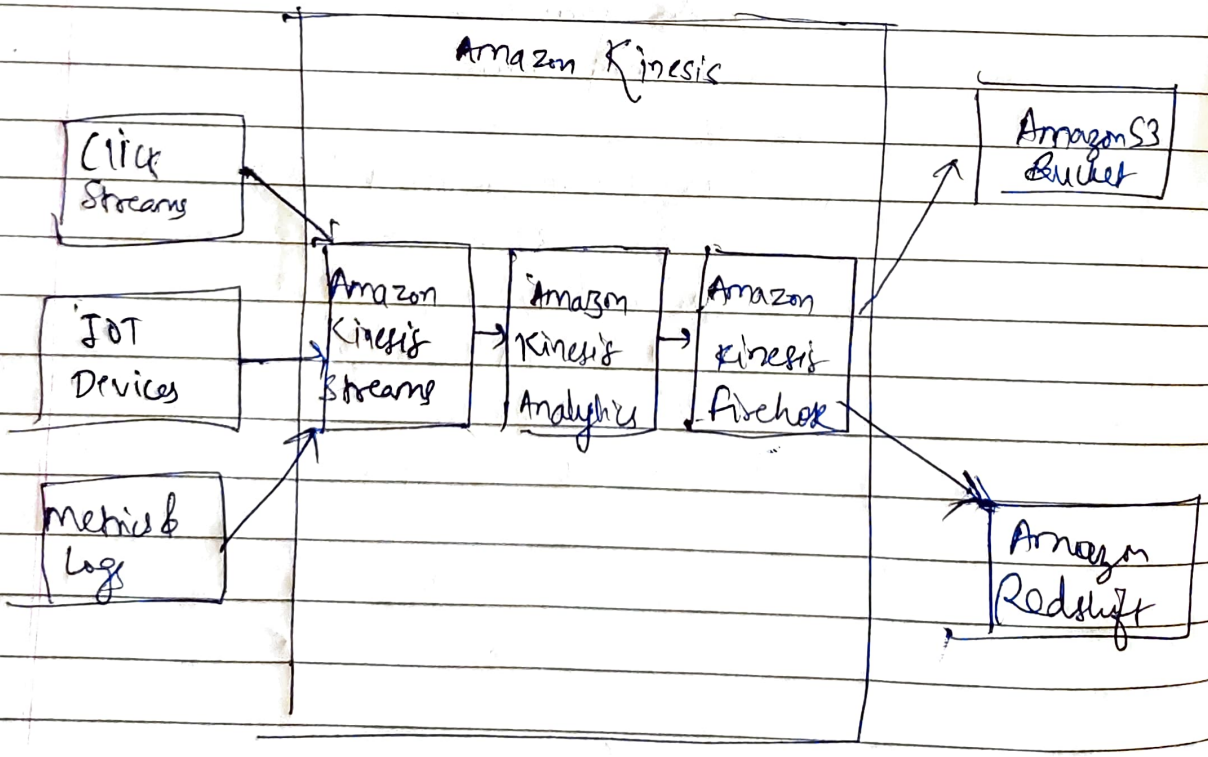
AWS

ML Specialty

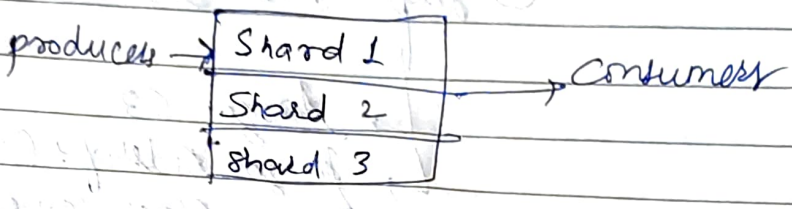
(1) Data engineering techn

- S3
  - VPC configuration
  - Object Encryption
- Kinesis → Real-time big data
  - Kinesis Streams
    - Low latency streaming
  - Kinesis Analytics:
    - Real-time analytics using SQL
  - Kinesis Firehose
    - load streams to S3, Redshift, etc.
  - Kinesis VideoStreams
    - meant for streaming video in real time



1. Kinesis Streams:

↳ Divided in ordered shards / partitions.



↳ Retention duration = 24 hrs (default)
≤ 365 days (max)

↳ Ability to reprocess / replay data

↳ Multiple applications can consume same stream

↳ Once data is inserted in Kinesis, it can not be deleted.

↳ Records can be upto 1MB in size.

Limits:

↳ Producer:

↳ 1MB/s or 1000 msg/sec at write per shard.

↳ "ProvisionedThroughputException" error if above limit is crossed.

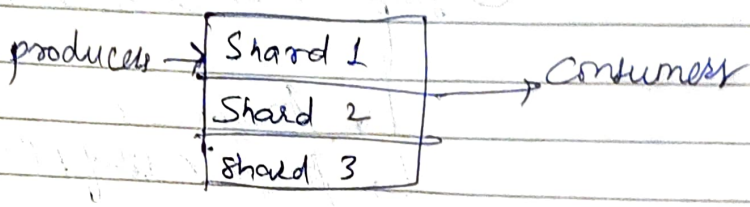
↳ Consumer:

↳ 2MB/s of read per shard across all consumers

↳ 5 API calls per second per shard across all consumers.

1 Kinesis Streams:

↳ Divided in ordered shards/ partition.



↳ Retention duration = 24 hours (default)
≤ 365 days (max)

↳ Ability to reprocess / replay data
↳ Multiple applications can consume same stream
↳ Once data is inserted in Kinesis, it can not be deleted.
↳ Records can be upto 1MB in size.

Limits:

↳ Producer:

↳ 1MB/s or 1000 msg/sec at write per shard.

↳ "ProvisionedThroughputException" error if above limit is crossed

↳ Consumer:

↳ 2MB/s at read per shard across all consumers

↳ 5 API calls per second per shard across all consumers.

(II) Kinesis Data Firehose:

↳ Imp AWS Destinations :—

↳ Amazon S3

↳ Amazon Redshift (Copy through S3)

↳ Amazon Elastic search

↳ Imp 3rd party Destinations:

↳ Data dog

↳ Splunk

↳ MongoDB.

↳ New Relic.

(III) Kinesis Video Stream

- Producer :-
  - Security cam, body-worn cam, etc
  - One producer per video stream.
- Video playback capability :-
- Consumers
  - Build your own (MXNet, TensorFlow, etc)
  - AWS Sagemaker.
  - Amazon Rekognition Video
- Keep data for 1hr to 10 years

★ Glue Data Catalog :-

- Metadata Repository for all your tables :-
  - Automated Schema Inference
  - Schemas are versioned.
- Integrates with Athena or Redshift Spectrum

↳ # Important:

Glue crawler automatically detects the schema of tables (jsons / csvs in an S3 bucket (specified to it)

It is also able to detect the partitioning scheme of a certain table file.

for eg: If a file is partioned using date like

s3://bucket_name/file_name/yyyy/mm/dd/hh/file.csv

★ Glue ETL :-

Transform data, clean & enrich data (before doing analysis)

- Fully managed, cost effect, pay as you use
- Jobs are run on a serverless Spark platform

★ Glue Scheduler :- To schedule the jobs

★ Glue Trigger :- To automate job runs based on "events"

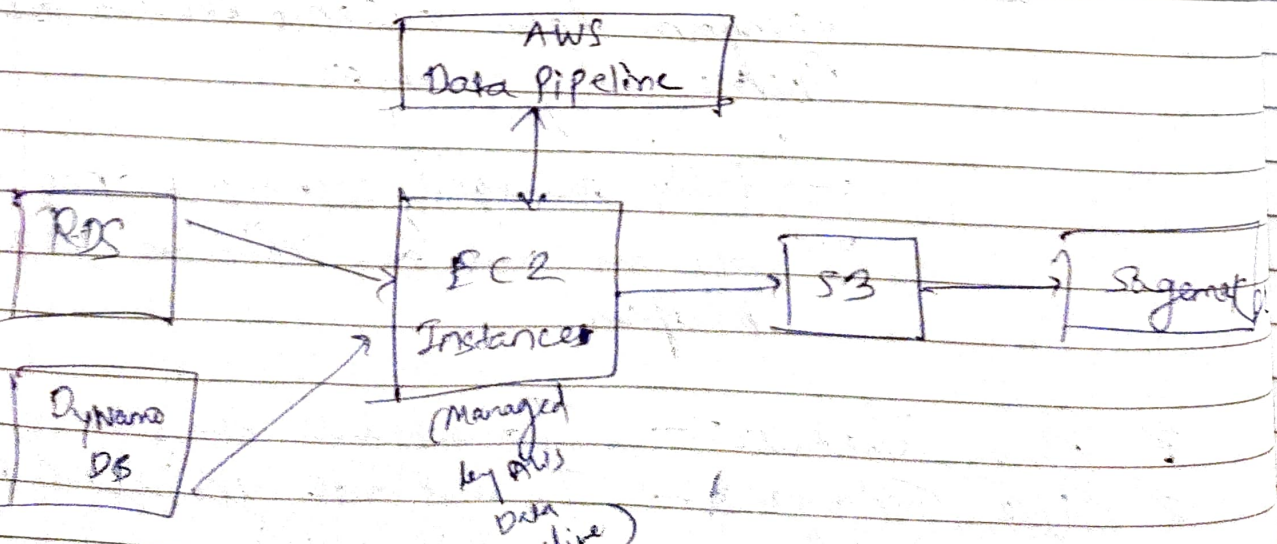
Glue ETL - Transformations :-

→ Bundled transformations :-
- DropFields, DropNullFields
- Filter
- Join - to enrich data
- Map - add, delete fields, perform external lookups.

→ ML transformations :-
- FindMatches ML :-
  identify duplicate or matching records in your dataset. (even when records do not have a common unique identifier & no fields match exactly).

→ Format conversions :-
  → CSV, JSON, Avro, Parquet, ORC, XML

AWS Data Pipeline :-



AWS Batch :

- → Run batch jobs as Docker Instances
- → Dynamic provisioning.

Quick Sight

- Visualization toolkit for data coming from all kinds of sources —
  - In premise
  - AWS S3, Redshift,
  - EC2 instances etc.

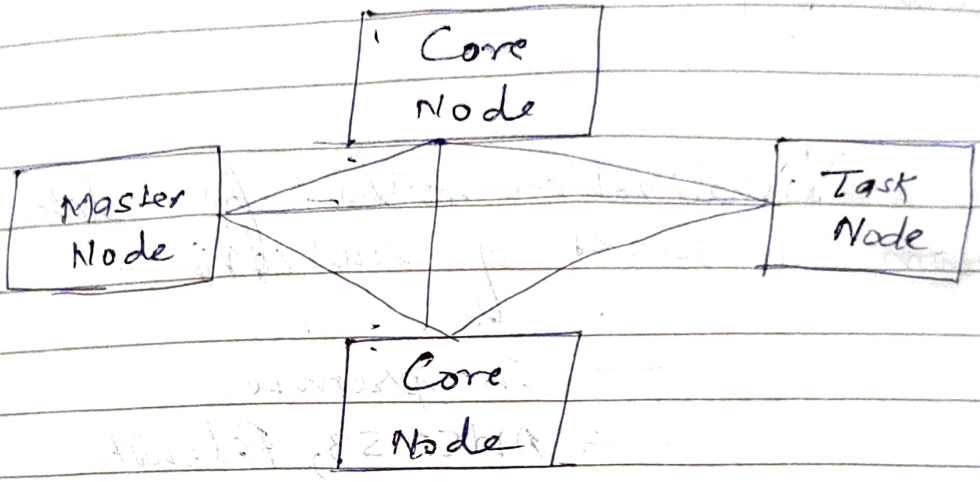
Different types of viz available :—

- Bar charts
  - Histograms
- Pie charts
- Scatter plots
- Heat maps
- Line plots
- Scatter plots

EMR

Elastic Map Reduce

EMR Cluster :—



EMR Usage :—

① → Transient vs. Long Running cluster

↓ Terminates automatically when all defined steps are completed

↓ Terminated by Load ~~automatically~~ ~~not~~ (Not automatically stopped)

EMR Storage :—

Temporary storage ← • HDFS → By default size of a block in HDFS - 128 MB

• EMRFS — Access S3 as if it were HDFS

Feature Engineering :-

→ Applying your knowledge of the data to the model you're using to create better features to train your model with.

→ Which features should I use?
→ Do I need to transform these features?
→ How do I handle missing data?
→ Should I create new features from existing ones?

Handling (Imputing) missing data.

- Replacing by mean
- Replacing by mode
- Replacing by median
- Using MICE → Multiple Imputation by Chained equations
- Replacing values using ML
  - By K-means for categorical missing cols
  - By Random-cut forest for missing column (numerical)

Dealing with unbalanced data

- # Replicate minory class label entries
- # Delete some majority class label entries
  - (This might lead to loss of data)

Binning, Transforming, Encoding, Scaling & Shuffling.

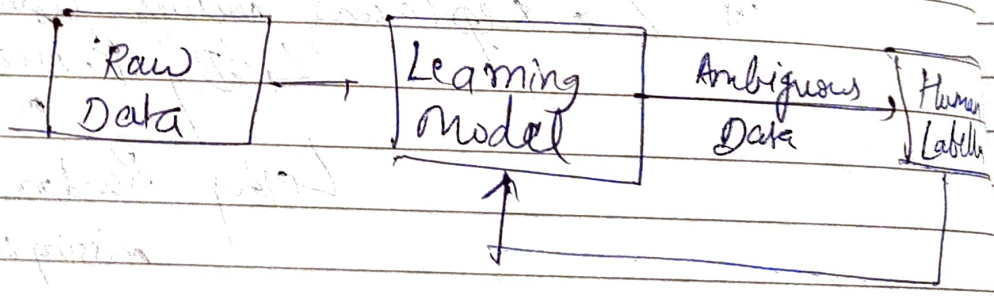
WAYS TO GENERATE TRAINING LABELS

Sagemaker Ground Truth

↳ AWS service to use human to label the missing data.

Ground Truth creates its own model labelled by people.

If this model learns only images it is not sure about are sent to human labellers.



Other ways to generate training labels:

↳ AWS Reckognition

↳ AWS Comprehend

↳ Any pre-trained model or unsupervised technique that may be helpful.

TF-IDF

↳ Term Frequency – Inverse Document Frequency

↳ Imp. for data search – figures out what terms are most relevant for a doc.

↳ TF measures how often a word occurs in a document.

↳ DF measures how often a word occurs in a whole set of documents.

Assumptions of TF-IDF :-

↳ A document is just a bag of words

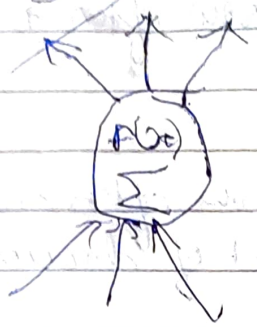
Unigrams — Individual words
Bi-grams — Set of two words taken at a time
n-grams — Set of n words taken at a time

Preparing data for TF-IDF

DEEP LEARNING

Activation function

→ Defines the output of a node/ neuron given its input signals



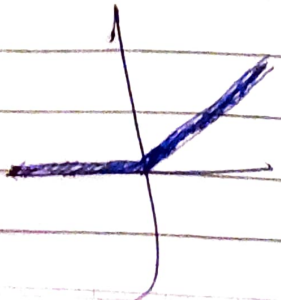
(I) Linear ~~Activation function~~
- Doesn't really do anything
- Doesn't allow backpropagation

(II) Binary Step function
- It's on or off.
- Can't handle multiple classification.
- Vertical slopes don't work well with calculus

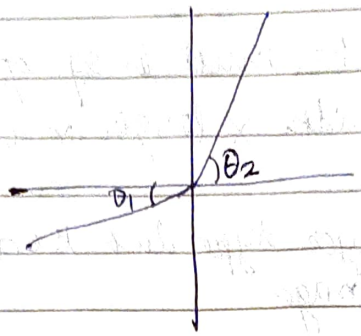
(III) Sigmoid / Logistic / TanH

(III) Rectifier Linear Unit (ReLU)

Easily calculable



Leaky ReLU



$$\theta_1 > \theta_2$$

Parametric ReLU (PReLU)

(↳ the θ_1 is decided by RNN)

↳ Complicated & computationally expensive

Softmax Activation function

↳ Used on the final o/p layer of a multiple classification problems.

↳ Basically converts o/p to probabilities of each classification.

CHOOSING AN ACTIVATION FUNCTIONS:—

1. For multiple classifications, use softmax on the o/p layer.
2. RNN's do well with Tanh
3. For every thing else.
   - ① → start with ReLU
   - ② If you need better try Leaky ReLU
   - ③ Last resort = PReLU, Maxout

④ SWISH for really deep networks

CNNs with Keras / Tensorflow

1. Source data must be of appropriate dimension ie width x length x color channels

2. Conv2D layer type does the actual convolution on a 2D image.

   Conv1D & Conv2D layers can be used - doesn't have to be image data

3. Max Pooling 2D layers can be used to reduce a 2D layer down by taking the maximum value in a given block.

4. Flatten layers will convert 2D layers to a 1D layer for passing into a first hidden layer of neurons.

5. Typical usage:-

   Conv2D → MaxPooling2D → Dropout → Flatten → Dense → Dropout → Softmax

Problems of CNNs

CNNs are very resource-intensive (CPU, GPU, RAMs)
Lots of hyperparameters :-
   → Kernel sizes.
   → Many layer with diff no. of units
   → Amount of pooling.
   → No. of layers
   → Choice of optimizer.
Getting the training data is the hardest part
Storing & Accessing it.

Specialized CNN Architectures :-

— Defines specific arrangements for layers, padding & other hyperparameters

(I) LeNet-5
- Good for handwriting recognition

(II) AlexNet
- Image classification, deeper than LeNet

(III) GoogleLeNet
- Even deeper, but with better performance
- Introduces 'inception modules'
- ~~ResNet~~ (

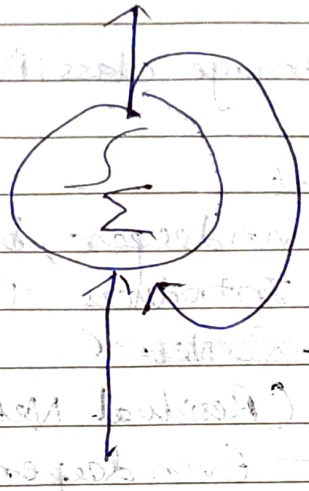
(IV) ResNet (Residual Network)
- Even deeper
- Maintains performance via skip connections

Recurrent Neural Networks (

What are they for

— Time Series Data.

— Data that consists of sequences of arbitrary length.



RNN Topologies

→ Sequences to sequence

— predict stock prices based on series of historical data

→ Sequence to vectors

— words in a sentence to sentiment

→ Vector to sequence

— Create caption from image

→ Encoder → Decoder

— Sequence → Vector → sequence

— Ie machine translation

11. GPU for Instance types on AWS for Deep learning:

- P2: 8 Tesla V100 GPUs
- P2: 16 K80 GPUs
- P3: 8 GPUs (All Nvidia chips)

12. Tuning Neural Networks:

- Learning Rate
  - NNs are trained by gradient descent.
- Batch Size

IMPORTANT

Large batch sizes tend to get stuck inside a "local minima" or random instead of a correct solution

Counter-intuitive points:

① Smaller batch sizes can work their way out of the "local minima" more easily

② Batch sizes that are too large can end up getting stuck in the wrong solution

③ Random shuffling at each epoch can make NN learn very inconsistent results from run to run.

Important Recap points:

(good) ① Small batch sizes tend not to get stuck in local minima

(bad) ② Large batch sizes can converge on the wrong solution at random

(bad) ③ Large learning rates can overshoot the correct solution.

(bad) ④ Small learning rates increase training time.

Drop out layers force the network to spread out its learning throughout the network & can prevent overfitting from learning concentrating on one set of.
↳ Early stopping also does the same.

Regularization :-

— Prevents overfitting
- A regularization term is added & weights are learned.

L1 & L2 Regularization :

↳ L1 term is the sum of weights

$$= \lambda \sum_{i=1}^{k} |w_i|$$

↳ L2 term is the sum of squares of weights

$$= \lambda \sum_{i=1}^{k} w_i^2$$

↳ Same idea is applied to loss function

IMPORTANT

Difference b/w L1 & L2 Regularization

L1 = Sum of weights

① Performs feature selection - entire feature is 0
② Computationally inefficient
③ Sparse output

L2 — Squared sum of weights

① All features are considered, just weighted
② Computationally efficient
③ Dense output

In sklearn, Lasso Regressor ⇒ L1 regularized model
In sklearn, Ridge Regressor ⇒ L2 regularized model

~~RNN~~ a LSTM is a specific type of RNN that solves the problem of items losing their weight over time. In NLP, words in a sentence may be significant, regardless of their position.

Why would I want L1?

→ Feature selection can reduce dimensionality.
→ But if you think all features as important, L2 is probably a better choice.

The Vanishing Gradient Problem :—

— When the slope of the learning curve approaches zero, things can get stuck.

— We end up working with very small numbers that slow down the training, or even introduce numerical errors.

— This becomes a problem with deeper networks & RNNs as these "vanishing gradients" propagate to to deeper layers.

— Opposite of this problem is called "Exploding gradients".

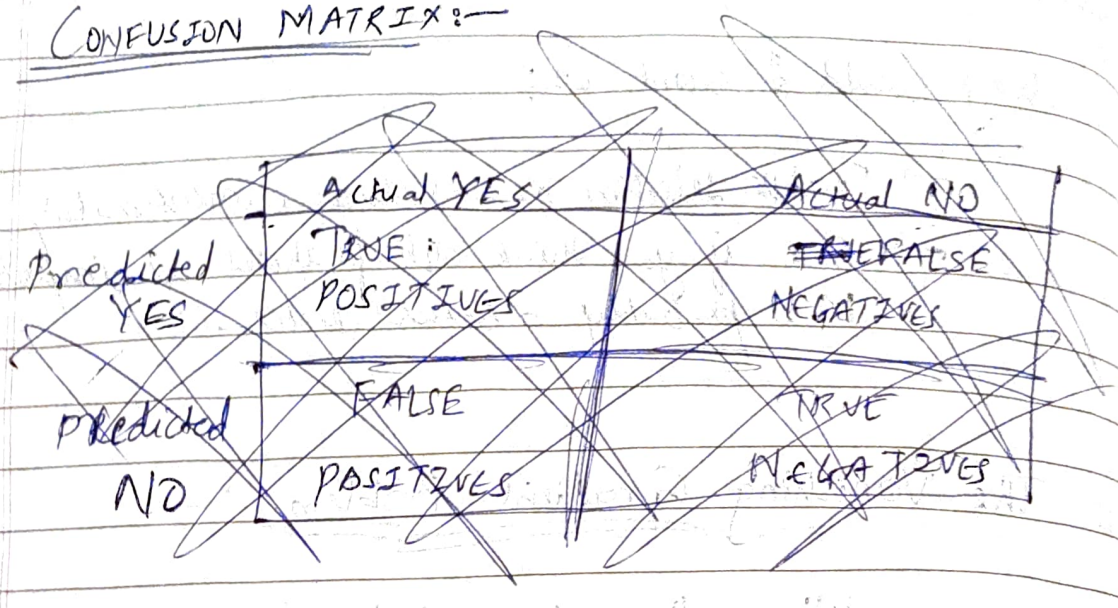
Fixing this problem:

— Instead of traing all layers together, we can train multi-level layer training.

— Break up levels into their own sub-levels trained individually.

— LSTM, RESNET, Better Activation func^{n}

CONFUSION MATRIX:—



	Actual YES	Actual NO
predicted YES	TRUE POSITIVES	FALSE POSITIVES
predicted NO	FALSE NEGATIVES	TRUE NEGATIVES

- Precision should be focussed when you care about False positives more.
- Recall should be focussed when you care about False negatives more.

Measuring your models :-

(I) Recall =>

$$\frac{TP}{TP + FN}$$

also called as
- True Positive Rate
- Sensitivity
- Completeness

$$\frac{FP}{FP + TN}$$

False Positive Rate

(II) Precision =>

$$\frac{TP}{TP + FP}$$

- Correct Positives Rate
- Percent of relevant results (relevancy)

(III) Specificity =

$$\frac{TN}{TN + FP}$$

- True Negative Rate

(IV) F1 Score :-

$$\frac{2TP}{2TP + FP + FN} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

= Harmonic mean of precision & sensitivity

(V) RMSE :-

- Root mean squared error:
- Accuracy measurement
- Only cares about right & ~~left~~ wrong answers

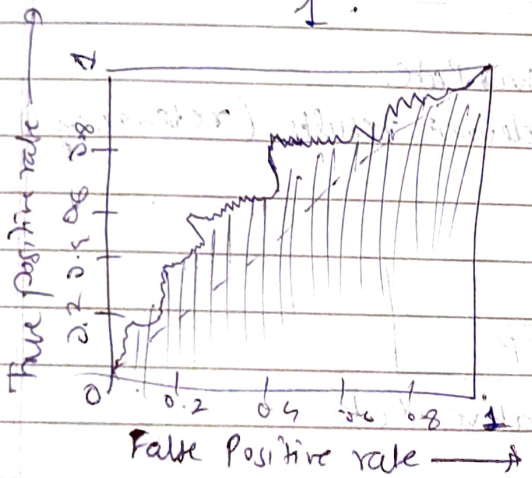
ROC Curve :- Receiver Operating Characteristic Curve

(1) Plot of TPR vs. FPR at various threshold settings.

(2) Points above the diagonal represents good classification (better than random).

(3) Area under the curve of ROC should be > 0.5. If it is less than that, then it's useless.

Ideal value of ROC curve AUC is 1.



ENSEMBLE LEARNING :-

Bagging: Generate N new training sets by random sampling with replacement.

Random sampling with replacement

Boosting :-

① Observations are weighted.

② Some will take part in new training sets more often.

③ Training is sequential; each classifier takes into account the previous one's success.

When to choose what?

Bagging : Vs : Boosting

① X.G.Boost is the latest hotness

② Boosting generally yields better accuracy.

① Bagging avoids overfitting

② Bagging is easier to parallelize

③ So, depends on your ~~god~~ goal.

Built-in Algorithms in Sagemaker:

1. Linear Learner Algorithms -

Uses

↳ Linear Regression :-

↳ Fit a line to your train data

↳ Predictions based on that line

↳ Can handle both regression & classification predictions

→ for classification a linear threshold is used.

→ Can do binary or multi class.

I/p format for this:

→ RecordIO - wrapped protobuf

→ CSV format (1st row assumed as header)

→ file or pipe mode both supported.

Special things about Linear Learner:

① Preprocessing :—

1. Traning data must be normalized so that all features are weighted the same.
2. Linear Learner can do this automatically.
3. Input data should be shuffled.

② Training :—

1. Uses SGD (Stochastic Gradient Descent)
2. Choose an optimization algo. (Adam, AdaGrad, etc)
3. Multiple models are optimized in parallel.
4. Tune L1 & L2 regularization.

③ Validation :—

1. Most optimal model is selected.

Important Hyperparameters For LinearLearner :—

i) balance_multiclass_weights
ii) learning_rate, mini_batch_size
iii) L1 ⇒ regularization
iv) wd ⇒ weight decay (L2 regularizat^n.)

Instances for XGBoost
- Uses CPUs only for training
- Memory bound, not compute bound
→ So, M5 is a good choice

For XGBoost 1-2, => P3 (GPU training)

(II) XGBoost Algorithm:

→ Boosted group of decision trees.

→ Models are serialized / deserialized using pickles.

Important Hyperparameters:

i) subsample – prevents overfitting

ii) eta – step size shrinkage, prevents overfitting

iii) gamma – Min. loss reduc^n to create a partition
– larger = more conservative

iv) alpha – L1 regularizaⁿ term.
– larger = more conservative

v) lambda – L2 regularization;
= larger = more conservative

vi) eval_metric = optimize on AUC, error, rmse, etc.
Ex: if you care more about false positives, choose, AUC.

vii) scaled_pos_weight =
– adjusts balance of +ve & -ve weights.
– helpful for unbalanced classes.

viii) max_depth =
- max depth of the tree.
- too high = you may overfit.

(iii) Seq2Seq Algorithm.

* I/p is a sequence of tokens, o/p is a seq. of token
* Used mainly for:
  i) M/c translation
  ii) Text summarization
  iii) Speech to text
  iv) Implemented with CNNs & RNNs.

I/p format for Seq2Seq ⇒ RecordIO-Protobuf
→ Start with tokenized text files

* Hyperparameters of Seq2Seq Important
  i) batch_size
  ii) optimizer_type (adam, sgd, rmsprop)
  iii) learning_rate
  iv) num_layers_encoder
  v) num_layers_decoder

(iv) DeepAR:
1) Forecasting one-D time series
2) Uses RNNs
3) Allows you to train the same model over several related time series
4) finds freq & seasonality.

Hyperparameters:
i) context_length
ii) epochs
iii) batch_size
iv) learning_rate (v) num_cells

(VI) Blazing Text + Word2Vec

(i) Text Classificaⁿ

(VII) ~~Seq 2 Seq~~ Object 2 Vec

(VIII) Object Detection in Sagemaker

(IX) Image Classification in Sagemaker

(X) Semantic Segmentation in Sagemaker (Pixel level obj. class

(XI) Random Cut Forest in Sagemaker - (For Anomaly detect - Unsup

(XII) Neural Topic Model in Sagemaker - (For - organize documents into topics)

(XIII) LDA in Sagemaker - (Topic Modelling Algo.)

(XIV) KNN in Sagemaker -

For Simple Classificaⁿ or regression.

Find the 'K' closest points to a sample point & return the most frequent label.

Find the 'K' closest points to a sample point & return the avg. value.

(XV) K-Means → For unsupervised clustering.

(XVI) PCA in Sagemaker → For Dimensionality Reduc".

(XVII) Dealing with sparse data → Factorization Machines

→ Click prediction
→ Item recommendations

(XVIII) Reinforcement learning in Sagemaker:

Q-Learning (Implementation of R-I-Learning)

i) A set of environmentable states 's'
ii) A set of possible actions in those states 'a'
iii) A value of each state/action Q.
Start with Q values as 0

Good things → after given action → Q↑

Bad things after given action ⇒ Q↓.

Sagemaker Autopilot

Automates:-

i) Algorithm selection
ii) Data preprocessing
iii) Model tuning.
iv) All infrastructure

It does all the trial & error
More broadly called as AutoML

Sagemaker Model Monitor:-

↳ Get alerts on quality deviations on your deployed models
↳ Detect anomalies & outliers
↳ Detect new features
↳ No code is needed.
↳ Visualize Data Drift

AWL AI/ML SERVICES :-

1) Amazon Comprehend :-
   - i) NLP + Text Analytics
   - ii) Extract Keyphrases, entities, sentiments, syntax, topics, document classifications.
   - iii) Can train your own data.

2) Amazon Translate :

3) Amazon Transcribe :)
   - ↳ Speech to text

4) Amazon Polly
   - ↳ Text to Speech

5) Amazon Rekognition
   - ↳ Computer Vision Tool
   - ↳ Object & ~~Text~~ Scene Detection
   - ↳ Facial Analysis :- (Age, Emotion on the face, Gender, etc)
   - ↳ Celebrity Analysis / Recognition

6) Amazon Forecast:

↳ Time Series analysis
↳ automatically chooses optimal model to train data from:
  ↳ ARIMA
  ↳ DeepAR
  ↳ ETS
  ↳ NPTS
  ↳ Prophet

7) Amazon Lex:

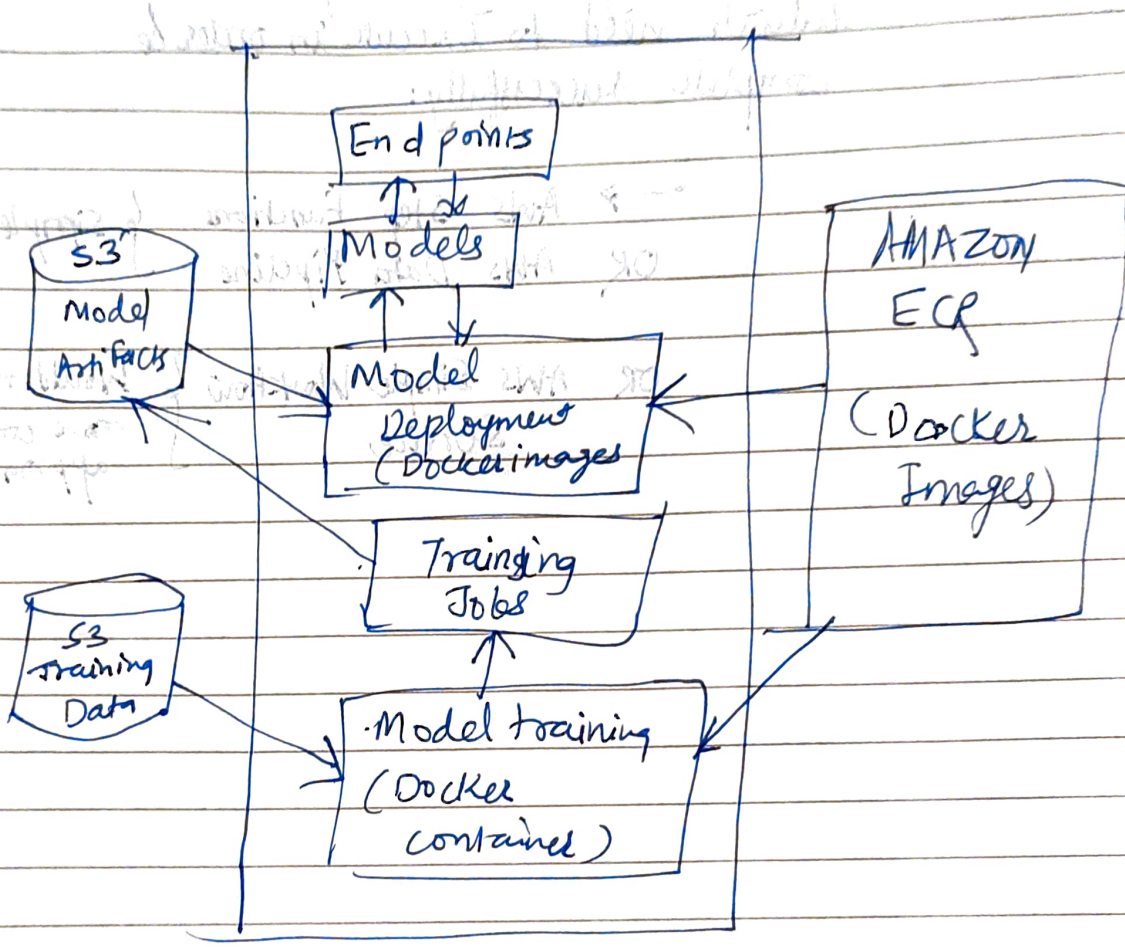
↳ Natural Language Chat-bot Engine

8) Amazon Personalize :-

↳ Fully managed Recommendation Engine

MLOps:

Using Sagemaker with Docker



$2 \cdot \frac{\frac{2}{3}(0.8)}{\frac{2}{3}+0.8} = 2 \cdot \frac{0.533 \times 0.8}{0.533+0.8} = 2 \cdot \frac{0.4264}{1.333} = 2 \times 0.32 = 0.64$

CASES & SERVICES THAT MIGHT BE CHOSEN:-

(I)

Multiple long-running ETL jobs which need to execute in order to complete successfully:

→ AWS Step Functions } Simpler
OR AWS Data Pipeline }

OR AWS Simple Workflow Service } Would require more complex approach