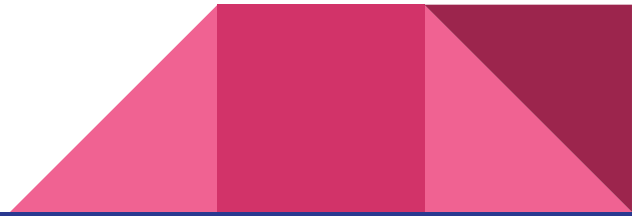
TP4

Organización de datos - 1C 2022

Tarea : KNN sobre dibujos

Utilice KNN sobre una reducción de dimensiones que elija para predecir la categoría de los dibujos. ¿Qué top-3 accuracy puede conseguir?



Dataset

The Quick, Draw! Dataset

The Quick Draw Dataset is a collection of 50 million drawings across 345 categories, contributed by players of the game Quick, Draw!. The drawings were captured as timestamped vectors, tagged with metadata including what the player was asked to draw and in which country the player was located. You can browse the recognized drawings on quickdraw.withgoogle.com/data.

We're sharing them here for developers, researchers, and artists to explore, study, and learn from. If you create something with this dataset, please let us know by e-mail or at A.I. Experiments.

We have also released a tutorial and model for training your own drawing classifier on tensorflow.org.

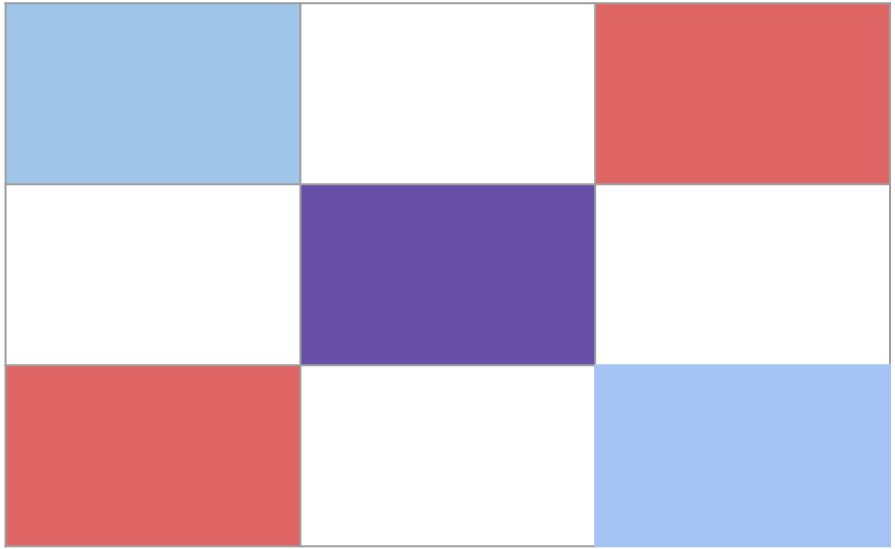
Please keep in mind that while this collection of drawings was individually moderated, it may still contain inappropriate content.

Link : Dataset

Conversión de datos

T1 ((1,1), (2,2), (3,3)) Rojo

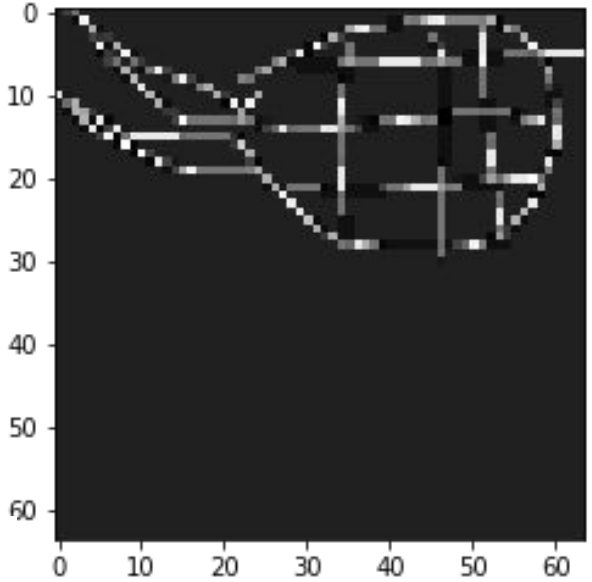
T2 ((3,1), (2,2), (1,3) Azul



Ej:

Ej de los trazos de una imagen,
y su respectiva imagen

```
((34, 10, 5, 5, 12, 29, 63, 93, 110, 115, 115, 112, 107, 82, 61),
 (90, 142, 186, 207, 225, 238, 242, 230, 215, 194, 156, 136, 128, 101, 87))
```



Baseline (Regresión Logística)

Matriz sin reducción:

	precision	recall	f1-score	support
apple	0.64	0.69	0.67	845
asparagus	0.76	0.79	0.77	820
banana	0.69	0.71	0.70	1570
blackberry	0.44	0.41	0.42	548
blueberry	0.30	0.25	0.27	617
broccoli	0.58	0.58	0.58	689
grapes	0.40	0.38	0.39	713
onion	0.45	0.38	0.41	686
pineapple	0.57	0.61	0.59	652
potato	0.73	0.81	0.77	1699
strawberry	0.48	0.47	0.47	535
watermelon	0.46	0.40	0.43	626
accuracy			0.59	10000
macro avg	0.54	0.54	0.54	10000
weighted avg	0.58	0.59	0.59	10000

```
top_k_accuracy_score(y_true = y_valid, y_score=proba_preds, k=3)
```

```
0.8443
```

Baseline (Regresión Logística)

PCA :

n° componentes : 200

	precision	recall	f1-score	support
apple	0.70	0.79	0.74	845
asparagus	0.79	0.82	0.81	820
banana	0.71	0.80	0.75	1570
blackberry	0.64	0.47	0.55	548
blueberry	0.48	0.29	0.36	617
broccoli	0.63	0.68	0.66	689
grapes	0.56	0.53	0.54	713
onion	0.60	0.51	0.55	686
pineapple	0.66	0.70	0.68	652
potato	0.75	0.88	0.81	1699
strawberry	0.60	0.53	0.57	535
watermelon	0.58	0.45	0.50	626
accuracy			0.67	10000
macro avg	0.64	0.62	0.63	10000
weighted avg	0.66	0.67	0.66	10000

```
top_k_accuracy_score(y_true = y_valid, y_score=proba_preds, k=3)
```

```
0.8865
```

Baseline (Regresión Logística)

SVD :

n° componentes : 200

	precision	recall	f1-score	support
apple	0.70	0.79	0.74	845
asparagus	0.80	0.82	0.81	820
banana	0.71	0.80	0.75	1570
blackberry	0.65	0.48	0.55	548
blueberry	0.47	0.28	0.35	617
broccoli	0.63	0.67	0.65	689
grapes	0.55	0.54	0.55	713
onion	0.60	0.50	0.55	686
pineapple	0.66	0.69	0.67	652
potato	0.75	0.88	0.81	1699
strawberry	0.59	0.53	0.56	535
watermelon	0.59	0.45	0.51	626
accuracy			0.67	10000
macro avg	0.64	0.62	0.63	10000
weighted avg	0.66	0.67	0.66	10000

```
top_k_accuracy_score(y_true = y_valid, y_score=proba_preds, k=3)
```

```
0.8866
```

KNN : Búsqueda de hiperparametros

```
search_space_svd = {
    "n_components": [10,15,20,22,30],
    "algorithm" : ["arpack", "randomized"],

}
```

```
search_space_knn = {
    "n_neighbors" : [10,15,20,25,40],
    "p": [1,2],
    "algorithm" : ["auto", "ball_tree", "kd_tree", "brute"],
}
```

KNN : Búsqueda de hiperparametros

```
best_estimators = {}
for i in tqdm(range(5)):
    nComponents = random.choice(search_space_svd["n_components"])
    alg = random.choice(search_space_svd["algorithm"])
    svd = TruncatedSVD(random_state=42, n_components = nComponents, algorithm = alg)
    x_train = svd.fit_transform(x_train)
    # 5 corridas x 2 RS = 10 combinaciones
    print("HP SVD: \nn_components : {}\nalgorithm: {}".format(nComponents, alg))
    rs = RandomizedSearchCV(estimator=knn,
                            param_distributions = search_space_knn,
                            scoring = "accuracy",
                            n_jobs=1,
                            n_iter = 2,
                            verbose=3)
    rs.fit(x_train, y_train)
    k = rs.best_params_
    k["n_components"] = nComponents
    k["svd_algorith"] = alg

    best_estimators[rs.best_score_] = k
    x_train = train_copy
```

Mejor modelo

```
{'algorithm': 'kd_tree',
 'n_components': 22,
 'n_neighbors': 20,
 'p': 2,
 'svd_algorith': 'randomized'}
```

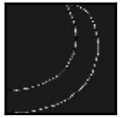
KNN : Resultados

```
top_k_accuracy_score(y_true = y_valid , y_score = preds, k=3)
```

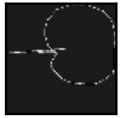
```
0.9116
```

Algunas Predicciones

y_true: banana
top_valid:
1° banana : 1.0
2° - : 0.0
3° - : 0.0



y_true: apple
top_valid:
1° apple : 0.9
2° blueberry : 0.05
3° blueberry : 0.05



y_true: banana
top_valid:
1° banana : 0.6
2° pineapple : 0.15
3° potato : 0.1

y_true: apple
top_valid:
1° potato : 0.65
2° blueberry : 0.25
3° banana : 0.1

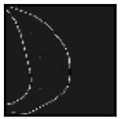
y_true: pineapple
top_valid:
1° blueberry : 0.4
2° strawberry : 0.25
3° apple : 0.2

y_true: potato
top_valid:
1° potato : 0.9
2° broccoli : 0.05
3° broccoli : 0.05

y_true: potato
top_valid:
1° potato : 1.0
2° - : 0.0
3° - : 0.0

y_true: apple
top_valid:
1° apple : 0.95
2° potato : 0.05
3° - : 0.0

y_true: watermelon
top_valid:
1° watermelon : 0.6
2° banana : 0.4
3° - : 0.0

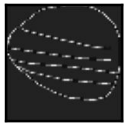


Algunas Predicciones

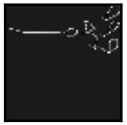
y_true: grapes
top_valid:
1° grapes : 0.6
2° blackberry : 0.25
3° blueberry : 0.1

y_true: watermelon
top_valid:
1° blueberry : 0.35
2° onion : 0.2
3° apple : 0.15

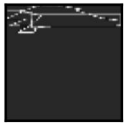
y_true: watermelon
top_valid:
1° watermelon : 0.5
2° onion : 0.4
3° blackberry : 0.05



y_true: grapes
top_valid:
1° banana : 0.45
2° grapes : 0.35
3° asparagus : 0.15



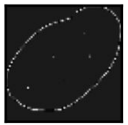
y_true: asparagus
top_valid:
1° asparagus : 1.0
2° - : 0.0
3° - : 0.0



y_true: watermelon
top_valid:
1° watermelon : 0.95
2° broccoli : 0.05
3° - : 0.0

y_true: broccoli
top_valid:
1° grapes : 0.85
2° blackberry : 0.15
3° - : 0.0

y_true: potato
top_valid:
1° potato : 1.0
2° - : 0.0
3° - : 0.0



y_true: banana
top_valid:
1° banana : 1.0
2° - : 0.0
3° - : 0.0