# BlazingText Algorithm

**[select 10 built-in algorithms]**

**Deadline: May 22, 2021**

**Summary**: [3 - 5 sentences describing the algorithm]

Blazing Text is considered to be the first step in delving into any kind of Natural Language Processing (NLP) processing with Text Classification, Natural Language Generators, and Sentiment Analysis being some of the prominent Word2Vec use cases that leverage the algorithm. Being a highly optimized implementation of the Word2Vec algorithm, Blazing Text is the optimal choice for many NLP applications as it enables the end user to scale large datasets easily while also offering end users the ability to train a model on more than a billion words in a couple of minutes using a multi-core CPU or a GPU. One of the main approaches that sets Blazing Text apart is the usage of word embeddings in the algorithm’s Unsupervised Learning Mode. Word embeddings are essentially a method that allows intelligence to be incorporated into Machine Learning (ML) projects by compressing sparse attributes data into a single fixed-length word embedding that can be directly fed into a ML model.

**Practical Problems it can solve:** [at least 2 examples]

* Sentiment Analysis
* Text Classification / Analysis
* Natural Language Generation
* Named Entity Recognition
* Machine Translation

**Terms and Definitions** [at least 3]:

* **word embedding** - A form of word representation that characterizes words to have a matching representation if they have a similar meaning.
* **Skip-gram** - An unsupervised learning technique used to find the most related words for a given target word by predicting the source context words.
* **continuous bag-of-words** - A model that attempts to predict the target word given the surrounding words and measuring their contextual accuracy.
* **Parallelizable** - Able to be processed in parallel on a shared-memory architecture.
* **N-grams** - A sequence of n words.
* **Out-of-vocabulary** - Words that are not part of the training text data
* **batch\_skipgram mode** - A feature in Blazing Text that BlazingText supports single or multiple CPU instances for skip-gram applications that enables shorter training durations.
* **Mini-batching** - Taking only a small and random subset of the data per iteration.
* **Negative Sampling**- A technique that allows the end user to only modify a small percentage of the weights for each iteration during training. This essentially maximizes the ratio of the similarity between the surrounding words and target word with the similarity of all other surrounding words and the target word by just randomly selecting some surrounding words instead of utilizing all of them.
* **BLAS operations** - Basic Linear Algebra Subroutines are operations that provide the standard building blocks for performing basic vector and matrix operations.
* **Word2Vec** - An NLP technique utilized to generate word embeddings to group similar words together.

**Notes: Training**

* **Train with File Mode**

\_\_label\_\_4 linux ready for prime time , intel says , despite all the linux hype , the open-source movement has yet to make a huge splash in the desktop market . that may be about to change , thanks to chipmaking giant intel corp .

\_\_label\_\_2 bowled by the slower one again , kolkata , november 14 the past caught up with sourav ganguly as the indian skippers return to international cricket was short lived .

* **Train with Augmented Manifest Text Format**

{"source":"linux ready for prime time , intel says , despite all the linux hy", "label":1}

{"source":"bowled by the slower one again , kolkata , november 14 the past caught up with sourav ganguly", "label":2}

* **Multi-label training**

{"source":"linux ready for prime time , intel says , despite all the linux hype", "label": [1, 3]}

{"source":"bowled by the slower one again , kolkata , november 14 the past caught up with sourav ganguly", "label": [2, 4, 5]}

**Notes: Inference**

{

"instances": ["the movie was excellent", "i did not like the plot ."]

}

{

"instances": ["the movie was excellent", "i did not like the plot ."],

"configuration": {"k": 2}

}

**Example**

* **Generating word vectors with BlazingText**

<https://github.com/aws/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/blazingtext_word2vec_text8/blazingtext_word2vec_text8.ipynb>

# XGBoost Algorithm

**Summary**:

The eXtreme Gradient Boosting or XGBoost is a gradient boosted trees algorithm based ensemble ML algorithm. The XGBoost algorithm is arguably one of the most prominent ML algorithms due to its high accuracy and low training times in numerous ML applications. It achieves this through scalability by enabling accelerated training times through parallel and distributed computing. In essence, the XGBoost algorithm utilizes an ensemble learning method which is a learning method that leverages a pool of smaller and weaker models to generate estimates.

**Practical Problems it can solve:** [at least 2 examples]

* Anomaly Detection
* Binary / Multi-class classification
* Regression
* Classification
* Ranking
* Server Load Forecasting
* Anomaly Detection

**Terms and Definitions** [at least 3]:

* **Ensemble learning** - An ML technique that combines multiple models to augment the overall performance of the algorithm to accurately predict a target variable.
* **estimator** - A function used to determine the model that is most likely to have the highest accuracy by generating estimates.
* **Bagging** - Bootstrap aggregating or bagging is an ensemble meta-algorithm that mitigates overfitting and reduces the variance of a decision tree by training weak models independently in parallel and using a deterministic averaging process to combine them afterwards.
* **Gradient boosting** - An ML technique used for building predictive models by optimizing a loss function, utilizing weak learners to generate inferences, and using an additive model to add weak learners to minimize the loss function.
* **Greedy algorithm** - A simple and intuitive algorithmic paradigm that makes the optimal choice at every step in an attempt to determine the global optimum or the overall most optimal solution to the problem.
* **Decision trees** - A supervised ML method that continuously splits the input data based on certain parameters.

**Notes: Training**

* **The SageMaker implementation of XGBoost supports CSV and libsvm formats for training and inference:**
* For Training ContentType, valid inputs are text/libsvm (default) or text/csv.
* For Inference ContentType, valid inputs are text/libsvm (default) or text/csv.

**Examples**

* **Multiclass classification with Amazon SageMaker XGBoost algorithm**

<https://github.com/aws/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/xgboost_mnist/xgboost_mnist.ipynb>

# K-Means Algorithm

**Summary**: [3 - 5 sentences describing the algorithm]

The K-Means algorithm is a distance-based algorithm that tries to minimize the distance of the points in a cluster with their centroid. This is done by initially choosing *k* number of clusters followed by selecting *k* random points from the data to be used as centroids. Afterwards, the third step is to assign each point to the closest centroid in order to form clusters. The fourth and final step is to recompute the centroids of newly-formed clusters. The third and fourth step are repeated and will stop depending on the specified stopping criteria.

**Practical Problems it can solve:** [at least 2 examples]

* Text Clustering
* Market Segmentation
* Document Clustering
* Image Segmentation
* Image Compression
* Recommendation Engines

**Terms and Definitions** [at least 3]:

* **centroid** - The centermost imaginary or real data point in a cluster.
* **inertia** - Based on the scikit-learn documentation, the KMeans inertia is recognized as a benchmark to determine if a cluster is optimal or not.
* **Dunn Index** - The ratio of the minimum of inter-cluster distances and maximum of intracluster distances used to evaluate a clustering algorithm.
* **Vector Quantization** - A signal processing technique used to quantize signal vectors by mapping k-dimensional vectors in the vector space into a finite set of vectors.

**Notes: Training**

* **Both recordIO-wrapped-protobuf and CSV formats are supported for training. You can use either File mode or Pipe mode to train models on data that is formatted as recordIO-wrapped-protobuf or as CSV.**

{"source":"linux ready for prime time , intel says , despite all the linux hy", "label":1}

{"source":"bowled by the slower one again , kolkata , november 14 the past caught up with sourav ganguly", "label":2}

**Notes: Inference**

* **JSON Response Format**

{

"predictions": [

{

"closest\_cluster": 1.0,

"distance\_to\_cluster": 3.0,

},

{

"closest\_cluster": 2.0,

"distance\_to\_cluster": 5.0,

},

....

]

}

* **JSONLINES Response Format**

{"closest\_cluster": 1.0, "distance\_to\_cluster": 3.0}

{"closest\_cluster": 2.0, "distance\_to\_cluster": 5.0}

* **RECORDIO Response Format**

[

Record = {

features = {},

label = {

'closest\_cluster': {

keys: [],

values: [1.0, 2.0] # float32

},

'distance\_to\_cluster': {

keys: [],

values: [3.0, 5.0] # float32

},

}

}

]

* **CSV Response Format**

The first value in each line corresponds to closest\_cluster. The second value in each line corresponds to distance\_to\_cluster.

1.0,3.0

2.0,5.0

**Example**

* **K-means clustering with Amazon SageMaker**

<https://aws.amazon.com/blogs/machine-learning/k-means-clustering-with-amazon-sagemaker/>

# Sequence-to-Sequence Algorithm

**Summary**: [3 - 5 sentences describing the algorithm]

The SageMaker seq2seq algorithm is a supervised learning algorithm that generates a sequence of tokens from another sequence of tokens that it accepts as input. Seq2seq uses Recurrent Neural Networks and Convolutional Neural Network models with encoder-decoder architectures in applications that include machine translation, abstractive summarization, and image captioning. One of the algorithm’s main disadvantages stems from it being based off of an encoder-decoder framework as the model performance decreases as and when the length of the source sequence increases because of the limit of how much information the fixed-length encoded feature vector can contain.

**Practical Problems it can solve:** [at least 2 examples]

* Machine translation
* Speech-to-text conversion
* Text summarization
* Image captioning

**Terms and Definitions** [at least 3]:

* **Tokens** - The building blocks of NLP. Normally in the form of words, characters, or subwords
* **Tokenization** - The process of separating a chunk of text into smaller units called tokens.
* **Transformer** - An architecture for transforming one sequence into another one with the help of two parts (Encoder and Decoder) that doesn’t imply any Recurrent Networks (GRU, LSTM, etc.).
* **Transfer Learning** - An ML method that reuses a model specifically developed for a task as the starting block for another model in the subsequent task.
* **Corpus** - A large and structured collection of machine-readable texts that have been produced in a natural communicative setting.

**Notes: Training**

SageMaker seq2seq expects data in **RecordIO-Protobuf format**. However, the tokens are expected as **integers**, not as floating points, as is usually the case.

A script to convert data from tokenized text files to the protobuf format is included in the seq2seq example notebook. In general, it packs the data into 32-bit integer tensors and generates the necessary vocabulary files, which are needed for metric calculation and inference.

After preprocessing is done, the algorithm can be invoked for training. The algorithm expects three channels:

**train**: It should contain the training data (for example, the train.rec file generated by the preprocessing script).

**validation**: It should contain the validation data (for example, the val.rec file generated by the preprocessing script).

**vocab**: It should contain two vocabulary files (vocab.src.json and vocab.trg.json)

If the algorithm doesn't find data in any of these three channels, training results in an error.

**Notes: Inference**

For hosted endpoints, inference supports two data formats. To perform inference using space separated text tokens, use the application/json format. Otherwise, use the recordio-protobuf format to work with the integer encoded data. Both modes support batching of input data. application/json format also allows you to visualize the attention matrix.

* **application/json**: Expects the input in JSON format and returns the output in JSON format. Both content and accept types should be application/json. Each sequence is expected to be a string with whitespace separated tokens. This format is recommended when the number of source sequences in the batch is small. It also supports the following additional configuration options:  
  configuration: {attention\_matrix: true}: Returns the attention matrix for the particular input sequence.
* **application/x-recordio-protobuf**: Expects the input in recordio-protobuf format and returns the output in recordio-protobuf format. Both content and accept types should be applications/x-recordio-protobuf. For this format, the source sequences must be converted into a list of integers for subsequent protobuf encoding. This format is recommended for bulk inference.

For **batch transform**, inference supports JSON Lines format. Batch transform expects the input in JSON Lines format and returns the output in JSON Lines format. Both content and accept types should be application/jsonlines. The format for input is as follows:

content-type: application/jsonlines

{"source": "source\_sequence\_0"}

{"source": "source\_sequence\_1"}

The format for response is as follows:

accept: application/jsonlines

{"target": "predicted\_sequence\_0"}

{"target": "predicted\_sequence\_1"}

**Example**

**Machine Translation English-German Example Using SageMaker Seq2Seq**

<https://github.com/aws/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/seq2seq_translation_en-de/SageMaker-Seq2Seq-Translation-English-German.ipynb>

# Object2Vec Algorithm

**Summary**: [3 - 5 sentences describing the algorithm]

The Object2Vec algorithm is one of the built-in algorithms in Amazon SageMaker. It is a highly customizable neural embedding algorithm that can learn low-dimensional dense embeddings of high-dimensional objects. One of the main benefits of using this algorithm is that it is able to generalize the well-known Word2Vec embedding technique for words that is optimized in the SageMaker BlazingText algorithm.

**Practical Problems it can solve:** [at least 2 examples]

* Text Clustering
* Market Segmentation
* Document Clustering
* Image Segmentation
* Image Compression
* Recommendation Engines

**Terms and Definitions** [at least 3]:

* **embedding** - A method used to represent discrete variables as continuous vectors. An embedding is essentially a translation of a high-dimensional vector into a low-dimensional space.
* **Long short-term memory** - Networks that are a type of recurrent neural network that are able to learn order dependence in sequence prediction problems.
* **Cross-entropy loss function** - Used to measure the performance of a classification model whose output is a probability value between 0 and 1.

**Notes: Training**

* **Input: JSON Lines Request Format**

Content-type: application/jsonlines

{"label": 0, "in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4], "in1": [16, 21, 13, 45, 14, 9, 80, 59, 164, 4]}

{"label": 1, "in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4], "in1": [22, 32, 13, 25, 1016, 573, 3252, 4]}

{"label": 1, "in0": [774, 14, 21, 206], "in1": [21, 366, 125]}

**Notes: Inference**

* **Input: Classification or Regression Request Format**

Content-type: application/json

{

"instances" : [

{"in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4], "in1": [16, 21, 13, 45, 14, 9, 80, 59, 164, 4]},

{"in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4], "in1": [22, 32, 13, 25, 1016, 573, 3252, 4]},

{"in0": [774, 14, 21, 206], "in1": [21, 366, 125]}

]

}

Content-type: application/jsonlines

{"in0": [6, 17, 606, 19, 53, 67, 52, 12, 5, 10, 15, 10178, 7, 33, 652, 80, 15, 69, 821, 4], "in1": [16, 21, 13, 45, 14, 9, 80, 59, 164, 4]}

{"in0": [22, 1016, 32, 13, 25, 11, 5, 64, 573, 45, 5, 80, 15, 67, 21, 7, 9, 107, 4], "in1": [22, 32, 13, 25, 1016, 573, 3252, 4]}

{"in0": [774, 14, 21, 206], "in1": [21, 366, 125]}

* **Output: Classification or Regression Response Format**

Accept: application/json

{

"predictions": [

{

"scores": [

0.6533935070037842,

0.07582679390907288,

0.2707797586917877

]

},

{

"scores": [

0.026291321963071823,

0.6577019095420837,

0.31600672006607056

]

}

]

}

Accept: application/jsonlines

{"scores":[0.195667684078216,0.395351558923721,0.408980727195739]}

{"scores":[0.251988261938095,0.258233487606048,0.489778339862823]}

{"scores":[0.280087798833847,0.368331134319305,0.351581096649169]}

**Example**

* **An Introduction to SageMaker ObjectToVec model for MovieLens recommendation**

<https://github.com/aws/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/object2vec_movie_recommendation/object2vec_movie_recommendation.ipynb>

# Object Detection Algorithm

**Summary**: [3 - 5 sentences describing the algorithm]

The SageMaker Object Detection Algorithm is a supervised learning algorithm that uses a single deep neural network to detect and classify objects in images. It accepts images as input, identifies all instances of the object based on a predetermined pool of categories, and yields which category the detected object belongs to. Furthermore, a rectangular bounding box is also indicated on the image which is a prediction of the object’s location and scale.

**Practical Problems it can solve:** [at least 2 examples]

* People Counting
* Robot Sensing
* Self-driving cars
* Vehicle Detection

**Terms and Definitions** [at least 3]:

* **Single Shot multibox Detector (SSD)** - A method for detecting objects in images using a single deep neural network.
* **VGG** - An acclaimed convolutional neural network model that achieved a 92.7% top-5 test accuracy in ImageNet, a dataset of over 14 million images belonging to 1000 classes.
* **ResNet-50** - A 50-layer-deep residual network that is commonly used in image processing applications.

**Notes: Training**

* **Train with the RecordIO Format**

specify both **train and validation channels** as values for the InputDataConfig parameter of the CreateTrainingJob request. Specify one **RecordIO (.rec)** file in the **train** channel and one **RecordIO** file in the **validation** channel. Set the content type for both channels to application/x-recordio.

* **Train with the Image Format**

{

"file": "your\_image\_directory/sample\_image1.jpg",

"image\_size": [

{

"width": 500,

"height": 400,

"depth": 3

}

],

"annotations": [

{

"class\_id": 0,

"left": 111,

"top": 134,

"width": 61,

"height": 128

},

{

"class\_id": 0,

"left": 161,

"top": 250,

"width": 79,

"height": 143

},

{

"class\_id": 1,

"left": 101,

"top": 185,

"width": 42,

"height": 130

}

],

"categories": [

{

"class\_id": 0,

"name": "dog"

},

{

"class\_id": 1,

"name": "cat"

}

]

}

* **Train with Augmented Manifest Image Format**

{"source-ref": "s3://your\_bucket/image1.jpg", "bounding-box":{"image\_size":[{ "width": 500, "height": 400, "depth":3}], "annotations":[{"class\_id": 0, "left": 111, "top": 134, "width": 61, "height": 128}, {"class\_id": 5, "left": 161, "top": 250, "width": 80, "height": 50}]}, "bounding-box-metadata":{"class-map":{"0": "dog", "5": "horse"}, "type": "groundtruth/object-detection"}}

{"source-ref": "s3://your\_bucket/image2.jpg", "bounding-box":{"image\_size":[{ "width": 400, "height": 300, "depth":3}], "annotations":[{"class\_id": 1, "left": 100, "top": 120, "width": 43, "height": 78}]}, "bounding-box-metadata":{"class-map":{"1": "cat"}, "type": "groundtruth/object-detection"}}

**Notes: Inference**

* **Request Format**

Query a trained model by using the model's endpoint. The endpoint takes .jpg and .png image formats with image/jpeg and image/png content-types.

* **Response Formats**

The response is the class index with a confidence score and bounding box coordinates for all objects within the image encoded in JSON format. The following is an example of response .json file:

{"prediction":[

[4.0, 0.86419455409049988, 0.3088374733924866, 0.07030484080314636, 0.7110607028007507, 0.9345266819000244],

[0.0, 0.73376623392105103, 0.5714187026023865, 0.40427327156066895, 0.827075183391571, 0.9712159633636475],

[4.0, 0.32643985450267792, 0.3677481412887573, 0.034883320331573486, 0.6318609714508057, 0.5967587828636169],

[8.0, 0.22552496790885925, 0.6152569651603699, 0.5722782611846924, 0.882301390171051, 0.8985623121261597],

[3.0, 0.42260299175977707, 0.019305512309074402, 0.08386176824569702, 0.39093565940856934, 0.9574796557426453]

]

* **Detection results**

{"prediction": [[label\_id, confidence\_score, xmin, ymin, xmax, ymax], [label\_id, confidence\_score, xmin, ymin, xmax, ymax]]}

* **OUTPUT: JSON Response Format**

accept: application/json;annotation=1

{

"image\_size": [

{

"width": 500,

"height": 400,

"depth": 3

}

],

"annotations": [

{

"class\_id": 0,

"score": 0.943,

"left": 111,

"top": 134,

"width": 61,

"height": 128

},

{

"class\_id": 0,

"score": 0.0013,

"left": 161,

"top": 250,

"width": 79,

"height": 143

},

{

"class\_id": 1,

"score": 0.0133,

"left": 101,

"top": 185,

"width": 42,

"height": 130

}

]

}

**Example**

* **Amazon SageMaker Object Detection for Bird Species**

<https://github.com/aws/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/object_detection_birds/object_detection_birds.ipynb>

# K-Nearest Neighbors Algorithm

**Summary**: [3 - 5 sentences describing the algorithm]

Amazon SageMaker’s kNN algorithm is an index-based algorithm that utilizes a non-parametric method for classification or regression. The basis that the algorithm leans toward is that similar data points should belong to the same class in most cases. The flow of the algorithm may be summarized in three steps, namely sampling, dimension reduction, and index building.

**Practical Problems it can solve:** [at least 2 examples]

* Semantic Search
* Recommendation systems
* Image / Text classification
* Anomaly detection

**Terms and Definitions** [at least 3]:

* **index-based** - Uses the index values of each data point as basis.
* **non-parametric method**- In statistics, this method is a mathematical approach for statistical inferences that do not consider the underlying assumptions on the shape of the probability distribution of the observation under study.
* **random projection** - An approach that uses decomposition to reduce the dimensionality of high-dimensional data.
* **fast Johnson-Lindenstrauss transform** - A low-distortion embedding that is faster than standard rando projections but is just as easy to implement.

**Notes: Training**

* **CSV Data Format**

content-type: text/csv; label\_size=1

4,1.2,1.3,9.6,20.3

* **RECORDIO Data Format**

content-type: application/x-recordio-protobuf

[

Record = {

features = {

'values': {

values: [1.2, 1.3, 9.6, 20.3] # float32

}

},

label = {

'values': {

values: [4] # float32

}

}

}

]

}

**Notes: Inference**

* **INPUT: CSV Request Format**

content-type: text/csv

1.2,1.3,9.6,20.3

* **INPUT: JSON Request Format**

content-type: application/json

{

"instances": [

{"data": {"features": {"values": [-3, -1, -4, 2]}}},

{"features": [3.0, 0.1, 0.04, 0.002]}]

}

* **INPUT: JSONLINES Request Format**

content-type: application/jsonlines

{"features": [1.5, 16.0, 14.0, 23.0]}

{"data": {"features": {"values": [1.5, 16.0, 14.0, 23.0]}}

* **INPUT: RECORDIO Request Format**

content-type: application/x-recordio-protobuf

[

Record = {

features = {

'values': {

values: [-3, -1, -4, 2] # float32

}

},

label = {}

},

Record = {

features = {

'values': {

values: [3.0, 0.1, 0.04, 0.002] # float32

}

},

label = {}

},

]

* **OUTPUT: JSON Response Format**

accept: application/json

{

"predictions": [

{"predicted\_label": 0.0},

{"predicted\_label": 2.0}

]

}

* **OUTPUT: JSONLINES Response Format**

accept: application/jsonlines

{"predicted\_label": 0.0}

{"predicted\_label": 2.0}

* **OUTPUT: VERBOSE JSON Response Format**

accept: application/json; verbose=true

{

"predictions": [

{

"predicted\_label": 0.0,

"distances": [3.11792408, 3.89746071, 6.32548437],

"labels": [0.0, 1.0, 0.0]

},

{

"predicted\_label": 2.0,

"distances": [1.08470316, 3.04917915, 5.25393973],

"labels": [2.0, 2.0, 0.0]

}

]

}

* **OUTPUT: RECORDIO-PROTOBUF Response Format**

content-type: application/x-recordio-protobuf

[

Record = {

features = {},

label = {

'predicted\_label': {

values: [0.0] # float32

}

}

},

Record = {

features = {},

label = {

'predicted\_label': {

values: [2.0] # float32

}

}

}

]

* **OUTPUT: VERBOSE RECORDIO-PROTOBUF Response Format**

accept: application/x-recordio-protobuf; verbose=true

[

Record = {

features = {},

label = {

'predicted\_label': {

values: [0.0] # float32

},

'distances': {

values: [3.11792408, 3.89746071, 6.32548437] # float32

},

'labels': {

values: [0.0, 1.0, 0.0] # float32

}

}

},

Record = {

features = {},

label = {

'predicted\_label': {

values: [0.0] # float32

},

'distances': {

values: [1.08470316, 3.04917915, 5.25393973] # float32

},

'labels': {

values: [2.0, 2.0, 0.0] # float32

}

}

}

]

* **SAMPLE OUTPUT for the k-NN Algorithm**

For regressor tasks:

[06/08/2018 20:15:33 INFO 140026520049408] #test\_score (algo-1) : ('mse', 0.013333333333333334)

For classifier tasks:

[06/08/2018 20:15:46 INFO 140285487171328] #test\_score (algo-1) : ('accuracy', 0.98666666666666669)

**Example**

* **Amazon SageMaker multi-class classification using kNN**

<https://github.com/aws/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/k_nearest_neighbors_covtype/k_nearest_neighbors_covtype.ipynb>

# DeepAR Forecasting Algorithm

**Summary**: [3 - 5 sentences describing the algorithm]

SageMaker’s DeepAR Forecasting algorithm primarily uses recurrent neural networks to forecast scalar time series data. One of the main differences that sets DeepAR apart from classic forecasting algorithms is that it is able to train a single model jointly over all of the time series. This is beneficial when the existing dataset contains hundreds of related time series.

The Amazon SageMaker DeepAR forecasting algorithm is a supervised learning algorithm for forecasting scalar (one-dimensional) time series using recurrent neural networks (RNN). .

**Practical Problems it can solve:** [at least 2 examples]

* Web Page Request Forecasting
* Sales Forecasting
* Server Load Forecasting

**Terms and Definitions** [at least 3]:

* **recurrent neural networks (RNN)** - A class of neural networks that uses sequential or time series data that allow previous outputs to be used as inputs while having hidden states.
* **autoregressive integrated moving average (ARIMA)** - A statistical analysis model that uses time-series data and statistical analysis to interpret the data to predict future trends.
* **exponential smoothing (ETS)** - A commonly-used local statistical algorithm for time-series forecasting.

**Notes: Training**

* **JSON Lines**

{"start": "2009-11-01 00:00:00", "target": [4.3, "NaN", 5.1, ...], "cat": [0, 1], "dynamic\_feat": [[1.1, 1.2, 0.5, ...]]}

{"start": "2012-01-30 00:00:00", "target": [1.0, -5.0, ...], "cat": [2, 3], "dynamic\_feat": [[1.1, 2.05, ...]]}

{"start": "1999-01-30 00:00:00", "target": [2.0, 1.0], "cat": [1, 4], "dynamic\_feat": [[1.3, 0.4]]}

* **Parquet**

{ "start": ..., "target": ..., "cat": [0, 0], ... } # red shoes

{ "start": ..., "target": ..., "cat": [1, 1], ... } # blue dress

{ "start": ..., "target": ..., "cat": [0, 1], ... } # blue shoes

{ "start": ..., "target": ..., "cat": [1, 0], ... } # red dress

**Notes: Inference**

* **JSON Request Format**

{

"instances": [

{

"start": "2009-11-01 00:00:00",

"target": [4.0, 10.0, "NaN", 100.0, 113.0],

"cat": [0, 1],

"dynamic\_feat": [[1.0, 1.1, 2.1, 0.5, 3.1, 4.1, 1.2, 5.0, ...]]

},

{

"start": "2012-01-30",

"target": [1.0],

"cat": [2, 1],

"dynamic\_feat": [[2.0, 3.1, 4.5, 1.5, 1.8, 3.2, 0.1, 3.0, ...]]

},

{

"start": "1999-01-30",

"target": [2.0, 1.0],

"cat": [1, 3],

"dynamic\_feat": [[1.0, 0.1, -2.5, 0.3, 2.0, -1.2, -0.1, -3.0, ...]]

}

],

"configuration": {

"num\_samples": 50,

"output\_types": ["mean", "quantiles", "samples"],

"quantiles": ["0.5", "0.9"]

}

}

* **JSON Response Formats**

{

"predictions": [

{

"quantiles": {

"0.9": [...],

"0.5": [...]

},

"samples": [...],

"mean": [...]

},

{

"quantiles": {

"0.9": [...],

"0.5": [...]

},

"samples": [...],

"mean": [...]

},

{

"quantiles": {

"0.9": [...],

"0.5": [...]

},

"samples": [...],

"mean": [...]

}

]

}

* **Batch Transform**

{"start": "2009-11-01 00:00:00", "target": [4.3, "NaN", 5.1, ...], "cat": [0, 1], "dynamic\_feat": [[1.1, 1.2, 0.5, ..]]}

{"start": "2012-01-30 00:00:00", "target": [1.0, -5.0, ...], "cat": [2, 3], "dynamic\_feat": [[1.1, 2.05, ...]]}

{"start": "1999-01-30 00:00:00", "target": [2.0, 1.0], "cat": [1, 4], "dynamic\_feat": [[1.3, 0.4]]}

* **OUTPUT: JSON Response Format**

{ "quantiles": { "0.1": [...], "0.2": [...] }, "samples": [...], "mean": [...] }

**Example**

* **SageMaker/DeepAR demo on electricity dataset¶**

<https://github.com/aws/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/deepar_electricity/DeepAR-Electricity.ipynb>

# Linear Learner Algorithm

**Summary**: [3 - 5 sentences describing the algorithm]

The SageMaker Linear Learner algorithm is the established algorithm used for solving either classification or regression problems. The key feature that the SageMaker Linear Learner algorithm has over other linear models is that it can train multiple models in parallel. The algorithm checks which set best optimizes a given criteria from a pool of models that are configured with different hyperparameters.

**Practical Problems it can solve:** [at least 2 examples]

* Binary / multi-class classification
* Generate insights on consumer behaviour
* Analyze the marketing effectiveness
* Assess risk in financial services

**Terms and Definitions** [at least 3]:

* **stochastic gradient descent (SGD)** - A simple yet very efficient approach to fitting linear classifiers and regressors under convex loss functions such as (linear) Support Vector Machines and Logistic Regression. SGD has been very successful in large-scale and sparse machine learning problems applications that are often encountered in text classification and NLP.
* **Adam** - The Adam optimization algorithm is an alternative to the SGD algorithm. It combines the best properties of the AdaGrad and RMSProp algorithms which enables it to easily handle sparse gradients on noisy problems.
* **AdaGrad** - A gradient-based optimization algorithm that adapts the learning rate to the parameters. As a result, it has smaller learning rates for parameters that are linked with frequently occurring features. In contrast, it has higher learning rates for parameters associated with infrequent features.

**Notes: Training**

* For **training**, the linear learner algorithm supports both recordIO-wrapped protobuf and CSV formats. For the application/x-recordio-protobuf input type, only Float32 tensors are supported. For the text/csv input type, the first column is assumed to be the label, which is the target variable for prediction. You can use either File mode or Pipe mode to train linear learner models on data that is formatted as recordIO-wrapped-protobuf or as CSV.

**Notes: Inference**

* **JSON Request Format**

Binary Classification

let response = {

"predictions": [

{

"score": 0.4,

"predicted\_label": 0

}

]

}

Multiclass Classification

let response = {

"predictions": [

{

"score": [0.1, 0.2, 0.4, 0.3],

"predicted\_label": 2

}

]

}

Regression

let response = {

"predictions": [

{

"score": 0.4

}

]

}

* **JSONLINES response formats**

Binary Classification

{"score": 0.4, "predicted\_label": 0}

Multiclass Classification

{"score": [0.1, 0.2, 0.4, 0.3], "predicted\_label": 2}

Regression

{"score": 0.4}

* **RECORDIO response formats**

Binary Classification

[

Record = {

features = {},

label = {

'score': {

keys: [],

values: [0.4] # float32

},

'predicted\_label': {

keys: [],

values: [0.0] # float32

}

}

}

]

Multiclass Classification

[

Record = {

"features": [],

"label": {

"score": {

"values": [0.1, 0.2, 0.3, 0.4]

},

"predicted\_label": {

"values": [3]

}

},

"uid": "abc123",

"metadata": "{created\_at: '2017-06-03'}"

}

]

Regression

[

Record = {

features = {},

label = {

'score': {

keys: [],

values: [0.4] # float32

}

}

}

]

**Example**

* **An Introduction to Linear Learner with MNIST**

<https://github.com/aws/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/linear_learner_mnist/linear_learner_mnist.ipynb>

# Latent Dirichlet Allocation Algorithm

**Summary**: [3 - 5 sentences describing the algorithm]

The LDA algorithm is one of the most commonly used unsupervised algorithms to identify a predetermined number of topics shared by documents in a text corpus. Since it is unsupervised, only the number of topics may be specified; the actual topics cannot be listed down. The algorithm assumes that documents are formed by sampling words from the set number of topics.

**Practical Problems it can solve:** [at least 2 examples]

* Content recommendation
* Population analysis
* Taxonomy building
* Auto categorization

**Terms and Definitions** [at least 3]:

* **Dirichlet distribution** - A family of continuous multivariate probability distributions parameterized by a vector of positive reals used to model random probability mass functions for finite sets. It fundamentally used to measure the uncertainty of probabilities.
* **Pointwise Mutual Information (PMI)** - A measure used to gauge the association between a feature word and a category.
* **perplexity** - A benchmark for measuring how well a model is able to predict a sample.

**Notes: Training**

* LDA supports both **recordIO-wrapped-protobuf (dense and sparse)** and **CSV** file formats. For CSV, the data must be dense and have dimension equal to number of records \* vocabulary size. LDA can be trained in **File** or **Pipe** mode when using recordIO-wrapped protobuf, but only in File mode for the CSV format.

**Notes: Inference**

* For inference, **text/csv**, **application/json**, and **application/x-recordio-protobuf** content types are supported. Sparse data can also be passed for **application/json** and **application/x-recordio-protobuf**.
* LDA inference returns **application/json or application/x-recordio-protobuf predictions**, which include the topic\_mixture vector for each observation.

**Example**

* **An Introduction to SageMaker LDA**

<https://github.com/aws/amazon-sagemaker-examples/blob/master/introduction_to_amazon_algorithms/lda_topic_modeling/LDA-Introduction.ipynb>

***[perform this 10 times x 2-3 pages for each algorithm]***