Microsoft* Azure* 上にセキュアな Kubeflow* パイプラインを構築

仮想マシン上でインテル® ソフトウェア・ガード・エクステンションズを活用してセキュアで高速化されたマシンラーニング・パイプラインをデプロイ

Kelli Belcher インテル コーポレーション AI ソフトウェア・ソリューション・エンジニア

多くのマシンラーニング・アプリケーションは、基礎となるコードとデータの機密性と整合性を確保する必要があります。これまで保存中 (ストレージ内) または転送中 (ネットワーク経由での転送) のデータの暗号化は注目されていましたが、最近まで使用中（メインメモリー内）のデータのセキュリティーは注目されていませんでした。インテル® ソフトウェア・ガード・エクステンションズ（インテル® SGX）（英語）は、アプリケーション・コードとデータを安全に処理および保存できる一連の手順を提供します。インテル® SGX は、CPU 内にトラステッド・エグゼキューション (信頼できる実行) 環境を作成することにより、コンテナからのユーザーレベルのコードが、エンクレーブと呼ばれるプライベートなメモリー領域を割り当てて、アプリケーション・コードを実行できるようにします。

Microsoft* Azure* Confidential Computing プラットフォームを使用すると、インテル® SGX が提供するセキュリティーと機密性を活用して、Windows* と Linux* の両方の仮想マシン（VM）をデプロイできます。このチュートリアルでは、Azure* Kubernetes* Services（AKS）クラスター上にインテル® SGX のノードをセットアップする方法を示します。次に、スケーラブルなマシンラーニング・パイプラインの構築とデプロイに使用できる Kubernetes* 向けのマシンラーニング・ツールキット、Kubeflow* をインストールします。最後に、XGBoost 向けインテル® オプティマイゼーション（英語）、インテル® oneAPI データ・アナリティクス・ライブラリー（インテル® oneDAL）、および scikit-learn 向けインテル® エクステンション（英語）を使用してモデルのトレーニングと推論を高速化する方法を説明します。

この最適化モジュールは、Amazon Web Services* (AWS*)、Microsoft* Azure*、Google Cloud Platform* (GCP*) などの主要なクラウド・プロバイダー上でインテルにより最適化された AI ソリューションの構築とデプロイを容易にするために設計された、クラウドネイティブのオープンソース・リファレンス・アーキテクチャーのセットである Microsoft* Azure* 向けインテル® クラウド・オプティマイゼーション・モジュール (英語) の一部です。各モジュール (リファレンス・アーキテクチャー) には、必要な手順とソースコードがすべて含まれています。

この最適化モジュールのクラウド・ソリューション・アーキテクチャーでは、インテル® SGX VM を中核コンポーネントとする AKS クラスターを使用します。Kubeflow* パイプラインがコンテナ化された Python* コンポーネントを構築できるように、Azure* Container Registry を AKS クラスターにアタッチします。これらの Azure* リソースは、Azure* リソースグループで管理されます。次に、Kubeflow* ソフトウェア・レイヤーを AKS クラスターにインストールします。Kubeflow* パイプラインを実行すると、各パイプライン Pod がインテル® SGX ノードに割り当てられます。クラウド・ソリューションのアーキテクチャーを**図 1** に示します。

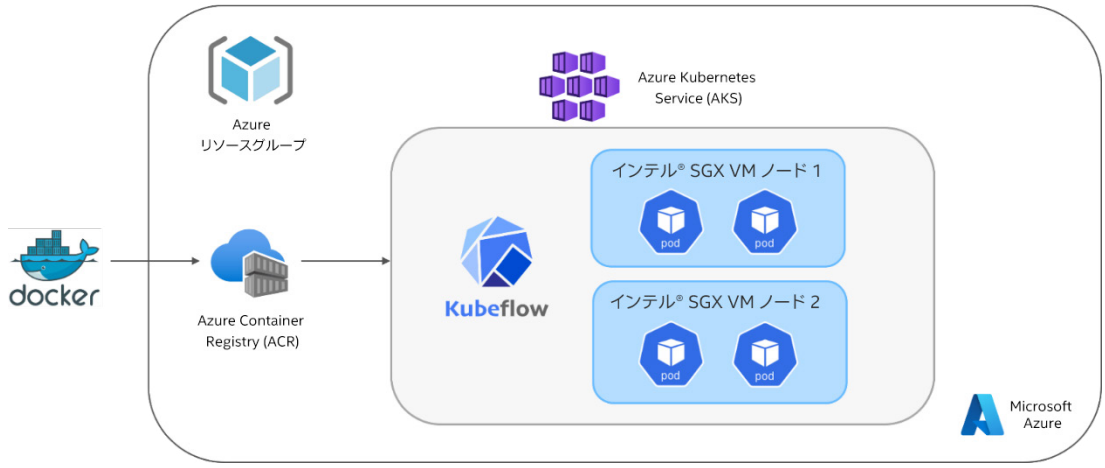

図 1. Kubeflow* パイプライン・アーキテクチャーの略図 (著者による画像)

準備

このチュートリアルを開始する前に、前提条件(英語)をダウンロードしてインストールしていることを確認してください。次に、新しいターミナルウィンドウで次のコマンドを実行し、Azure* コマンドライン・インターフェイスを使用して Microsoft* Azure* アカウントにログインします。

```
az login
```

次に、ソリューションの Azure* リソースを保持するリソースグループを作成します。リソースグループの名前を `intel-aks-kubeflow` にして、場所を `eastus` に設定します。

```
# Set the names of the Resource Group and Location
export RG=intel-aks-kubeflow
export LOC=eastus

# Create the Azure Resource Group
az group create -n $RG -l $LOC
```

Confidential Computing ノードを使用して AKS クラスターをセットアップするには、最初にシステム・ノード・プールを作成し、Confidential Computing アドオンを有効にします。Confidential Computing アドオンは、対象の各 VM ノードがインテル® SGX 向けの Azure* デバイスプラグイン Pod のコピーを実行するようにクラスターの DaemonSet を設定します。次のコマンドは、Dv5 シリーズ（第 3 世代インテル® Xeon® プロセッサー）の標準 VM を使用してノードプールをプロビジョニングします。これは、`CoreDNS` や `metrics-server` などの AKS システム Pod をホストするノードです。次のコマンドはさらに、クラスターのマネージド ID を有効にして、標準の Azure* Load Balancer をプロビジョニングします。

```
# Set the name of the AKS cluster
export AKS=aks-intel-sgx-kubeflow

# Create the AKS system node pool
az aks create --name $AKS \
--resource-group $RG \
--node-count 1 \
--node-vm-size Standard_D4_v5 \
--enable-addons confcom \
--enable-managed-identity \
--generate-ssh-keys -l $LOC \
--load-balancer-sku standard
```

すでに Azure* Container Registry をセットアップしている場合は、パラメーター `--attach-acr <registry-name>` を追加することでクラスターにアタッチできます。

システム・ノード・プールがデプロイされたら、インテル® SGX ノードプールをクラスターに追加します。次のコマンドは、Azure* DCSv3 シリーズから 2 つの 4 コアのインテル® SGX ノードをプロビジョニングします。キー `intelvm` と値 `sgx` を使用して、このノードプールにノードラベルを追加しています。このキー / 値のペアは、Kubernetes* `nodeSelector` で参照され、Kubeflow* パイプライン Pod をインテル® SGX ノードに割り当てます。

```
az aks nodepool add --name intelsgx \
--resource-group $RG \
--cluster-name $AKS \
--node-vm-size Standard_DC4s_v3 \
--node-count 2 \
--labels intelvm=sgx
```

Confidential ノードプールが設定されたら、次のコマンドを使用してクラスターのアクセス認証情報を取得し、ローカルの `.kube/config` ファイルにマージします。

```
az aks get-credentials -n $AKS -g $RG
```

次のコマンドでクラスターの認証情報が正しく設定されているかどうかを確認できます。AKS クラスターの名前が返されるはずです。

```
kubectl config current-context
```

インテル® SGX VM ノードが正常に作成されたことを確認するには、次のコマンドを実行します。

```
kubectl get nodes
```

`aks-intelsgx` という名前で始まる 2 つのエージェント・ノードが実行されていることが分かるはずです。DaemonSet が正常に作成されたことを確認するには、次のコマンドを実行します。

```
kubectl get pods -A
```

`kube-system` 名前空間で、`sgxplugin` という名前で始まる 2 つの Pod が実行されていることが分かるはずです。上記の Pod とノードプールが実行中であることを確認できたら、AKS クラスターで機密アプリケーションを実行する準備は完了です。

Kubeflow* のインストール

AKS クラスターに Kubeflow* をインストールするには、まず、Kubeflow* Manifests GitHub* リポジトリーをクローンします。

```
git clone https://github.com/kubeflow/manifests.git
```

カレント・ディレクトリーを、新しくクローンしたマニフェストのディレクトリーに変更します。

```
cd manifests
```

オプションのステップとして、次のコマンドを使用して、Kubeflow* ダッシュボードにアクセスするときに使用する（デフォルトの）パスワードを変更できます。

```
python3 -c 'from passlib.hash import bcrypt; import getpass;
print(bcrypt.using(rounds=12, ident="2y").hash(getpass.getpass()))'
```

`dex` ディレクトリーの `config-map.yaml` に移動して、新しいパスワードを設定ファイルの 22 行目付近のハッシュ値に設定します。

```
nano common/dex/base/config-map.yaml
```

```
    staticPasswords:
    - email: user@example.com
      hash:
```

次に、Istio* Ingress Gateway を `ClusterIP` から `LoadBalancer` に変更します。ブラウザーからダッシュボードにアクセスするために使用できる外部 IP アドレスが構成されます。`common/istio-1-16/istio-install/base/patches/service.yaml` に移動して、7 行目付近の `spec` の `type` を `LoadBalancer` に変更します。

```
apiVersion: v1
kind: Service
metadata:
  name: istio-ingressgateway
  namespace: istio-system
spec:
  type: LoadBalancer
```

AKS クラスターの場合は、Istio Webhook から AKS アドミッション・エンフォーサーを無効にする必要もあります。Istio の `install.yaml` に移動して、2694 行目付近に次のアノテーションを追加します。

```
nano common/istio-1-16/istio-install/base/install.yaml
```

```
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
metadata:
  name: istio-sidecar-injector
  annotations:
    admissions.enforcer/disabled: 'true'
  labels:
```

次に、Istio Gateway を更新して Transport Layer Security(TLS)プロトコルを構成します。HTTPS 経由でダッシュボードにアクセスできるようになります。`kf-istio-resources.yaml` に移動して、ファイルの最後、14 行目付近に次の内容を追加します。

```
nano common/istio-1-16/kubeflow-istio-resources/base/kf-istio-resources.yaml
```

```
    tls:
      httpsRedirect: true
  - port:
      number: 443
      name: https
      protocol: HTTPS
    hosts:
    - "*"
    tls:
      mode: SIMPLE
      privateKey: /etc/istio/ingressgateway-certs/tls.key
      serverCertificate: /etc/istio/ingressgateway-certs/tls.crt
```

これで、Kubeflow* をインストールする準備ができました。ここでは、`kustomize` を使用して、単一コマンドでコンポーネントをインストールします。コンポーネントを個別にインストール（英語）することもできます。

```
while ! kustomize build example | awk '!/well-defined/' | kubectl apply -f -; do echo
"Retrying to apply resources"; sleep 10; done
```

すべてのコンポーネントのインストールが完了するまでしばらくかかります。一部のコンポーネントは最初の試行で失敗する可能性があります。これは、Kubernetes* と kubectl 固有の動作です（例えば、CRD の準備ができた後に CR を作成する必要があります）。解決策は、コマンドが成功するまで再実行することです。コンポーネントのインストールが完了したら、次のコマンドを実行してすべての Pod が実行されていることを確認します。

```
kubectl get pods -A
```

オプション : Kubeflow* の新しいパスワードを作成した場合は、`dex` Pod を再起動して、新しいパスワードを使用していることを確認します。

```
kubectl rollout restart deployment dex -n auth
```

最後に、Istio ロードバランサーの外部 IP アドレスを使用して、TLS プロトコルの自己署名証明書を作成します。外部 IP アドレスを取得するには、次のコマンドを実行します。

```
kubectl get svc -n istio-system
```

Istio 証明書を作成して、次の内容をコピーします。

```
nano certificate.yaml
```

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: istio-ingressgateway-certs
  namespace: istio-system
spec:
  secretName: istio-ingressgateway-certs
  ipAddresses:
    - <Istio IP address>
  isCA: true
  issuerRef:
    name: kubeflow-self-signing-issuer
    kind: ClusterIssuer
    group: cert-manager.io
```

次に、証明書を適用します。

```
kubectl apply -f certificate.yaml
```

証明書が正常に作成されたことを確認します。

```
kubectl get certificate -n istio-system
```

これで、Kubeflow* ダッシュボードを起動する準備ができました。ダッシュボードにログインするには、Istio の IP アドレスをブラウザーに入力します。ダッシュボードに初めてアクセスした場合、自己署名証明書を使用しているために接続に関する警告が表示されることがあります。SSL CA 証明書がある場合は、その証明書に置き換えることもできます。自己署名証明書を使用する場合は、画面の表示に従います。dex のログイン画面が表示されます。ユーザー名とパスワードを入力します。Kubeflow* のデフォルトのユーザー名は `user@example.com`、デフォルトのパスワードは `12341234` です。

Kubeflow* ダッシュボードにログインしたら、Kubeflow* パイプラインをデプロイする準備は完了です。このモジュールの Kubeflow* パイプラインは、インテルと Accenture が協力して開発したローン不履行リスク予測 AI リファレンス・キット（英語）から派生したものです。コードはリファクタリングにより強化され、モジュール性と Kubeflow* パイプラインへの適合性が向上しました。このパイプラインは、借り手のローン不履行のリスクを予測する XGBoost モデルを構築し、XGBoost 向けインテル® オプティマイゼーション（英語）とインテル® oneDAL を使用してモデルのトレーニングと推論を高速化します。パイプライン全体のグラフを**図 2** に示します。

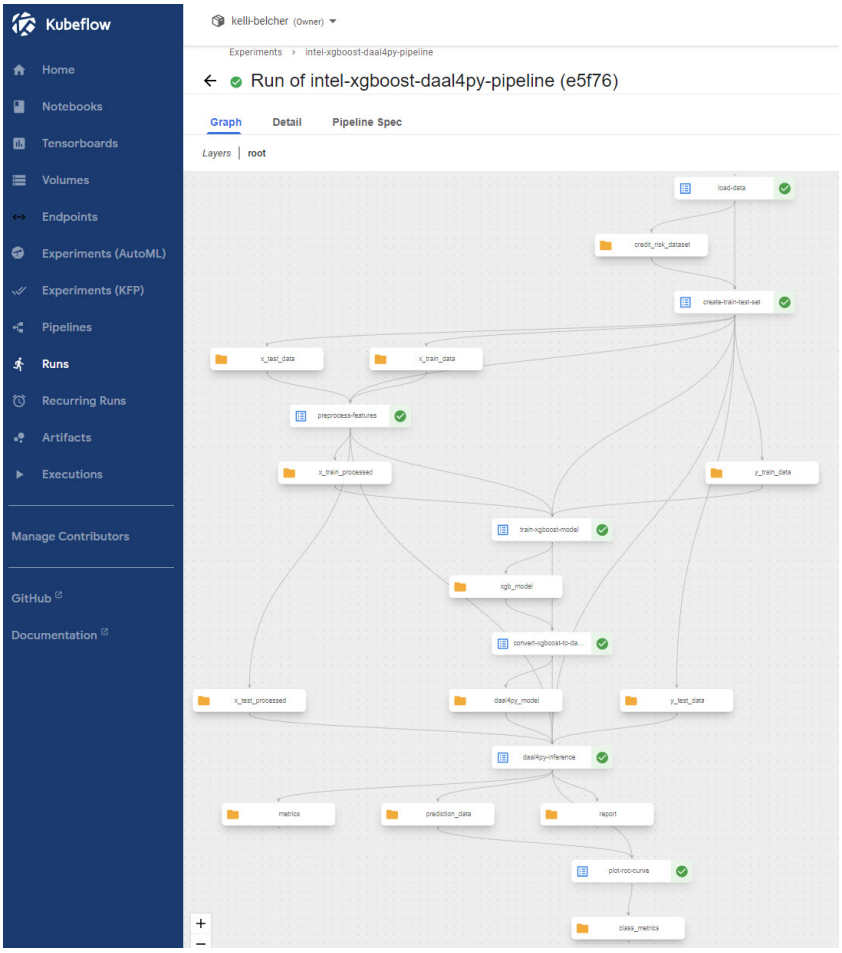

図 2. XGBoost Kubeflow* パイプラインのグラフ（著者による画像）

Kubeflow* パイプラインは、次の 7 つのコンポーネントで構成されます。

- **データのロード：**パイプライン実行パラメーターで指定された URL からデータセット（`credit_risk_dataset.csv`）をロードし、合成データの拡張を実行します。Kaggle の Credit Risk データセットをダウンロード（英語）します。
- **トレーニング・セットとテストセットの作成：**モデル評価のために、データを約 75:25 の比率でトレーニング・セットとテストセットに分割します。
- **特徴の前処理：**one-hot エンコーディングを使用してトレーニング・セットとテストセットのカテゴリー特徴を変換し、欠損値のインピュテーションを行い、数値特徴をべき乗変換します。
- **XGBoost モデルのトレーニング：**このコンポーネントは、XGBoost 向けインテル® オプティマイゼーションで提供されるアクセラレーションを活用してモデルをトレーニングします。
- **XGBoost モデルの daal4py への変換：**XGBoost モデルを推論に最適化された daal4py 分類器に変換します。
- **daal4py 推論：**推論に最適化された daal4py 分類器を使用して予測を計算し、モデルのパフォーマンスを評価します。各クラスの適合率（精度）、再現率、F1 スコア、モデルの曲線下面積（AUC）および精度スコアの出力概要が返されます。
- **受信者動作特性（ROC）曲線のプロット：**テストデータに対してモデル検証を実行し、ROC 曲線のプロットを生成します。

パイプラインのコードは、GitHub* リポジトリーの src（英語）フォルダー内にある `intel-xgboost-daal4py-pipeline-azure.py` です。Python* スクリプトを実行する前に、Kubeflow* パイプライン SDK のバージョン 2.0 以降がインストールされていることを確認します。SDK のバージョンを更新するには、次のコマンドを使用します。

```
pip install -U kfp
```

各パイプライン Pod がインテル® SGX ノードに確実に割り当てられるように、Kubernetes* の `nodeSelector` を使用します。`nodeSelector` は、Pod をスケジュールするときに、ラベルのキーと値のペアが一致するノードを検索します。次のコードは、インテル® SGX ノードプールに追加したノードラベルを使用して、データ前処理パイプライン・タスク、`preprocess_features_op` をインテル® SGX ノードに割り当てます。

```
from kfp import kubernetes

kubernetes.add_node_selector(task = preprocess_features_op,
      label_key = 'intelvm', label_value = 'sgx')
```

パイプラインのコンポーネント

パイプラインの最初のコンポーネント、`load_data` は、パイプライン実行パラメーターで指定された URL から Credit Risk データセットをダウンロードし、指定されたサイズまでデータを合成的に拡張します。新しいデータセットは、Kubeflow* MinIO ボリュームに出力アーティファクトとして保存されます。このデータセットは、コンポーネントの次のステップ、`create_train_test_set` に読み込まれます。このコンポーネントは、モデル評価のためにデータを約 75:25 に分割します。トレーニング・セットとテストセット、`X_train`、`y_train`、`X_test`、および `y_test` は、出力データセット・アーティファクトとして保存されます。

`preprocess_features` コンポーネントでは、XGBoost モデル向けのデータを準備するため、データ前処理パイプラインを作成します。このコンポーネントは、MinIO ストレージから `X_train` と `X_test` ファイルをロードし、one-hot エンコーディングを使用してカテゴリー特徴を変換し、欠損値のインピュテーションを行い、数値特徴をべき乗変換します。

次のコンポーネントは、XGBoost モデルをトレーニングします。インテルの CPU で XGBoost を使用すると、コードを変更することなく、oneAPI のソフトウェア・アクセラレーションを活用できます。サイズ 100 万の増分トレーニング更新の初期テストで、XGBoost 向けインテル® オプティマイゼーション v1.4.2 は、XGBoost v0.81 と比較して最大 1.54 倍のスピードアップを達成しました（ローン不履行リスク予測リファレンス・キットのパフォーマンス結果（英語）は、GitHub* で確認できます）。次のコードが `train_xgboost_model` コンポーネントに実装されています。

```
# Create the XGBoost DMatrix, which is optimized for memory efficiency and training speed
dtrain = xgb.DMatrix(X_train.values, y_train.values)

# Define model parameters
params = {"objective": "binary:logistic",
     "eval_metric": "logloss",
     "nthread": 4,   # num_cpu
     "tree_method": "hist",
     "learning_rate": 0.02,
     "max_depth": 10,
     "min_child_weight": 6,
     "n_jobs": 4,   # num_cpu
     "verbosity": 1}

# Train initial XGBoost model
clf = xgb.train(params = params, dtrain = dtrain, num_boost_round = 500)
```

モデルの予測速度をさらに最適化するため、トレーニング済みの XGBoost モデルを、`convert_xgboost_to_daal4py` コンポーネントで推論に最適化された daal4py 分類器に変換します。daal4py は oneDAL の Python* API です。サイズ 100 万のバッチ推論のテストで、oneDAL は最大 4.44 倍のスピードアップを達成しました。XGBoost モデルの daal4py への変換は、わずか 1 行のコードで済みます。

```
# Convert XGBoost model to daal4py
daal_model = d4p.get_gbt_model_from_xgboost(clf)
```

次に、`daal4py_inference` コンポーネントで、daal4py モデルを使用してテストセットでのモデルのパフォーマンスを評価します。

```
# Compute both class labels and probabilities
daal_prediction = d4p.gbt_classification_prediction(
     nClasses = 2,
     resultsToEvaluate = "computeClassLabels|computeClassProbabilities"
).compute(X_test, daal_model)
```

`gbt_classification_prediction` メソッドを呼び出して、バイナリークラスのラベルと確率の両方を計算します（上記のコードを参照）。この関数を使用して対数確率を計算することもできます。このコンポーネントは、CSV 形式の分類レポートと、2 つの Kubeflow* メトリクス・アーティファクト（曲線下面積とモデルの精度）を返します。これらのアーティファクトは、`metrics` アーティファクトの **[Visualization]** タブで確認できます。

パイプラインの最後のコンポーネント、`plot_roc_curve` は、確率と `true` クラスラベルを含む予測データをロードし、scikit-learn 向けインテル® エクステンション（英語）の CPU 高速化バージョンを使用して ROC 曲線を計算します。パイプラインの実行が終了すると、結果が Kubeflow* ClassificationMetric アーティファクトとして保存されます。このアーティファクトは、`roc_curve_daal4py` アーティファクトの **[Visualization]** タブで確認できます（**図 3**）。

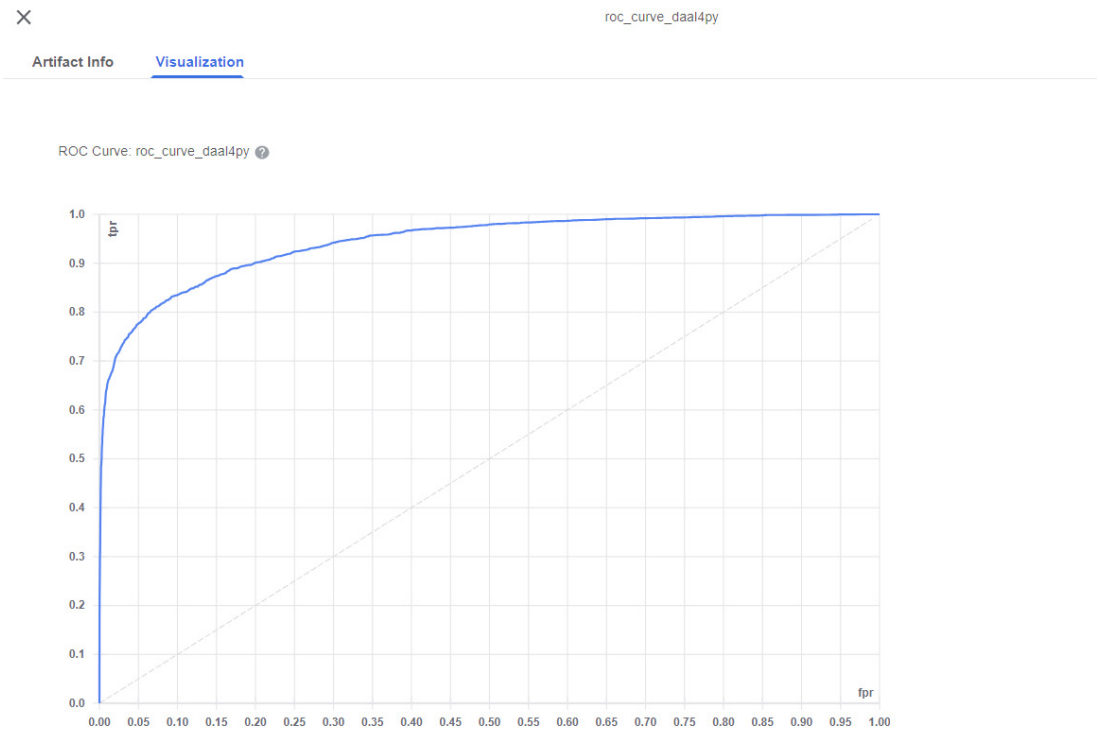

図 3. 分類器の ROC 曲線

次のステップ

完全なソースコード（英語）は、GitHub* にあります。実装のサポートが必要な場合は、こちらのサイト（英語）からお問い合わせください。インテル® クラウド・オプティマイゼーション・モジュール（英語）で詳細を確認したり、インテル® DevHub Discord（英語）サーバーでほかの開発者とチャットで交流することができます。