500	0.4357	12.5643
LR	30.316s	0.76996	0.761285	0.78656	0.773716	0.84487	0.494765	28.6205
NN	11h32m	0.74184	0.719759	0.79208	0.75419	0.81671	0.907074	-30.86317
RF	7m31s	0.72004	0.70937	0.74552	0.726996	0.79469	0.649241	6.334262

However, the training time is especially long, for the same Logistic Regression model, it took 10 times compared with training in SDK. For 1 Hidden Layer Neural Network, it takes 10 hours to train which is quite unacceptable, partly because we only created a free trial version of workspace for the service, actually it costs $9.99 one months subscription.

We can view all Hyperparameters for each training Epoch, save the best Hyperparameters for future usage. For eg., for the Random Forest model, we can visualize each constructed tree in training epochs which is great for newcomers and high-level understanding of the model.

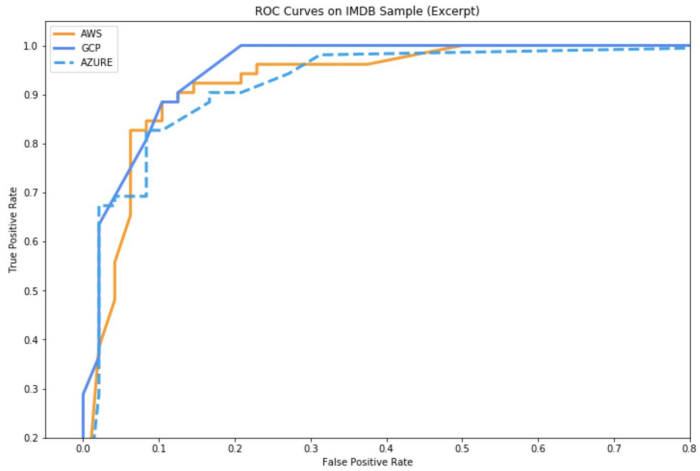
rows 50 columns 11

Minimum number of samples per leaf node	Number of random splits per node	Maximum depth of the decision trees	Number of decision trees	Accuracy	Precision	Recall	F-Score	AUC	Average Log Loss	Training Log Loss
view as										
3	98	12	25	0.72004	0.70937	0.74552	0.726996	0.794694	0.649241	6.334262
5	82	10	16	0.71008	0.705477	0.72128	0.713291	0.780936	0.657016	5.212632
2	337	30	6	0.69728	0.691222	0.71312	0.702	0.762184	0.588141	15.149211
3	254	2	23	0.67908	0.67964	0.67752	0.678579	0.747978	0.681915	1.620426
3	537	28	4	0.68212	0.67687	0.69696	0.686768	0.74251	0.632397	8.764372
1	69	10	9	0.66084	0.663442	0.65288	0.658119	0.725942	0.66533	4.013136
4	393	51	2	0.65656	0.657975	0.65208	0.655014	0.708513	0.885219	-27.710107
7	484	15	2	0.64932	0.646957	0.65736	0.652117	0.707527	0.632893	8.692794
4	34	24	5	0.63664	0.644893	0.60816	0.625988	0.687943	0.664914	4.073243
2	219	5	9	0.66588	0.642833	0.74656	0.690824	0.730661	0.656002	5.358932
3	83	3	3	0.56244	0.641318	0.28336	0.393053	0.593844	0.682454	1.542711
5	59	43	3	0.63348	0.640482	0.60856	0.624113	0.686693	0.649995	6.225586

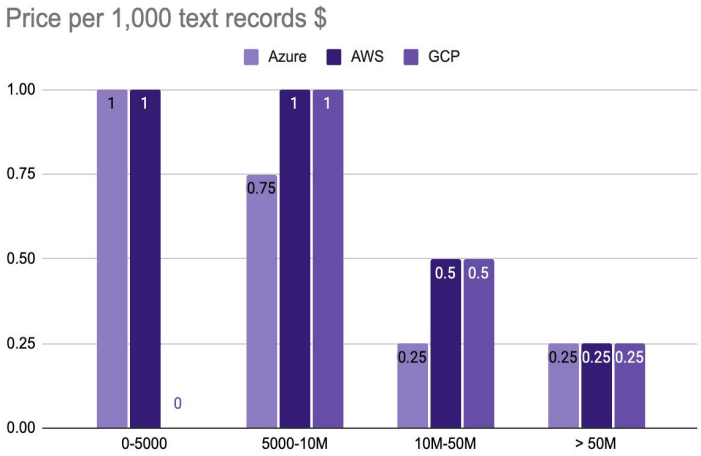


Evaluation - API Comparison:

For a lightweight project, the easiest way is to just call the API provided by various cloud platforms. So we conducted experiments to compare AWS Sagemaker vs Azure cognitive services vs GCP Cloud Natural Language through a third-party platform called RAPYD.AI. Since in a specific case, the datasets and label could be different from what the overall 'sentiment', we set accuracy as the primary metric to evaluate. Price for API is also a major consideration as pricing of API is varied by usage, a service with frequency request or occasional usage could be far from each other. Below are the results on the imdb datasets.



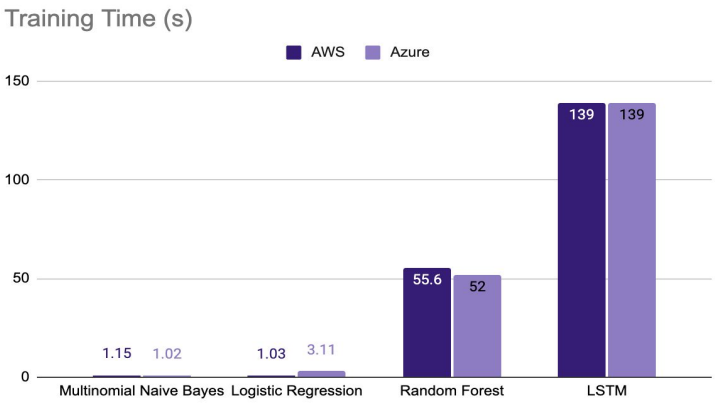

```
AWS: ROC AUC=0.9379
GCP: ROC AUC=0.9589
AZURE: ROC AUC=0.9329
```

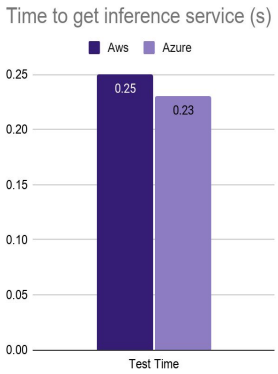


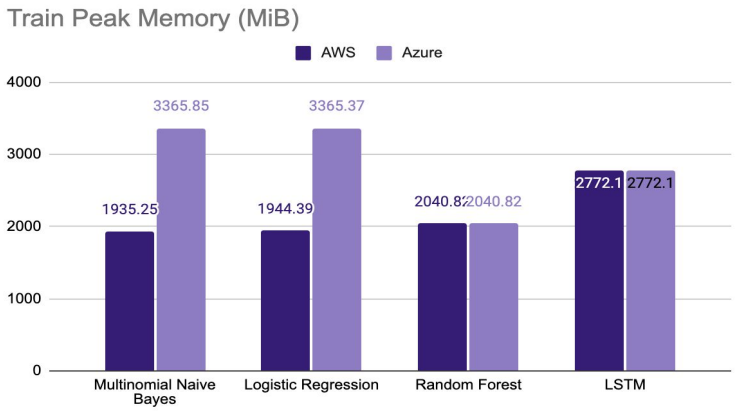
GCP wins accuracy followed by AWS. For pricing of APIs, GCP offers 5000 text requests each month, but it limits the API usage up to 20 million each month. However, AWS provides API usage up to process requests over 10 billions each month. The more requests a project needs, the more favorable choice is to go with AWS. Azure provides their cognitive service API one in all, which means if developers also require CV, face API or Custom Search engine to launch their services, Azure cognitive service API could be a one-stop solution.

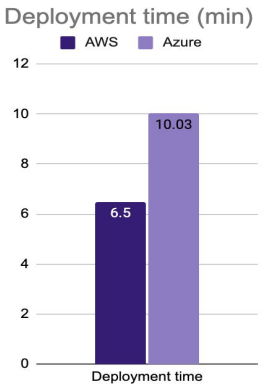
Final Comparison Across Cloud Platforms - Quantitative Comparison

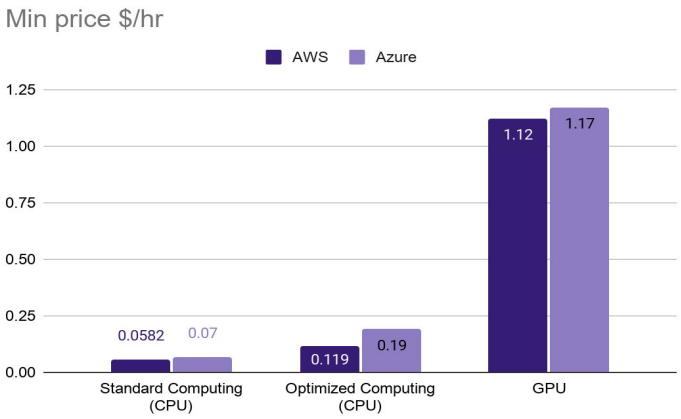
Below is the comparison of training/deployment time, memory usage, pricing model of AWS and Azure. AWS has better performance than Azure in training/deploying time,memory usage.

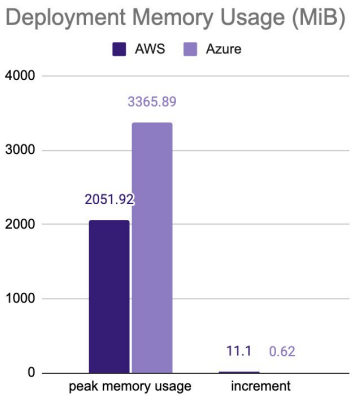












	AWS		**Azure**	
	Pros	**Cons**	**Pros**	**Cons**
ML Platforms	Comprehensive Documentation Fully Scalable, on-demand, Managed Infrastructure Distributes inference compute by deploying Endpoints Notebooks (examples of popular algorithms) Rich marketplace	Expensive! Knowledge of coding and Data Engineering processes is required May/may not be suitable for beginners No scheduled training job Expensive!	Drag-and-Drop UI Suitable for beginners Automated and Semi-automated ML services (anomaly detection, recommender system, ranking) More flexible with Algorithms Easily combined with C#, .Net service	RESTful API does not support AJAX request No built-in algorithm support in PaaS Relative longer deployment time Scenario-based documentation, unclear text tips for error
On-demand Pricing	Billed for the duration for use (per hour) $0.28/hr (ml.m4.xlarge 4 CPU, 16GB RAM)		Billed for the duration for use (per hour) $ 0.08/hr (standard computing, 1 CPU 3.5 GB RAM)	

Both platforms charge more for their custom services, for example AWS built-in algorithms can only be achieved through expensive instances, Azure autoML service also has to be run on an optimized CPU. Also, Azure provides low-priority computing instances with $ 0.02/hr (standard computing, 1 CPU 3.5 GB RAM), which does not guarantee that once the job is submitted, it can take over the computing instance. In all, Azure is more favored for beginners and small projects, AWS is more useful for large-scale systems.

Discussion - AWS Challenges:

- Since the service of JupyterLab, sagemaker endpoints, lambda functions are chargeable so every time they need to be shut down in order to avoid getting charged.
- The connection between web api, Lambda and deployed model has to be done through proper variables with data in same format (this took major time to resolve).

Discussion - Azure Challenges:

- When writing the entry script, the path to our model is a virtual path created during the deployment process and we can not debug the script until we deploy the web service.
- The deployment process could take >= 15mins. It's difficult to debug the code or environment setting and we have to look for any deployment raise error. When making changes to the model or script file, the procedure to update the service is again by creating the docker inside the Azure which takes exactly the same time.

Discussion - Demo: (Code: https://github.com/sonsharma/FinalCloudMLProject.git)
Since we already have shut down the services in Azure and AWS (to avoid getting charged), we don't have a working demo web API but we have shown the working demo of our application to professors during class presentation (can be found in class recordings).

Discussion - Use case summary:
Both AWS and Azure provide similar products for model training and management, although in our project, AWS wins a little over Azure in overall performance. Other considerations-

- For a .NET webservice, Azure Machine Learning has tailored tool kits for it
- For drag-and-drop, a zero code experience, Azure Machine Learning studio is best
- For serverless computing and AJAX usage of deployed models, we can only use AWS
- For built-in algorithms - AWS Sagemeker is the choice (Azure ML don't have this)

Conclusion and Future Work
From a practical view, there are so many things we couldn't have the chance to explore.

- Host the Web-API on another cloud through docker containers (to realize high availability and fault tolerance)
- How to scale service, realtime inference, batch inference & improve Model Accuracy
- What if the data is not labeled? Auto label? (A great effort is invested in collecting and processing data from websites/social media and labeling the data as positive or negative. The auto-label process provided by cloud can as well be a deciding factor)
- MLOps: Model management (This matters more than code from scratch. As a result, how the CI/CD process offered by the platform should also be tested).