MACHINE LEARNING SU AWS: COME CREARE E DEPLOYARE UN SERVIZIO CON AWS SAGEMAKER

AI | How-to | ML

beSharp | 20 Settembre 2019

Nell'ultimo decennio l'approccio alla gestione del dato è drasticamente cambiato: da un lato, grazie al diffondersi di servizi di public Cloud, **il costo di storage è sensibilmente diminuito.** Dall'altro, con l'utilizzo sempre più diffuso di ERPs, CRM, piattaforme IoT e altri software di monitoring e profilazione, la raccolta di dati è diventata estremamente più semplice; è possibile avere a disposizione **enormi quantità di dati** sia sui processi aziendali interni, che sulle preferenze e i comportamenti dei clienti. In altre parole, oggi abbiamo l'opportunità di sfruttare **quantità di informazioni sempre più grandi e di qualità sempre migliore** ad un costo infinitamente ridotto.

Negli ultimi anni la facilità crescente di costruzione di dataset di grandi dimensioni su cui poter effettuare analisi ha fatto aumentare enormemente l'interesse verso temi come **Intelligenza Artificiale e Machine Learning (ML),** che consentono di creare modelli predittivi a partire dai dataset.

La diffusione di dispositivi interconnessi e dell'IoT in generale ha aumentato le opportunità di generare e collezionare molti dati a basso costo. Da qui, grazie all'applicazione di tecniche di Machine Learning hanno preso vita innumerevoli possibilità, fino a qualche anno fa inimmaginabili. Si pensi ad esempio alla profilazione dei clienti: di loro oggi possiamo sapere praticamente tutto: quali prodotti usano, come li usano, quali azioni inizialmente non pensate dalle aziende compiono ogni giorno, quali aspetti di ciascun prodotto sono realmente rilevanti nella loro vita quotidiana e molto altro. Spostandosi dal quotidiano all'industria, poi, è facile ottenere informazioni rispetto ad esempio a quali componenti industriali siano più soggetti all'usura per agire quindi di conseguenza (manutenzione predittiva), sulla base di una semplice raccolta di immagini sapere in anticipo se le componenti meccaniche prodotte siano difettose oppure no, e così via.

La capacità di capire quali dati estrarre, di interpretarli correttamente e di sfruttarli al massimo per farne un uso il più possibile utile è ciò che fa realmente la differenza nell'applicazione di questo nuovo approccio e ciò che può davvero attribuire un vantaggio competitivo ad un'azienda rispetto ad un'azienda competitor.

I principali provider Cloud come **Amazon Web Services** oggi offrono già una vasta gamma di servizi gestiti dedicati al Machine Learning, utili a soddisfare la stragrande maggioranza dei casi d'uso richiesti dagli utenti.

Prima di entrare nel merito di AWS SageMaker quindi facciamo una veloce carrellata dei principali servizi offerti da AWS e delle loro potenzialità:

- **Amazon Comprehend:** "capisce" ed estrae informazioni a partire da un testo destrutturato (es.. mails, tickets, etc)
- **Amazon Lex:** il servizio dedicato allo sviluppo di interfacce di comunicazione come i Chatbot. (Ne abbiamo parlato in questo articolo)
- **Amazon Textract:** il servizio che permette di ottenere testo digitale a partire da documenti digitalizzati con o anche senza una struttura precisa
- **Amazon Translate:** il servizio per la traduzione automatica di testi da una lingua ad un'altra
- **Amazon Transcribe e Amazon Polly:** sono due servizi AWS impiegati nelle trasformazioni "speech to text" e/o "text to speech"
- **Amazon Recommendations and Forecasting:** il servizio che sfrutta Amazon.com per effettuare predizioni di acquisto utili agli shop online
- **Amazon Rekognition:** estrae informazioni da immagini e video.

Si tratta però di servizi *completamente gestiti*; per chi invece è in cerca di una **soluzione più personalizzabile e flessibile** con cui divertirsi ad esplorare le infinite potenzialità del Machine Learning, **ecco AWS SageMaker.**

Entriamo quindi nel dettaglio di questo servizio per poi analizzare, nella seconda parte dell'articolo, la struttura di un progetto basato su AWS SageMaker, come creare un semplice modello di previsione e come effettuare **il deploy di un servizio HTTP basato su un modello di Machine Learning.**

SageMaker: un Building Block

Dalla console di AWS SageMaker possiamo subito notare nella sidebar la lista di tutti **i componenti di SageMaker con le loro configurazioni.**

Il servizio AWS SageMaker è composto da *sottoservizi:*

- **Ground Truth:** questo servizio permette di etichettare un dataset esistente (ad esempio è possibile creare bounding boxes per un intero dataset di immagini). Per farlo il servizio sfrutta la human workforce messa a disposizione tramite Amazon Mechanical Turk. È anche possibile utilizzare modelli esistenti per etichettare dati e poi delegare alla human workforce esclusivamente gli elementi da classificare sui quali l'incertezza è elevata
- **Notebook:** è un servizio che permette di lanciare in pochi click una Jupyter notebook instance preconfigurata con le librerie più comuni di data mining e ML (con Tensorflow or PyTorch). È possibile avviare l'istanza sia all'interno di una VPC (per aumentare il controllo su accessi e reti), che fuori dalle VPC, così come accade per le Lambda functions.

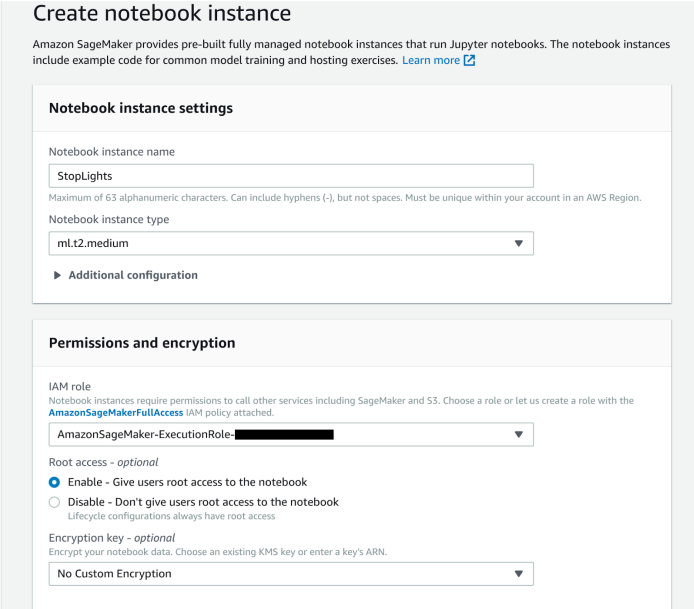
- **Training:** permette di effettuare il deploy di training jobs. I data scientist possono selezionare i modelli rilevanti da trainare, scegliendo tra moltissime librerie di modelli generici già esistenti. Una volta che il modello sarà *addestrato*, sarà possibile effettuare il deploy.
- **Models:** una volta che l'algoritmo è stato trainato sulla basa di un determinato training dataset, è possibile deployarlo usando la sezione models.

È possibile anche deployare progetti completi di SageMaker direttamente dall' AWS Marketplace per soddisfare tasks specifici.

I progetti di ML offerti su Marketplace sono affidabili e trattano un ampio insieme di casistiche; per questo è importante scegliere il progetto che più si addice alle proprie esigenze prima di procedere con lo sviluppo del progetto.

Un esempio di progetto Machine Learning

Quando si sviluppa un servizio basato su ML, il primo passo consiste nella **creazione di una Notebook instance** tramite la console AWS. Come mostrato nello screenshot, saranno richiesti un nome per l'istanza, la definizione dell'IAM Role che l'istanza utilizzerà e dovremo decidere se vogliamo o meno che la Notebook instance sia inserita in una VPC (In questo caso sarà possibile definire anche Security Group e Subnet)



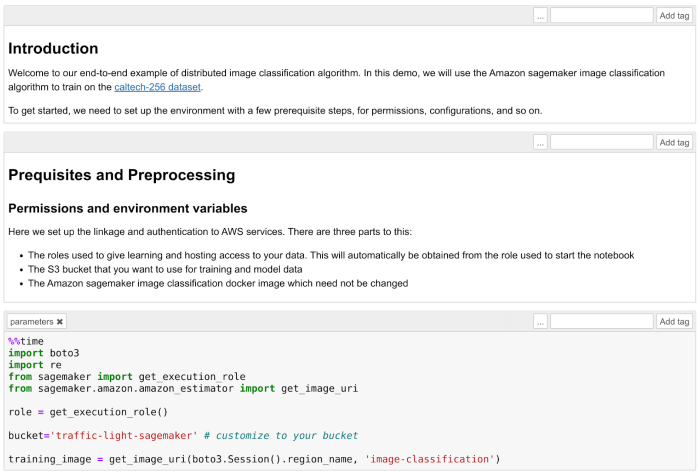
Anche se decidiamo di non inserire l'istanza in una VPC, **lo Jupiter notebook potrà essere raggiunto esclusivamente un pre-sign URL, generato dalle nostre credenziali AWS.** Una volta aperta la Jupiter notebook, ci troveremo di fronte all'interfaccia standard Jupiter Nootebook:

Nella "Example tab", AWS fornisce una lista di esempi per aiutare i data scientists a prendere confidenza con SageMaker.

Noi abbiamo scelto di seguire l'esempio di Image Classification utilizzando il dataset **Caltech 256.**

Clicchiamo su ***Use Image-classification-fulltraining.ipynb e cominciamo.***

Verrà mostrato un normale Jupiter Notebook. Andiamo semplicemente a seguire le istruzioni fornite:



Il primo passo sarà **creare un bucket S3.** Assicuriamoci quindi che il ruolo che abbiamo assegnato alla Notebook instance abbia i **permessi di lettura e scrittura** su quel bucket.

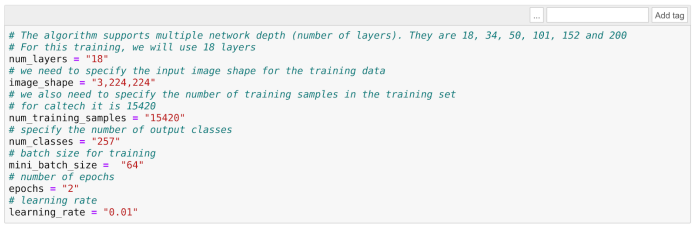
Avviato il template, **scaricheremo il dataset Caltech 256** e prepareremo il training job per l'algoritmo di image classification. L'algoritmo utilizzato in questo esempio è l'algoritmo **AWS Image Classification** il quale si basa su una rete neurale convoluzionale (ResNet). AWS fornisce un ampio array di algoritmi generali ottimizzati per SageMaker (Per più info, leggi la documentazione di AWS). In ogni caso sarà comunque possibile deployare un modello custom da trainare all'interno di un docker container.

È importante comprendere come splittare il dataset in due parti: una parte per il **training** e una parte di **validazione**. In questo esempio ogni classe dell'immagine (ball, bathtub, Mars, riffle, ecc) verrà divisa in due set: 60 immagini di training e le restanti per la validazione.

Prima di iniziare col training, è fondamentale **scegliere con cura gli Hyperparameters sensibili:** una scelta non accurata avrà come risultato un algoritmo non performante. La scelta di questi parametri è spesso una questione di "prove ed errori"; per questa particolare applicazione, i parametri più rilevanti saranno learning rate ed epochs number.

Il **Learning Rate** non è altro che la lunghezza del vettore dell'algoritmo di Steepest Descent: se troppo alta, avremo oltrepassato l'optimum, se troppo bassa il training andrà avanti all'infinito e non otterremo mai un risultato stabile. L'**epochs number** è invece il valore che indica quante volte l'intero dataset è passato avanti e indietro attraverso la rete neurale. Un epoch number troppo

basso significherà che il modello non avrà appreso abbastanza e otterremo quindi cattivi risultati ma il tempo di training sarà breve. Al contrario, un epoch numer alto porterà ad un overfitting dei dati e perciò di nuovo a risultati poco accurati e a un tempo di apprendimento molto lungo. **Sia l'overfitting, che l'underfitting sono negativi** in quanto causeranno una cattiva performance del modello trainato durante la validazione dei dati.



```
# The algorithm supports multiple network depth (number of layers). They are 18, 34, 50, 101, 152 and 200
# For this training, we will use 18 layers
num_layers = "18"
# we need to specify the input image shape for the training data
image_shape = "3,224,224"
# we also need to specify the number of training samples in the training set
# for caltech it is 15420
num_training_samples = "15420"
# specify the number of output classes
num_classes = "257"
# batch size for training
mini_batch_size =  "64"
# number of epochs
epochs = "2"
# learning rate
learning_rate = "0.01"
```

Ad esempio, utilizzando i valori di default forniti da AWS, in questo caso il tempo di training è molto basso (una manciata di minuti), ma il modello classificherà ogni immagine vagamente rossiccia o appena rotondeggiante come... Marte 🙂

```
# create the Amazon SageMaker training job
sagemaker = boto3.client(service_name='sagemaker')
sagemaker.create_training_job(**training_params)

# confirm that the training job has started
status = sagemaker.describe_training_job(TrainingJobName=job_name)['TrainingJobStatus']
print('Training job current status: {}'.format(status))

try:
    # wait for the job to finish and report the ending status
    sagemaker.get_waiter('training_job_completed_or_stopped').wait(TrainingJobName=job_name)
    training_info = sagemaker.describe_training_job(TrainingJobName=job_name)
    status = training_info['TrainingJobStatus']
    print("Training job ended with status: " + status)
except:
    print('Training failed to start')
     # if exception is raised, that means it has failed
    message = sagemaker.describe_training_job(TrainingJobName=job_name)['FailureReason']
    print('Training failed with the following error: {}'.format(message))
```

```
Training job current status: InProgress
Training job ended with status: Completed
```

Una volta che il training sarà completato, sarà ora di **deployare il modello che è stato trainato meglio.** Possiamo creare il modello utilizzando il risultato del training:

```
model_name="DEMO-full-image-classification-model" + time.strftime('-%Y-%m-%d-%H-%M-%S'
, time.gmtime())
print(model_name)
info = sage.describe_training_job(TrainingJobName=job_name)
model_data = info['ModelArtifacts']['S3ModelArtifacts']
print(model_data)

hosting_image = get_image_uri(boto3.Session().region_name, 'image-classification')

primary_container = {
    'Image': hosting_image,
    'ModelDataUrl': model_data,
}

create_model_response = sage.create_model(
    ModelName = model_name,
    ExecutionRoleArn = role,
    PrimaryContainer = primary_container)

print(create_model_response['ModelArn'])
```

A questo punto, procediamo **scegliendo la parte di dataset da utilizzare per il testing:** in questo caso la scelta è ricaduta su **bathtubs:**

```
batch_input = 's3://{}/image-classification-full-training/test/'.format(bucket)
test_images = '/tmp/images/008.bathtub'

!aws s3 cp $test_images $batch_input --recursive --quiet
```

Creiamo ora un **batch job** per mandare tutte le immagini di test al modello per classificarle.

```
timestamp = time.strftime('-%Y-%m-%d-%H-%M-%S', time.gmtime())
batch_job_name = "image-classification-model" + timestamp
request = \
{
    "TransformJobName": batch_job_name,
    "ModelName": model_name,
    "MaxConcurrentTransforms": 16,
    "MaxPayloadInMB": 6,
    "BatchStrategy": "SingleRecord",
    "TransformOutput": {
        "S3OutputPath": 's3://{}/{}/output'.format(bucket, batch_job_name)
    },
    "TransformInput": {
        "DataSource": {
            "S3DataSource": {
                "S3DataType": "S3Prefix",
                "S3Uri": batch_input
            }
        },
        "ContentType": "application/x-image",
        "SplitType": "None",
        "CompressionType": "None"
    },
    "TransformResources": {
            "InstanceType": "ml.p2.xlarge",
            "InstanceCount": 1
    }
}

sagemaker = boto3.client('sagemaker')
sagemaker.create_transform_job(**request)

print("Created Transform job with name: ", batch_job_name)

while(True):
    response = sagemaker.describe_transform_job(TransformJobName=batch_job_name)
    status = response['TransformJobStatus']
    if status == 'Completed':
        print("Transform job ended with status: " + status)
        break
    if status == 'Failed':
        message = response['FailureReason']
        print('Transform failed with the following error: {}'.format(message))
        raise Exception('Transform job failed')
    time.sleep(30)
```

La creazione di un batch job accenderà una istanza EC2 (del taglio prescelto) per procedere con la classificazione delle immagini. Una volta che il processo sarà concluso, potremo controllare il

risultato: se saranno stati utilizzati solo due epochs durante il training, pochissime immagini saranno classificate correttamente.

Quando saremo soddisfatti del risultato, potremo **procedere con la creazione di un endpoint e, successivamente, a deployare il modello:**

```
from time import gmtime, strftime
job_name_prefix = "test"

timestamp = time.strftime('-%Y-%m-%d-%H-%M-%S', time.gmtime())
endpoint_config_name = job_name_prefix + '-epc-' + timestamp
endpoint_config_response = sage.create_endpoint_config(
    EndpointConfigName = endpoint_config_name,
    ProductionVariants=[{
        'InstanceType':'ml.m4.xlarge',
        'InitialInstanceCount':1,
        'ModelName':model_name,
        'VariantName':'AllTraffic'}])

timestamp = time.strftime('-%Y-%m-%d-%H-%M-%S', time.gmtime())
endpoint_name = job_name_prefix + '-ep-' + timestamp
print('Endpoint name: {}'.format(endpoint_name))

endpoint_params = {
    'EndpointName': endpoint_name,
    'EndpointConfigName': endpoint_config_name,
}
endpoint_response = sagemaker.create_endpoint(**endpoint_params)
```

Una volta creato l'endpoint, possiamo inviargli le immagini **tramite HTTP utilizzando gli AWS SDKs,** ad esempio utilizzando Python boto3.

Il comando sarà:

```
runtime.invoke_endpoint(EndpointName=endpoint_name,
                                    ContentType='application/x-image',
                                    Body=payload)
```

Dove

```
payload
```

è l'immagine.

L'endpoint può essere utilizzato sia da applicazioni deployate su AWS – a condizione che il loro IAM Role permetta di invocare gli endpoints di AWS SageMaker (es. Lambda functions, Microservizi docker che funzionano su Fargate, applicazioni deployate su istanze EC2), sia da applicazioni on-premise o addirittura direttamente su Client a condizione che esse riescano ad autenticare le loro API calls utilizzando le credenziali AWS.

In questo articolo abbiamo visto insieme **come deployare una semplice applicazione di Machine Learning con AWS SageMaker.** Divertente vero? Il Machine Learning vi affascina? Allora restate sintonizzati: presto molti altri articoli! E se avete curiosità contattateci!

beSharp

Dal 2011 beSharp guida le aziende italiane sul Cloud. Dalla piccola impresa alla grande multinazionale, dal manifatturiero al terziario avanzato, aiutiamo le realtà più all'avanguardia a realizzare progetti innovativi in campo IT.

Get in touch

beSharp.it
proud2becloud@besharp.it