Master 1 Informatique

Rapport de Projet : Neural Network And Learning

Sujet :

Construction d'un classificateur de photo de scènes naturelles

Yann MARTIN D'ESCRIENNE
Yohann TOGNETTI

« Année universitaire 2020 - 2021 »

Table des matières

1 Présentation générale du projet **2**

2 Descriptif du sujet et ses finalités **2**
2.1 Descriptif du sujet 2
2.2 Les catégories 2

3 Jeux de données **2**
3.1 Structure des jeux de données 2
3.2 Modification des jeux de données 3

4 Implémentation de l'algorithme **4**
4.1 preprocessing 4
4.2 PreProcessing.py 5
4.3 Classify.py 5
4.4 PlotResult.py 6

5 Résultat obtenus **7**
5.1 Résultat général 7
5.2 Résultat pour la valeur "1" 8
5.3 Résultat par entités 9
5.4 Résultat de la polarité 10

6 Les améliorations possibles **11**
6.1 Les classe 11
6.2 Le jeu de données 11
6.3 Les marques et modèles 11
6.4 La polarité 11

7 Conclusion **12**

8 Références **12**

1 Présentation générale du projet

Dans le cadre de notre cours de Neural Network and Learning, il nous a été demandé d'effectuer un projet impliquant une intelligence artificielle travaillant sur des images. Celle-ci a pour but de construire un classificateur de photo de scènes naturelles comme ceux qu'on pourrait intégrer dans un appareil photo intelligent qui à chaque prise de photo va taguer l'image avec le nom d'une catégorie qui y correspond.

2 Descriptif du sujet et ses finalités

2.1 Descriptif du sujet

Ainsi, pour entrer dans les détails le but qui est de "taguer l'image" correspond au fait de la catégoriser grâce à un classificateur entrainé possédant les catégories de scènes naturelles désirées. Pour cela avons dû utiliser un réseau neuronal convolutif étant donné que nous traitons des images. Une fois la réponse du classificateur reçue, elle correspond initialement à un entier, nous avions donc le devoir de le convertir en tag correspondant au nom de la catégorie trouvée.

L'affichage du résultat doit se faire à l'aide d'une méthode *predict* qui prend en entrée le chemin de fichier d'une image et une chaine de caractères 'mode' qui renvoie le nom de la catégorie correspondant à l'image si **mode='category'** où un vecteur de probabilités si **mode='probabilities'** ou chaque éléments du vecteur indique la probabilité que l'image appartienne à la catégorie correspondante.

2.2 Les catégories

Celles-ci sont au nombre de 6, chacune correspond à une catégorie de paysage ou de scène naturelle de la liste suivante :

- buildings
- forest
- glacier
- mountain
- sea
- street

Building correspond ainsi à des bâtiments comme une maison ou un immeuble qui se trouve généralement dans une ville.
Forest comme son nom l'indique correspond à un paysage forestier avec la présence ou non d'animaux.
Glacier quant à lui représente une montagne enneigée ou bien un glacier ou morceaux de glace de la banquise, cette catégorie est extrêmement corrélée avec *Moutain* car celle-ci correspond à une montagne pouvant parfois aussi être enneigé. Le découpage n'est pas toujours pertinent et encore plus ici, nous en parleront dans la section *Description des données*.
Sea représente les décors marins en général, que ce soit une plage, la mer avec ou sans terre ferme, ou bien même un décor sous-marin, tout cela va dans cette catégorie.
Enfin la catégorie *Street* met en avant une rue, qui peut donc facilement se rapprocher de *building* étant donnée la présence quasi-permanente de bâtiments dans une rue, on y trouve souvent des passant et des voitures.

3 Jeux de données

Les jeux de données se divisent en deux groupes : le jeu de données d'entrée correspondant à ce que l'IA va utiliser pour s'entrainer et celui de test sur lequel les mesures seront effectuées. Tout deux proviennent du site du SemEval.

3.1 Structure des jeux de données

Ce sont tout deux des documents XML possédant la structure suivante (simplifiée) :

FIGURE 1 – structure du fichier XML

3.2 Modification des jeux de données

Suite à notre implémentation de notre IA qui sera décrite dans le chapitre suivant, de nombreux problèmes nous ont forcés à ajouter nous même des phrases dans le jeu de données d'entrainement.

En effet notre IA répond de la présence ou non de chaque couple E#C pour chaque combinaison d'entité E et catégorie C. Cela se traduit par une nécessité de nombreuses données appartenant à chacun des couples pouvant apparaitre afin d'obtenir une fonction d'évaluation optimale. Dans le cas contraire le résultat est rarement celui attendu, il nous a donc fallu compléter le jeu de données.

Environ 350 phrases ont été ajoutées, toutes sont la section 'sentences' d'**ID 198**. Certaines proviennent d'avis de consommateur sur les ordinateur portables les plus commentés sur Amazon (en anglais), notamment pour les phrases portant sur les clavier, écran, pavé tactile et batterie, qui sont de manière générale beaucoup cité dans une revue de produit. Nous avons rédigés le reste pour les catégories plus complexe et moins abordées. Par exemple la qualité des lecteurs DVD ou bien l'ergonomie des OS. Cela s'éloigne d'un cas réel mais reste tout de même nécessaire au bon apprentissage de l'IA.

Remarque : Il est à noté que certain couple n'apparaissent ni dans les données d'entrainement, ni dans les données de test. Par simplification, ces couples ont été ignorés et aucunes phrases n'est étiquetées avec.

FIGURE 2 – dataframe du fichier XML

4 Implémentation de l'algorithme

Notre algorithme est développé en python. Il se trouve dans le fichier *Program.py* et possède 4 méthodes majeures qui ont permis de répondre aux attentes du sujet. Nous allons donc présenter chacune de ces méthodes avec un peu plus de détails.

La réussite de ce sujet est grandement due au livre **L'APPRENTISSAGE PROFOND AVEC PYTHON** de *François CHOLLET*. La lecture de son travail nous à permis un apprentissage rapide du fonctionnement des réseaux neuronal convolutif ainsi que leur mise en application pour notre IA.

4.1 preprocessing

Cette méthode prend comme argument le directory qui contient les images avec la structure décrite dans la partie *Description des données* afin d'effectuer le prétraitement.

Dans un premier temps cette méthode va donc créer un répertoire pour les données d'entrainement, puis un autre pour les données de validation en ce basant sur le répertoire donné en argument.

Ensuite, deux *ImageDataGenerator* vont être créer avec une mise à l'échelle de 1/255. En effet il est beaucoup plus simple pour l'IA de travailler avec des données entre 0 et 1 que 0 et 255 concernant le RGB de chaque pixels. Le premier ImageDataGenerator va servir pour les données d'entrainement et le deuxième pour la validation.

Finalement deux générateurs vont être créés pour les deux parties avec comme taille voulue 150x150 pixels étant donnée que cela correspond à la taille des images dans le jeu de données. Le *batch_size* est de **32** et le *class_mode* est en **binary** étant donné que nous avons comme résultat un vecteur constitué de 0 et de 1.

Les générateurs d'entrainement et validation ainsi créés vont être retourné par la méthodes afin de les utiliser dans les autres.

4.2 PreProcessing.py

Ce fichier permet d'effectuer un pré traitement sur les textes du premier data frame afin de facilité le travail d'apprentissage de l'IA. Il s'appuie notamment sur la librairie *nltk*.

Tout d'abord le texte va être mit en minuscule afin de ne pas différencier des mots n'ayant pas la même casse. Ensuite chacun de ses mots (y compris la ponctuation) va être transformé en « **token** ». De ces tokens sera retiré la ponctuation ainsi que ce que l'on appelle les **STOPWORDS**. Ce sont les mots n'ayant pas de grande valeurs syntaxique comme les déterminants, adjectifs possessifs, conjonctions de coordination, verbes communs (être, avoir)...

L'étape suivante consiste à effectuer une **lemmatisation** sur chacun des tokens. La lemmatisation est un traitement lexical qui consiste à appliquer aux verbes, substantifs, adjectifs... un codage renvoyant à leur entrée lexicale commune, leur « forme canonique ».

Exemple :
Les / la
étoiles /étoile
claires / clair
luisent / luire
noire / noir

Voici une image des phrases du dataframe précédent où celles-ci ont été pré-traitées :

FIGURE 3 – Phrases pré-traitées

4.3 Classify.py

Ce fichier a pour fonction de créer un classificateur pour chaque classes qui possèdent des données pour s'entrainer. L'intégralité de ce fichier utilise principalement la librairie *sklearn* qui est une librairie complète pour l'apprentissage d'une IA.

Remarque : Comme dit précédemment , certaines classes n'étant pas existante dans les données d'entrainement (ET dans les données de test), elles n'ont donc pas de classificateur attitré.

La première opération à faire pour pouvoir créer un classificateur, est de transformer les phrases en vecteurs. Pour cela, nous avons utilisé **TfidfVectorizer** de la librairie sklearn. Au niveau des paramètres, nous avons décidé d'utiliser **la norme "l2"** et d'utiliser des **ngram entre 1 et 8**. Une fois créé nous avons entrainé ce vectoriseur avec

les commentaires de notre jeu de d'entrainement.

Ensuite, nous pouvons commencer la classification. Pour celle-ci nous avons décidé d'utiliser la méthode **one-versus-rest** (« une contre le reste »). Cette méthode consiste à traiter chaque classe indépendamment des autre. Chaque classe représente ici un de nos couple E#C.

Dû au peu de données présente pour certaine classe, et l'effort pour en ajouter d'autre étant trop élevé, nous avons eu recourt à un algorithme d'échantillonnage pour augmenter les classes minoritaires. Cet algorithme n'est autre que **ADASYN** (Adaptive Synthetic). A partir des données existantes (au moins 5 ici), l'algorithme va en créer d'autre en utilisant le principe des *« K plus proche voisins »*. L'image ci-dessous montre l'efficacité d'ADASYN.

FIGURE 4 – L'algorithme ADASYN

Après avoir fait cela, il nous reste plus qu'à entrainer chacun des classificateurs. Nous avons utiliser donc utilisé **OneVsRestClassifier** avec comme **solver "sag"**, un **poids équilibré entre chaque classe** et un nombre d'**itération maximum de 1000**. Afin de pouvoir réutiliser ces classificateurs pour notamment évaluer une phrase à la volée, nous les stockons dans un dictionnaire ayant pour clé la catégorie et pour valeur le classificateur de cette catégorie.

La dernière fonction de cette classe est la prédiction un dataframe test. Pour ce faire, à l'aide du vectoriseur et des classificateurs, il nous suffit de pré-traiter et vectoriser chaque phrases et d'appeler pour chacune l'intégralité des classificateurs stockés dans le dictionnaire pour prédire à quelles classes (ou couple E#C) appartient chaque commentaire.

4.4 PlotResult.py

Ce fichier utilise la libraire *matplotlib* de python. Il permet de mettre en place la représentation graphiques des résultats obtenus. Les données sont récupérées lors de la classification du fichier précédent, puis sont misent dans des diagrammes en barre. Cela permet d'avoir une vision générale mais rapide des différents score évalués. En effet il y a beaucoup de classes différentes et un rapport de classification détaillés pour chacune d'elle ne serais pas pertinent.

5 Résultat obtenus

Il est à noter que les résultats peuvent être légèrement différent à chaque appel du programme ainsi les graphiques qui vont suivre ne sont pas forcément absolus.

5.1 Résultat général

Il est facilement remarquable sur la *figure 5* que la valeur 0 (donc l'absence du couple E#C) possèdent un bien meilleur score que la valeur 1. En effet un commentaire parlant d'un clavier uniquement ne va pas parler d'un écran ou d'une souris, ce qui créer un nombre conséquent de données négative (c-à-d 0) pour chaque classes. Il est alors plus facile pour l'IA de déterminé si la phrase n'appartient pas à un couple que l'inverse.

On constate effectivement que le f1-score des valeurs 1 de chaque classe est assez variable et atteins parfois même 0%. Cela s'explique avec deux raisons :
Soit par une *précision* de 0%, car certain couple dans les données de test n'ont qu'une apparition ce qui rend le résultat binaire ou bien simplement par l'échec de l'IA à classer correctement les phrases (les justifications de cette lacune sera expliquée dans l'analyse du résultat suivant).
Soit par un *recall* faible, car l'IA peut facilement trouvé la bonne entité mais pas la bonne catégorie et vis-versa, et cela se traduit par une réduction du recall et donc du f1-score.

Comme dit précédemment, la valeur du f1-score des valeurs 0 est très élévé, environ 95%, ainsi ne nous y attarderont pas plus que ça dans les analyses suivantes et nous concentrerons plus sur les résultats et scores obtenus pour les valeurs 1.

5.2 Résultat pour la valeur "1"

On remarque premièrement que les précisions de chaque couple E#C ne sont pas homogène. Cela s'explique par la différence majeure entre chaque entités et les catégories qui s'y rapportent. En effet il est beaucoup plus simple et parler de l'ergonomie d'une souris (ou d'un pavé tactile) que de l'ergonomie d'un OS, ce qui explique par exemple l'écart de précision entre ces deux couples.

De plus, certain couples possèdent un jeu de données qui se veut parfois peu ressemblant entre chaque élément y appartenant. Par exemple, dans le commentaire d'un clavier, il y aura beaucoup de chance de trouver les mots "touche","clavier" ou bien "pavé numérique", ce qui focalise l'IA sur certain éléments clés et ressort de tout cela une bonne précision. Mais par exemple, le couple *COMPANY#GENERAL* peut très bien parler de *Windows* comme d'*Apple*, *Dell* , *toshiba* ... de mille et une façon, ce que l'IA a du mal à assimiler.

Enfin, une relation logique apparait entre le support (nombre d'éléments) et la précision. Moins on a de données plus la précision est extrême (0% ou bien 100%) car soit l'IA a tout juste, soit tout faux. Au contraire, lors d'une présence d'un nombre de données assez conséquent (environ 10), la précision tend à dépasser les 50% ce qui pourrait être plus représentatif du niveau de classification. Typiquement, le nombre de données le plus élevé (214) est celui du couple *LAPTOP#GENERAL* (ce qui est assez logique en soit) et il possède une précision de 71%.

5.3 Résultat par entités

Suite à de nombreux tests sur des classifications de phrases, nous avons remarqué que l'IA se trompe parfois dans le ou les couple E#C mais arrive tout de même à reconnaitre la ou les entités E qui sont ciblés dans la phrase. Nous avons donc regroupé par entité pour plus de pertinence et avons calculé la *precision*, le *recall* et le *f1-score* de chacune.

Là encore attention, malgré le regroupement par entité, certaines n'ont qu'un seule données, ce qui explique les scores à 50%. De plus, nous avons récupéré la macro-moyenne (moyenne brute) et non la moyenne pondérée car celle-ci est figé à 99% pour toutes les entités (dû à grand nombre de cas à 0 et très peu d'erreur de l'IA dessus comme expliqué précédemment), il n'est donc pas intéressant de faire une étude dessus. Cette macro-moyenne se retrouve donc à cette valeur lorsqu'il n'y a qu'une donnée pour la valeur 1 et que l'IA ne retrouve pas la bonne entité E suivant le simple calcule suivant :

$$(99 + 0)/2 \simeq 50\%. \tag{1}$$

Mis à part ces cas précis, les scores obtenus sont relativement corrects et on peut facilement affirmer que l'IA arrive à reconnaitre les entités cible d'un commentaires. Toutefois, nous avons remarqué que lorsque la phrase aborde différentes entités de manière brève, par exemple *"L'ordinateur est cool, le clavier et le son sont correct mais l'affichage est magnifique"*, la classification de toutes ces entités semble très difficile pour l'IA.

5.4 Résultat de la polarité

Tout d'abord voici le nombre de données de base par polarité :

— negative : 768
— positive : 917
— neutral : 430
— mixed : 46

On constate que la *précision*, le *recall* et le *f1-score* sont relativement proche pour chaque polarité ce qui ne souligne pas de problème particulier dans une classe. Il est notable que *positive* et *negative* sont les deux classes qui comportent le plus de données, il est donc normal que leur score soit les plus élevés.

Le fait d'avoir un commentaire positif semble d'ailleurs plus facile à détecter qu'un commentaire négatif. Cela s'explique par le fait que souvent, un commentaire négatif consiste en l'énumération des défauts de l'ordinateur sans forcément de mot clé signifiant une insatisfaction ou un mécontentement, l'IA à donc plus de mal à déterminé si cela est négatif (une énumération de fait pouvant être neutre).

Ainsi, la polarité *neutral* peut ressortir également de plusieurs tournure de phrase. Aucun mot clé n'est propre à un avis neutre, ce qui rend plus difficile sa classification et lui donne des score relativement médians.

Finalement, la polarité *mixed* possède le score le plus bas car d'une part, le jeu de données est très faible relativement aux autres, mais également, c'est la polarité la plus dure à classifiée et n'existe pas réellement. Nous l'avons créer dû à la simplification que nous avons faite d'avoir une polarité générale et non propre à chaque couples E#C. Il est donc évident qu'elle n'est pas absolue et consiste à la fusion d'un avis positif et négatif.

6 Les améliorations possibles

Dans cette nous allons voir ce que nous aurions pu améliorer dans notre projet.

6.1 Les classe

Le plus gros problème de notre sujet sont les classes qui ont été choisies pour le jeu de données, il y en a beaucoup trop (126) dont la plupart qui sont similaire. Avec plus de temps, il aurais été préférable de refaire la plupart des classes sans garder la forme *entité#categorie*. Car pour des phrase comme *"L'ordinateur est rapide"*, cela veux dire que l'on parle de plusieurs entités comme DISPLAY, CPU, LAPTOP, GRAPHICS, SOFTWARE, ... et pour chacune, des catégories GENERAL et OPÉRATION PERFORMANCE.
Pour corriger cela, le mieux aurait été de faire moins de classe mais avec des spécificité plus générale (comme *"composant"*, *"service"*..) ce qui aurais permit de mieux classifier et d'être plus précis.

6.2 Le jeu de données

La deuxième amélioration majeure que l'on aurais pu faire aurait été d'ajouter plus de données à notre jeu d'entrainement ou bien même avoir un jeu de données plus conséquent et plus orienté dans la conception choisie avec les différents couples. Cet élément est ce qui nous à poser le plus de problème dans le projet.

6.3 Les marques et modèles

Quand on parle de matériels informatique, il arrive souvent de parler d'un modèle spécifique. Pour les processeur le vocabulaire CPU ou processeur ne vas pas toujours être employé, l'utilisateur utilisera plutôt les mots *"i7"* ou *"intel"* ou encore *"ryzen"* qui sont des modèle des processeurs. Cela se retrouve sur plusieurs entités différentes. Il aurait été possible de changer les Tokens des modèles spécifiques lors de leur récupération par un mot plus général, si nous reprenons l'exemple il serait possible de tout remplacer par le mot *"processeur"*. Cela aurait surement amélioré l'efficacité de l'IA dans sa classification.

6.4 La polarité

Avec le choix de faire une polarité générale par phrase, il y a un travail assez intéressant qui s'est retrouvé écarté. En effet, il aurait été intéressant d'essayer de faire une polarité pour chaque couple trouvées comme demandé dans le SemEval. Pour cela, il aurais fallu réussir à isoler le morceau de phrase qui a permit de détecter la classe. Puis à partir de ce sous-ensemble, faire de nouveau une classification sur la polarité. Par manque de temps et d'expérience, cette isolation de sous-ensemble par couple fut trop complexe, il aurai fallu partir d'une IA avec une classification déjà presque parfaite afin de ne pas sélectionner une sous-ensemble incohérent. Ce qui n'était pas notre cas.

7 Conclusion

La mise en place de ce projet fut un challenge. En effet, il nous a fallu construire nous même chaque couple entité-catégorie et utiliser la classification *one-vs-rest* que nous ne maitrisions pas du tout.
La partie la plus simple fut de pré-traiter le texte et de récupérer les données depuis le document XML et d'en faire des diagrammes. Sûrement dû à nos connaissances acquises au court de notre cursus et de cette matière.
Une fois la classification mise en place, nous avons heurter le problème du manque de données qui dans un premier temps ne fût pas corriger par l'algorithme *ADASYN*, car le jeu données étant vraiment trop pauvre pour certaine classes celui-ci en créait de nouvelles un peu diverse (l'IA répondait toujours non aux tests pour les classes concernées). Il nous a donc fallu ajouter énormément de données ce qui est dommage car nous n'en avons tiré aucun apprentissage.
Plus généralement, ce projet nous aura permit de nous familiariser avec les librairies *sklearn* et *nltk*, d'apprendre de nouvelles façon de classifier lorsque plusieurs classes peuvent apparaitre pour un même cas, mais également de solutionner le problème du dés-équilibrage des classes et des valeurs et finalement de manipuler l'apprentissage d'une IA avec des données textuelles et leur pré-traitement.

8 Références

— Top Rated in Laptop Computers :
https://www.amazon.com/pcr/Top-Rated-Laptop-Computers-Reviews/565108
— The "Imbalanced classification problem" :
https://medium.com/@bluedme_tech/comment-traiter-les-probl%C3%A8mes-de-classification-d%C3%A9s%C3%A9quilibr%C3%A9e-en-machine-learning-8c3bc95ca25b
— Deep dive into multi-label classification :
https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff
— Tutoriel TAL pour les débutants :
https://www.actuia.com/contribution/victorbigand/tutoriel-tal-pour-les-debutants-classification-de-texte/
— SemEval-2015 Task 12 :
https://alt.qcri.org/semeval2015/task12/index.php?id=data-and-tools
— WIKIPEDIA FR :
https://fr.wikipedia.org/wiki/Wikip%C3%A9dia:Accueil_principal

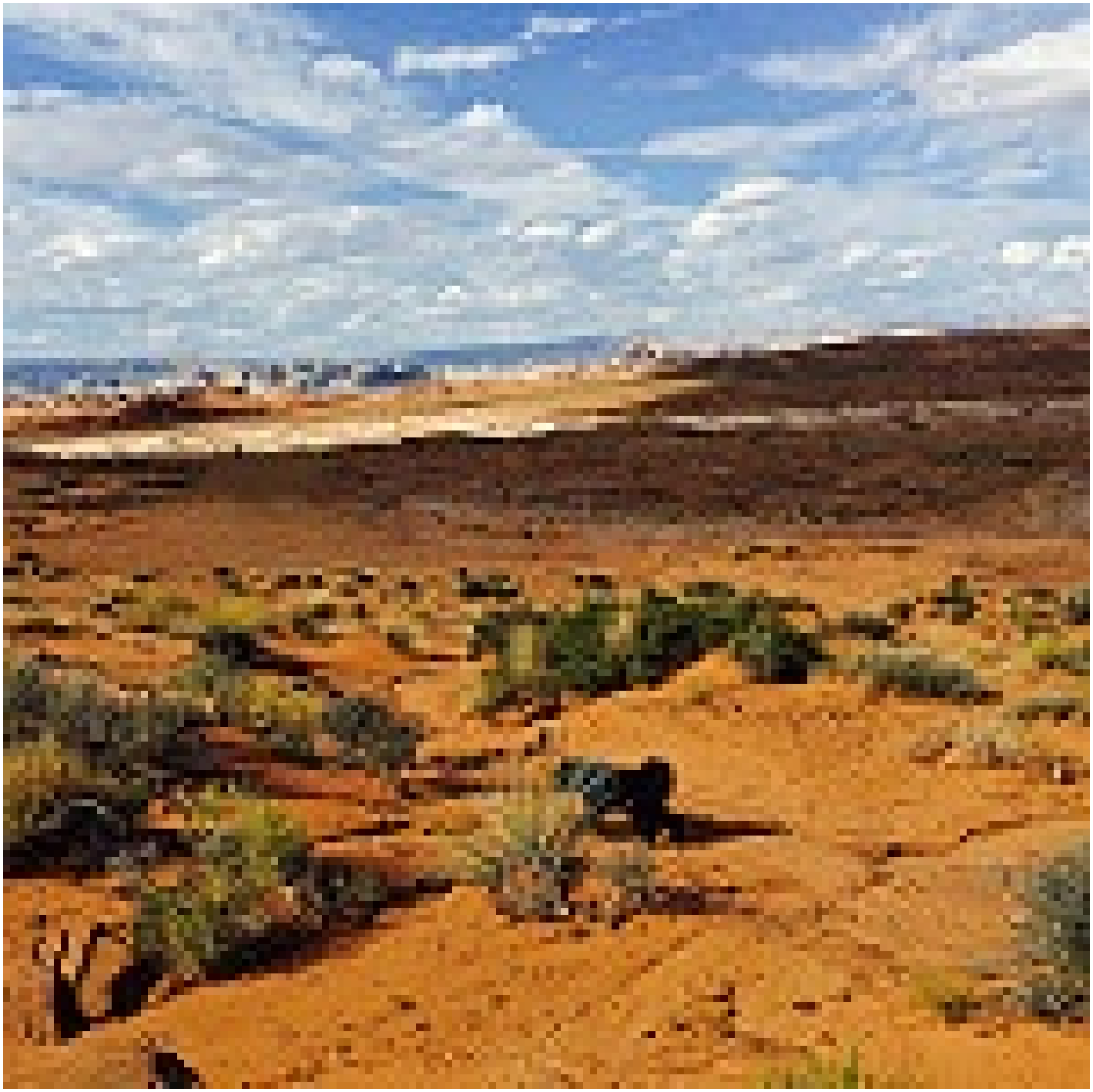

FIGURE 5 – F1-score de chaque classe pour 0 et 1

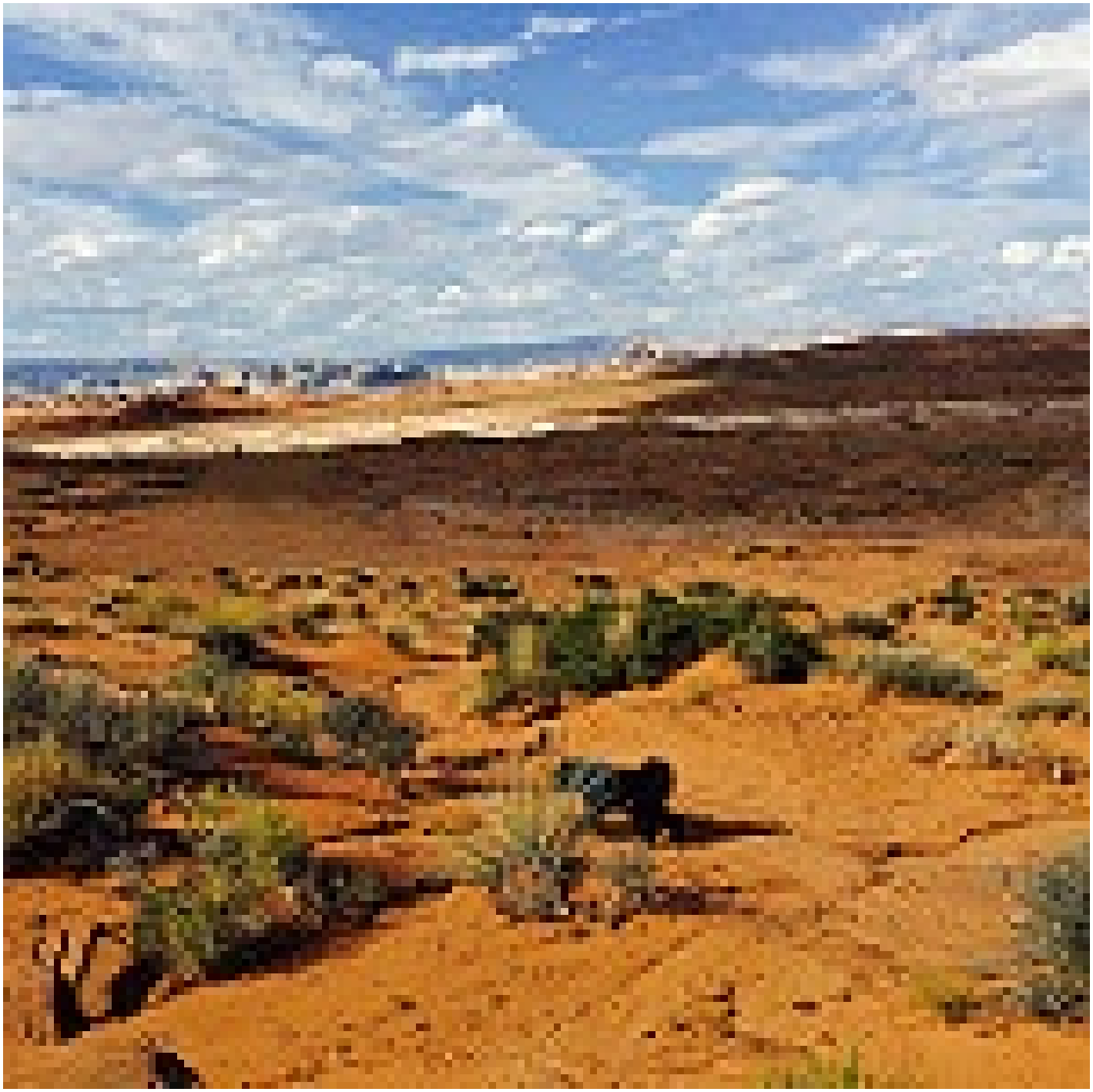

FIGURE 6 – précision et support de la valeur 1

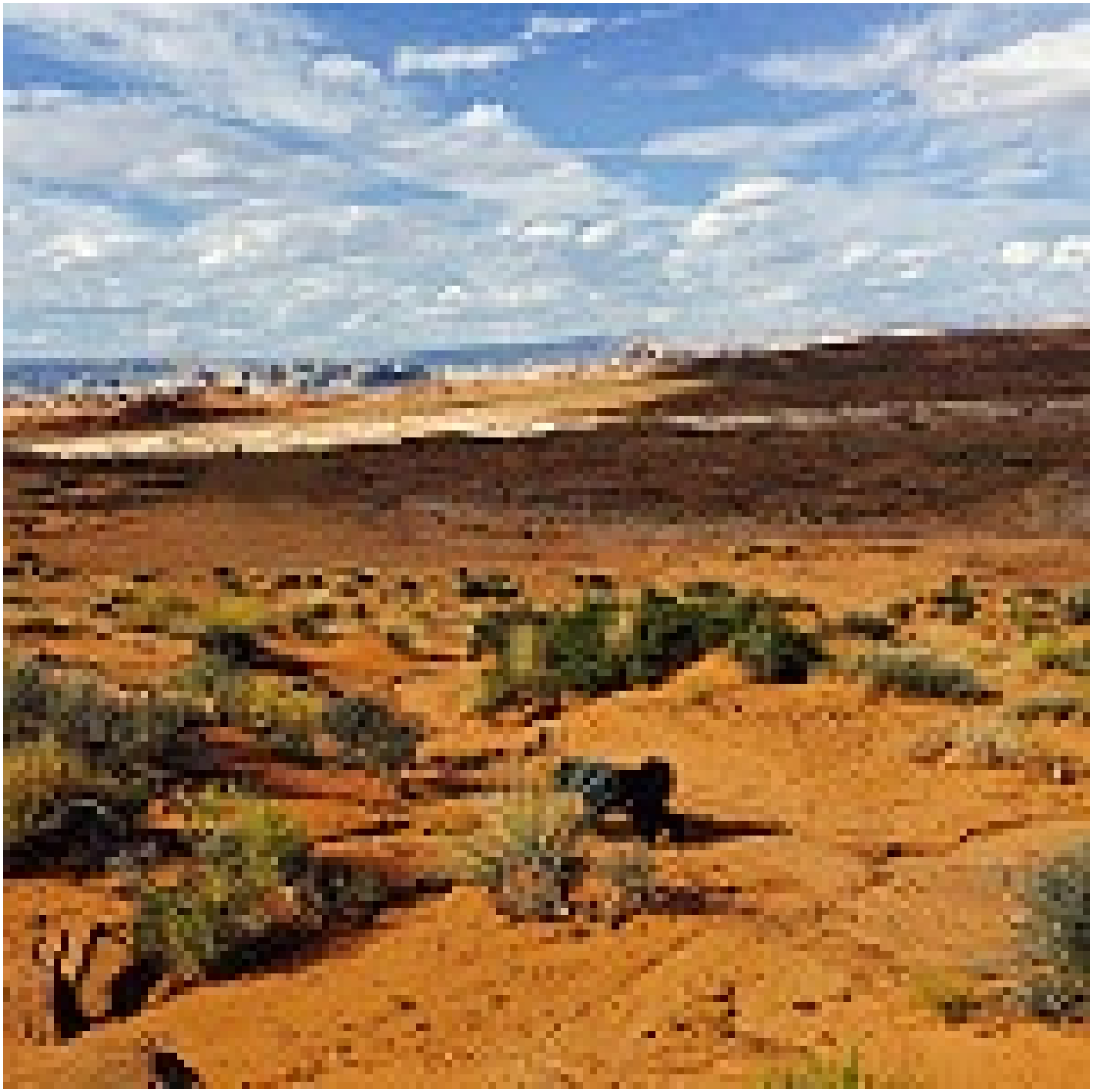

FIGURE 7 – score par entités

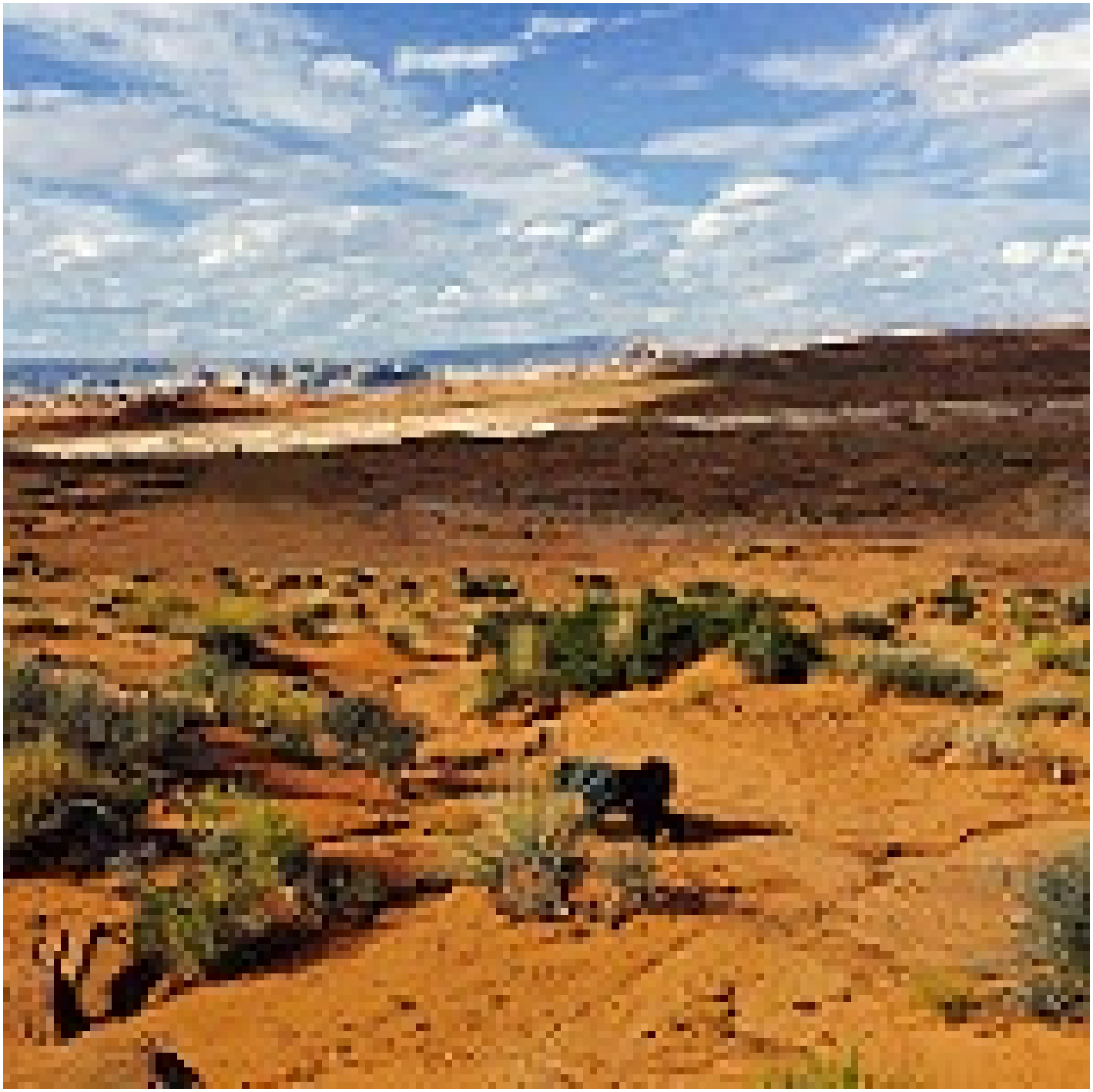

FIGURE 8 – score de la polarité