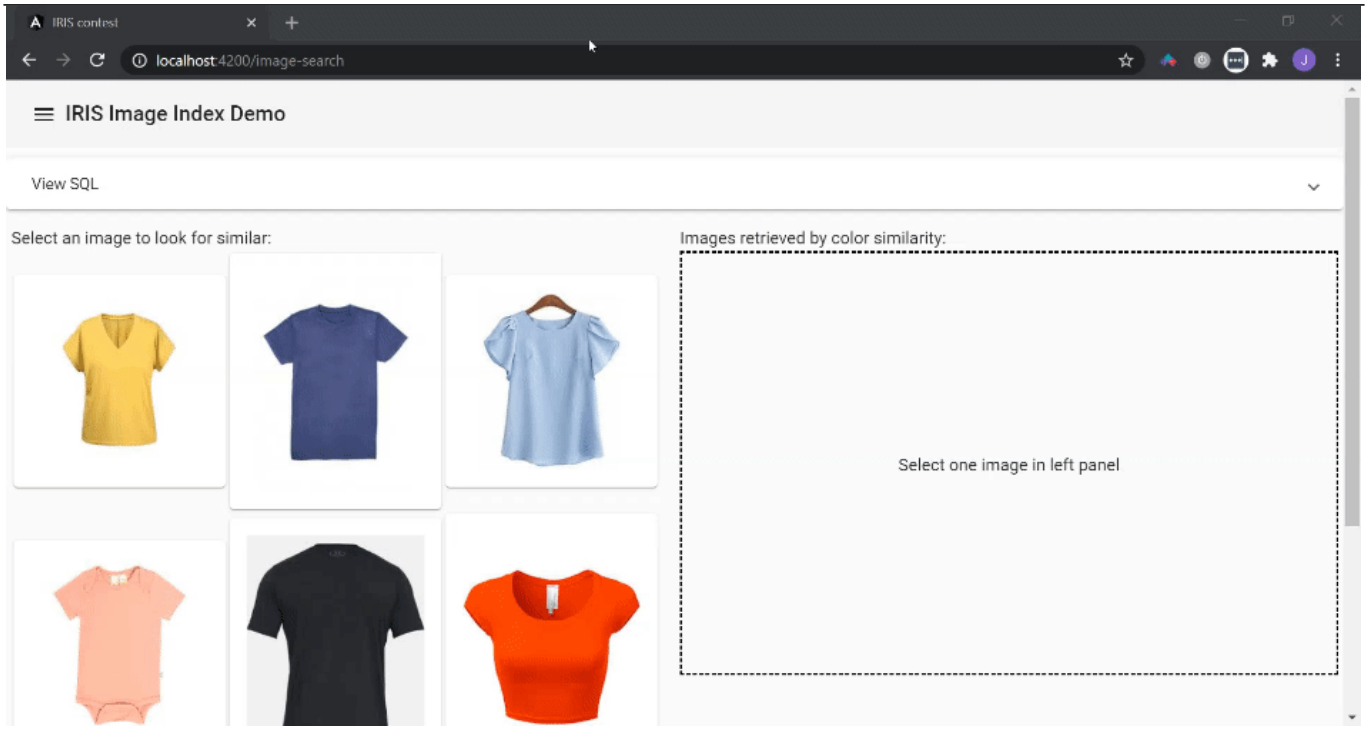
Article
Lorenzo Scalese · Mai 18, 2022 14m de lecture

Open Exchange

Personalisation des index SQL avec des fonctions Python

La recherche d'images comme celle de Google est une fonctionnalité intéressante qui m'émerveille - comme presque tout ce qui est lié au traitement des images.

Il y a quelques mois, InterSystems a publié un aperçu de Python Embedded. Comme Python dispose de nombreuses librairies pour le traitement d'images, j'ai décidé de lancer ma propre tentative pour jouer avec une sorte de recherche d'images - une version beaucoup plus modeste en fait :-)



Un peu de théorie

Pour réaliser un système de recherche d'images, il faut d'abord sélectionner un ensemble de caractéristiques à extraire des images - ces caractéristiques sont également appelées descripteurs. L'étendue de chaque composante de ces descripteurs crée ce qu'on appelle un espace de caractéristiques, et chaque instance de cet espace est appelée un vecteur. Le nombre d de composantes nécessaires pour décrire les vecteurs, définit la dimension de l'espace des caractéristiques et des vecteurs, appelé d-dimensionnel.

Figure 1 - Un espace caractéristique tridimensionnel et un vecteur descripteur dans cet espace.
Credits: https://tinyurl.com/ddd76dln

Une fois l'ensemble des descripteurs définis, tout ce que vous avez à faire pour rechercher une image dans la base de données est d'extraire les mêmes descripteurs d'une image à rechercher et de les comparer aux descripteurs des images de la base de données - qui ont été précédemment extraits.

Dans ce travail, on a simplement utilisé la couleur dominante de l'image comme descripteur (j'ai dit que c'était une version modeste...). Comme une représentation RVB des couleurs a été utilisée, l'espace caractéristique est un

espace tridimensionnel - 3d en abrégé. Chaque vecteur dans un tel espace a 3 composantes - (r,g,b), dans la gamme [0, 255].

Figure 2 - L'espace tridimensionnel des caractéristiques RVB
Credits: https://www.baslerweb.com/fp-1485687434/media/editorial/contentimages/f...

En traitement du signal, il est très fréquent d'avoir des espaces à n dimensions avec des valeurs de n bien supérieures à 3. En fait, vous pouvez combiner un grand nombre de descripteurs dans un même vecteur afin d'obtenir une meilleure précision. C'est ce qu'on appelle la sélection de caractéristiques et c'est une étape très importante dans les tâches de classification/reconnaissance.

Il est également courant de normaliser la plage de dimensions en [0, 1], mais pour des raisons de simplicité, ce travail utilise la plage par défaut [0, 255].

L'avantage de modéliser des caractéristiques sous forme de vecteurs est la possibilité de les comparer à travers des métriques de distance. Il existe de nombreuses distances, chacune ayant ses avantages et ses inconvénients, selon que l'on recherche la performance ou la précision. Dans ce travail, j'ai choisi des distances faciles à calculer - manhattan et chebyshev, qui sont essentiellement des différences absolues avec une précision raisonnable.

Figure 3 - Représentation de certains paramètres de distance
Credits: https://i0.wp.com/dataaspirant.com/wp-content/uploads/2015/04/coverpost...

Index fonctionnel

Mais il ne s'agit que des outils nécessaires pour comparer les images en fonction de leur contenu. Si vous ne disposez pas d'un langage de requête comme SQL, vous vous retrouverez avec des méthodes et des paramètres de recherche fastidieux... De plus, en utilisant SQL, vous pouvez combiner cet index avec d'autres opérateurs bien connus, créant ainsi des requêtes complexes.

C'est ici où Functional Index d'InterSystems est très utile.

Un index fonctionnel est une classe qui implémente la classe abstraite %Library.FunctionalIndex qui implémente certaines méthodes afin de gérer la tâche d'indexation dans une instruction SQL. Ces méthodes traitent essentiellement les insertions, les suppressions et les mises à jour.

```
/// Indexation fonctionnelle permettant d'optimiser les requêtes sur les données d'im
age
Class dc.multimodel.ImageIndex.Index Extends %Library.FunctionalIndex [ System = 3 ]
{

/// Cardinalité de l'espace des caractéristiques
/// Comme cette classe est destinée à indexer l'image dans l'espace RVB, sa cardinali
té est de 3
Paramètre Cardinalité = 3 ;

/// Cette méthode est invoquée lorsqu'une instance existante d'une classe est supprim
ée.
ClassMethod DeleteIndex(pID As %CacheString, pArg... As %Binary) [ CodeMode = generat
or, ServerOnly = 1 ]
{
    If (%mode '= "method") {
        $$$GENERATE("Set indexer = ##class(dc.multimodel.ImageIndex.Indexer).GetInsta
nce("""_%class_""", """_%property_""")")
        $$$GENERATE("Set indexer.Cardinality = "_..#Cardinality)
        $$$GENERATE("Do indexer.Delete(pID, pArg...)")
    }
    Return $$$OK
}
```

```
ClassMethod Find(pSearch As %Binary) As %Library.Binary [ CodeMode = generator, Serve
rOnly = 1, SqlProc ]
{
    If (%mode '= "method") {
        $$$GENERATE("Set result = """")
        $$$GENERATE("Set result = ##class(dc.multimodel.ImageIndex.SQLFind).%New()")
        $$$GENERATE("Set indexer = ##class(dc.multimodel.ImageIndex.Indexer).GetInsta
nce(""_%class_""", ""_%property_""")")
        $$$GENERATE("Set indexer.Cardinality = "_..#Cardinality)
        $$$GENERATE("Set result.Indexer = indexer")
        $$$GENERATE("Do result.PrepareFind(pSearch)")
        $$$GENERATE("Return result")
    }
    Return $$$OK
}

/// Cette méthode est invoquée lorsqu'une nouvelle instance d'une classe est insérée
dans la base de données.
ClassMethod InsertIndex(pID As %CacheString, pArg... As %Binary) [ CodeMode = generat
or, ServerOnly = 1 ]
{
    If (%mode '= "method") {
        $$$GENERATE("Set indexer = ##class(dc.multimodel.ImageIndex.Indexer).GetInsta
nce(""_%class_""", ""_%property_""")")
        $$$GENERATE("Set indexer.Cardinality = "_..#Cardinality)
        $$$GENERATE("Do indexer.Insert(pID, pArg...)")
    }
    Return $$$OK
}

ClassMethod PurgeIndex() [ CodeMode = generator, ServerOnly = 1 ]
{
    If (%mode '= "method") {
        $$$GENERATE("Set indexer = ##class(dc.multimodel.ImageIndex.Indexer).GetInsta
nce(""_%class_""", ""_%property_""")")
        $$$GENERATE("Set indexer.Cardinality = "_..#Cardinality)
        $$$GENERATE("Set indexGbl = indexer.GetIndexLocation()")
        $$$GENERATE("Do indexer.Purge()")
    }
    Return $$$OK
}

/// Cette méthode est invoquée lorsqu'une instance existante d'une classe est mise à
jour.
ClassMethod UpdateIndex(pID As %CacheString, pArg... As %Binary) [ CodeMode = generat
or, ServerOnly = 1 ]
{
    If (%mode '= "method") {
        $$$GENERATE("Set indexer = ##class(dc.multimodel.ImageIndex.Indexer).GetInsta
nce(""_%class_""", ""_%property_""")")
        $$$GENERATE("Set indexer.Cardinality = "_..#Cardinality)
        $$$GENERATE("Do indexer.Update(pID, pArg...)")
    }
    Return $$$OK
}

}
```

J'ai caché une partie du code d'implémentation pour des raisons de lisibilité ; vous pouvez consulter le code dans le lien OpenExchange.

Une autre classe abstraite doit être implémentée, c'est %SQL.AbstractFind, afin de rendre disponible l'utilisation de l'opérateur %FIND pour demander au moteur SQL d'utiliser votre index personnalisé.

Une explication beaucoup plus détaillée et conviviale des index fonctionnels est donnée par @alexander-koblov qui constitue également un excellent exemple d'index fonctionnel. Je vous recommande vivement de le lire.

Si vous souhaitez aller plus loin, vous pouvez jouer avec le code source des index %iFind et %UIMA index.

Dans ce travail, j'ai configuré une classe de test de persistance simple, où le chemin des images est stocké, et un index personnalisé pour la recherche d'images est défini pour ce champ.

```
Class dc.multimodel.ImageIndex.Test Extends %Persistent
{

Property Name As %String;

Property ImageFile As %String(MAXLEN = 1024);

Index idxName On Name [ Type = bitmap ];

Index idxImageFile On (ImageFile) As dc.multimodel.ImageIndex.Index;
```

Notez que idxImageFile est un index personnalisé (dc.multimodel.ImageIndex.Index) pour le champ Image (qui stocke le chemin de l'image).

Le tour de Python (et COS) !

Ainsi, les classes abstraites d'index fonctionnel vous donneront les points d'entrée où vous pourrez effectuer l'extraction de caractéristiques et la recherche lors de l'exécution des instructions SQL. Maintenant, c'est au tour de Python !

Vous pouvez importer et exécuter le code Python dans un contexte COS en utilisant Python intégré. Par exemple, pour extraire la couleur dominante d'une image :

```
Method GetDominantColorRGB(pFile As %String, ByRef pVector) As %Status
{
  Set sc = $$$OK
  Try {
    Set json = ##class(%SYS.Python).Import("json")
    Set fastcolorthief = ##class(%SYS.Python).Import("fast_colorthief")
    Set imagepath = pFile
    Set dominantcolor = fastcolorthief."get_dominant_color"(imagepath, 1)
    Set vector = {}.%FromJSON(json.dumps(dominantcolor))
    Set n = ..Cardinality - 1
    For i = 0:1:n {
      Set pVector(i) = vector.%Get(i)
    }
  } Catch(e) {
    Set sc = e.AsStatus()
  }
  Return sc
}
```

Dans cette méthode, deux librairies Python sont importées (json et fastcolorthief). La librairie fastcolorthief renvoie une représentation Python de type tableau 3-d avec les valeurs de RGB ; l'autre librairie - json, sérialise ce tableau dans un %DynamicArray.

La couleur dominante est extraite pour chaque enregistrement qui est inséré ou mis à jour - une fois que l'index fonctionnel lève les appels aux méthodes InsertIndex et UpdateIndex en réponse aux insertions et mises à jour dans le tableau. Ces caractéristiques sont stockées dans l'index global du tableau :

```
Method Insert(pID As %CacheString, pArgs... As %Binary)
{
    // pArgs(1) has the image path
    $$$ThrowOnError(..GetDominantColor(pArgs(1), .rgb))
    Set idxGbl = ..GetIndexLocation()
    Set @idxGbl@("model", pID) = ""
    Merge @idxGbl@("model", pID, "rgb") = rgb
    Set @idxGbl@("last-modification") = $ZTIMESTAMP
}

Method Update(pID As %CacheString, pArg... As %Binary)
{
    // pArgs(1) has the image path
    Set idxGbl = ..GetIndexLocation()
    Do ..GetDominantColor(pArg(1), .rgb)
    Kill @idxGbl@("model", pID)
    Set @idxGbl@("model", pID) = ""
    Merge @idxGbl@("model", pID, "rgb") = rgb
    Set @idxGbl@("last-modification") = $ZTIMESTAMP
}
```

De la même manière, lorsque des enregistrements sont supprimés, l'index fonctionnel lance des appels aux méthodes DeleteIndex et PurgeIndex. À leur tour, les fonctionnalités doivent être supprimées de l'index global du tableau :

```
Method Delete(pID As %CacheString, pArg... As %Binary)
{
    Set idxGbl = ..GetIndexLocation()
    Kill @idxGbl@("model", pID)
    Set @idxGbl@("last-modification") = $ZTIMESTAMP
}

Method Purge(pID As %CacheString, pArg... As %Binary)
{
    Set idxGbl = ..GetIndexLocation()
    Kill @idxGbl
    Set @idxGbl@("last-modification") = $ZTIMESTAMP
}
```

L'index global est récupéré par introspection dans la classe persistante :

```
Method GetIndexLocation() As %String
{
    Set storage = ##class(%Dictionary.ClassDefinition).%OpenId(..ClassName).Storages.
GetAt(1).IndexLocation
    Return $NAME(@storage@(..IndexName))
```

```
}
```

Lorsque les utilisateurs utilisent l'index dans les clauses WHERE, la méthode Find() est activée par l'index de la fonction. Les instructions de la requête sont transmises afin que vous puissiez les analyser et décider de ce qu'il faut faire. Dans ce travail, les paramètres sont sérialisés en JSON afin de faciliter leur analyse. Les paramètres de la requête ont la structure suivante :

```
SELECT ImageFile
FROM dc_multimodel_ImageIndex.Test
WHERE ID %FIND search_index(idxImageFile, '{"color_similarity":{"image":"/data/img/te
st/161074693598711.jpg","first":5,"strategy":"knn"}}')
```

Dans cette instruction, vous pouvez voir l'utilisation de l'opérateur %FIND et de la fonction searchindex. C'est ainsi que SQL accède à notre index personnalisé.

Les paramètres de searchindex définissent l'index à rechercher - idxImageFile, dans ce cas ; et la valeur à envoyer à l'index. Ici, l'index attend un objet JSON, avec une configuration d'objet définissant : (i) le chemin de l'image, (ii) une limite pour les résultats, et (iii) une stratégie de recherche.

Une stratégie de recherche est simplement l'algorithme à utiliser pour effectuer la recherche. Actuellement, deux stratégies sont mises en œuvre : (i) fullscan et (ii) knn, qui correspond à k-proches voisins.

La stratégie fullscan consiste simplement en une recherche exhaustive mesurant la distance entre l'image recherchée et chaque image stockée dans la base de données.

```
Method FullScanFindStrategy(ByRef pSearchVector, ByRef pResult) As %Status
{
    Set sc = $$$OK
    Try {
        Set idxGbl = ..Indexer.GetIndexLocation()
        Set rankGbl = ..Indexer.GetRankLocation()

        Set id = $ORDER(@idxGbl@("model", ""))
        While (id '= "") {
            If ($ISVALIDNUM(id)) {
                Merge vector = @idxGbl@("model", id, "rgb")
                Set distance = ..Indexer.GetL1Distance(.pSearchVector, .vector)
                Set result(distance, id) = ""
            }
            Set id = $ORDER(@idxGbl@("model", id))
        }

        Kill @rankGbl@(..ImagePath, ..FindStrategy)
        If (..First '= "") {
            Set c = 0
            Set distance = $ORDER(result(""))
            While (distance '= "") && (c < ..First) {
                Merge resultTmp(distance) = result(distance)

                Set id = $ORDER(result(distance, ""))
                While (id '= "") {
                    Set @rankGbl@(..ImagePath, ..FindStrategy, id) = distance
                    Set id = $ORDER(result(distance, id))
                }
```

```
                Set c = c + 1
                Set distance = $ORDER(result(distance))
            }
            Kill result
            Merge result = resultTmp
        }

        Merge pResult = result
    }
    Catch ex {
        Set sc = ex.AsStatus()
    }
    Return sc
}
```

La stratégie KNN utilise une approche plus sophistiquée. Elle utilise une librairie Python pour créer une structure arborescente appelée Ball Tree. Une telle arborescence convient à une recherche efficace dans un espace à n dimensions.

```
Method KNNFindStrategy(ByRef pSearchVector, ByRef pResult) As %Status
{
    Do ..Log(" ------ KNNFindStrategy ------ ")
    Set sc = $$$OK
    Try {
        Set idxGbl = ..Indexer.GetIndexLocation()
        Set rankGbl = ..Indexer.GetRankLocation()

        Set json = ##class(%SYS.Python).Import("json")
        Set knn = ##class(%SYS.Python).Import("knn")

        Set first = ..First
        Set k = $GET(first, 5)

        Set n = ..Indexer.Cardinality - 1
        Set x = ""
        For i = 0:1:n {
            Set $LIST(x, * + 1) = pSearchVector(i)
        }
        Set x = "[["_$LISTTOSTRING(x, ",")_"]]"

        $$$ThrowOnError(..CreateOrUpdateKNNIndex())
        Set ind = knn.query(x, k, idxGbl)
        Set ind = {}.%FromJSON(json.dumps(ind.tolist()))
        Set ind = ind.%Get(0)

        Kill result
        Kill @rankGbl@(..ImagePath, ..FindStrategy)
        Set n = k - 1
        For i=0:1:n {
            Set id = ind.%Get(i)
            Set result(i, id) = ""
            Set @rankGbl@(..ImagePath, ..FindStrategy, id) = i
        }
        Merge pResult = result
    }
    Catch ex {
        Set sc = ex.AsStatus()
```

```
    }
    Return sc
}
```

Le code Python pour générer une arborescence Ball Tree est présenté ci-dessous :

```
from sklearn.neighbors import BallTree
import numpy as np
import pickle
import base64
import irisnative

def get_iris():
  ip = "127.0.0.1"
  port = 1972
  namespace = "USER"
  username = "superuser"
  password = "SYS"

  connection = irisnative.createConnection(ip,port,namespace,username,password)
  dbnative = irisnative.createIris(connection)

  return (connection, dbnative)

def release_iris(connection):
  connection.close()

def normalize_filename(filename):
  filename = filename.encode('UTF-8')
  return base64.urlsafe_b64encode(filename).decode('UTF-8')

def create_index(index_global, cardinality):
  connection, dbnative = get_iris()
  X = get_data(dbnative, index_global, cardinality)
  tree = BallTree(X, metric = "chebyshev")
  filename = f"/tmp/${normalize_filename(index_global)}.p"
  pickle.dump(tree, open(filename, "wb"))
  release_iris(connection)
  return tree

def get_data(dbnative, index_global, cardinality):
  X = []
  iter_ = dbnative.iterator(index_global, "model")
  for subscript, value in iter_.items():
    id_ = subscript
    v = []
    for i in range(cardinality):
      v.append(
        dbnative.get(index_global, "model", id_, "rgb", i) / 255
      )
    X.append(v)
  return X

def query(x, k, index_global):
  filename = f"/tmp/${normalize_filename(index_global)}.p"
  tree = pickle.load(open(filename, "rb"))
  x = eval(x)
```

```
  x_ = [xi / 255 for xi in x[0]]
  dist, ind = tree.query([x_], k)
  return ind
```

Lorsqu'une image est recherchée, l'index personnalisé appelle la méthode de requête de l'objet Ball Tree en Python. Vous pouvez également noter l'utilisation de l'API native d'IRIS afin d'accéder aux valeurs RVB globales de l'index pour la construction de l'arborescence Ball Tree.

Pour ordonner les images par similarité, il a été développé une procédure SQL qui traverse une globale stockant les distances précédemment calculées pour chaque image recherchée :

```
Method DiffRank(pSearch As %Binary, pId As %String) As %Float
{
    Set search = {}.%FromJSON(pSearch)
    If (search.%IsDefined("color_similarity")) {
        Set config = search.%Get("color_similarity")
        Set imagePath = config.%Get("image")
        If (config.%IsDefined("strategy")) {
            Set findStrategy = config.%Get("strategy")
        }
        Set rankGbl = ..Indexer.GetRankLocation()
        Set rank = $GET(@rankGbl@(imagePath, findStrategy, pId))
        Return rank
    }
    Return ""
}
```

Vous pouvez donc modifier l'instruction SQL pour classer les résultats par similarité :

```
SELECT ImageFile, dc_multimodel_ImageIndex.Test_idxImageFileDiffRank('{"color_similar
ity":{"image":"/data/img/test/161074693598711.jpg","first":5,"strategy":"knn"}}', id)
 AS DiffRank
FROM dc_multimodel_ImageIndex.Test
WHERE ID %FIND search_index(idxImageFile, '{"color_similarity":{"image":"/data/img/te
st/161074693598711.jpg","first":5,"strategy":"knn"}}')
ORDER BY DiffRank
```

Conclusion

L'objectif de ce travail était de montrer comment combiner la définition d'index fonctionnels dans COS avec des appels au code Python utilisant leurs étonnantes bibliothèques. De plus, en utilisant cette technique, vous pouvez accéder à des fonctionnalités complexes fournies par les librairies Python dans des instructions SQL, ce qui vous permet d'ajouter de nouvelles fonctionnalités à vos applications.

#Embedded Python #Indexation #Multi-model #SQL #InterSystems IRIS
Voir l'application sur InterSystems Open Exchange