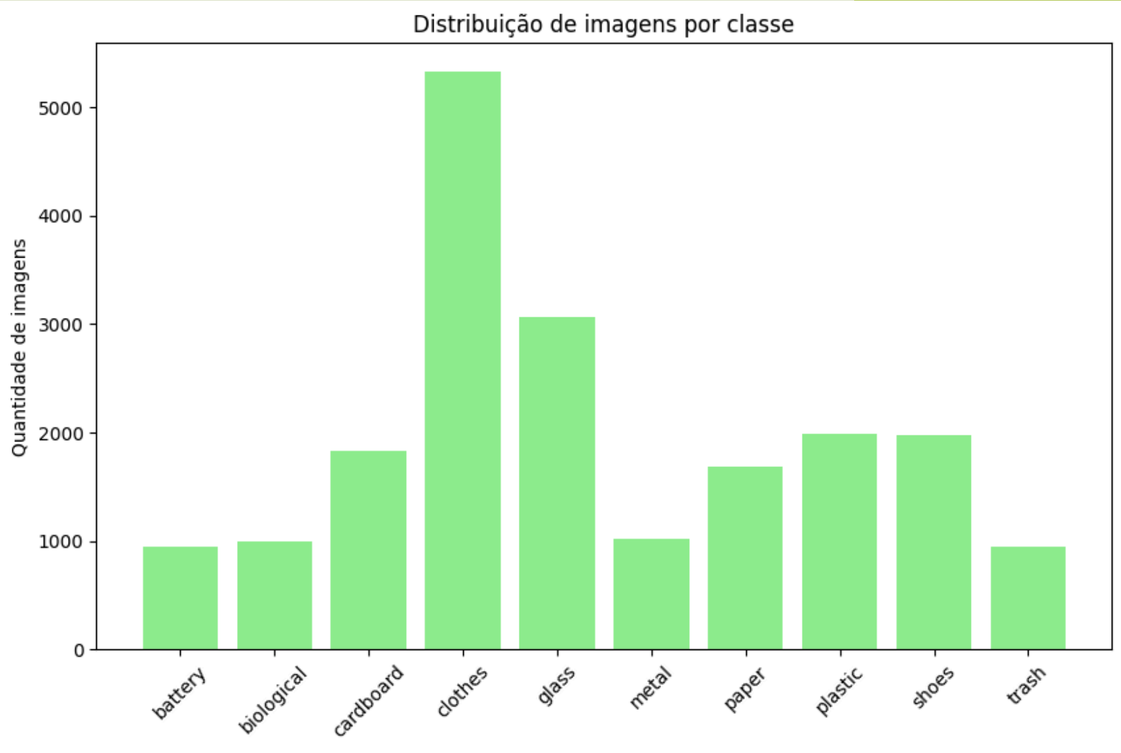
Fernando Felix
Luis Felipe

RECICLAGEM INTELIGENTE:

COMPARANDO ALGORITMOS DE ML NA IDENTIFICAÇÃO DE RESÍDUOS

Trabalho final de IA

DATASET ESCOLHIDO



Fonte
https://www.kaggle.com/datasets/sumn2u/garbage-classification-v2

KNN

- Adaptável
- Poucos hiperparâmetros
- Sensível a outliers

KNeighborsClassifier

Parameters

n_neighbors	3
weights	'uniform'
algorithm	'auto'
leaf_size	30
p	2
metric	'minkowski'
metric_params	None
n_jobs	None

```python
# Treinamento do classificador KNN
knn = KNeighborsClassifier(n_neighbors=n)
knn.fit(X_train, y_train)
```

✓ 0.0s

	precision	recall	f1-score	support
battery	0.36	0.51	0.42	182
biological	0.27	0.62	0.37	197
cardboard	0.42	0.52	0.47	361
clothes	0.69	0.73	0.71	1043
glass	0.54	0.48	0.51	626
metal	0.40	0.32	0.35	215
paper	0.47	0.44	0.45	357
plastic	0.51	0.31	0.39	390
shoes	0.44	0.30	0.35	391
trash	0.49	0.36	0.41	191
accuracy			0.50	3953
macro avg	0.46	0.46	0.44	3953
weighted avg	0.52	0.50	0.50	3953

RANDOM FOREST

- Bom poder preditivo
- Escalável
- Computacionalmente intensivo

RandomForestClassifier

Parameters

n_estimators	100
criterion	'gini'
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	'sqrt'
max_leaf_nodes	None
min_impurity_decrease	0.0
bootstrap	True
oob_score	False
n_jobs	None
random_state	None
verbose	0
warm_start	False
class_weight	None
ccp_alpha	0.0
max_samples	None
monotonic_cst	None

```python
# Treinamento do classificador Random Forest
rf = RandomForestClassifier(n_estimators=n)
rf.fit(X_train, y_train)
```

✓ 15.8s

	precision	recall	f1-score	support
battery	0.79	0.53	0.63	207
biological	0.67	0.65	0.66	192
cardboard	0.71	0.71	0.71	353
clothes	0.63	0.92	0.75	1090
glass	0.64	0.74	0.68	600
metal	0.76	0.32	0.45	219
paper	0.73	0.54	0.62	324
plastic	0.69	0.52	0.60	392
shoes	0.60	0.38	0.47	381
trash	0.71	0.42	0.53	195
accuracy			0.66	3953
macro avg	0.69	0.57	0.61	3953
weighted avg	0.67	0.66	0.65	3953

SVM(SUPPORT VECTOR MACHINE)

- Geralmente alcança bons resultados
- Suporta diferentes funções kernel
- Costuma ser mais lento para treinar

SVC

▼ Parameters

C	1.0
kernel	'rbf'
degree	3
gamma	'scale'
coef0	0.0
shrinking	True
probability	False
tol	0.001
cache_size	200
class_weight	None
verbose	False
max_iter	-1
decision_function_shape	'ovr'
break_ties	False
random_state	None

```python
# Treinar modelo SVM
clf = SVC()
clf.fit(X_train, y_train)
```

✓ 1m 57.1s

	precision	recall	f1-score	support
battery	0.56	0.49	0.52	164
biological	0.55	0.44	0.49	176
cardboard	0.54	0.56	0.55	395
clothes	0.57	0.95	0.72	1089
glass	0.46	0.61	0.53	596
metal	0.56	0.21	0.30	202
paper	0.41	0.28	0.33	344
plastic	0.57	0.22	0.31	413
shoes	0.54	0.21	0.31	390
trash	0.79	0.12	0.22	184
accuracy			0.54	3953
macro avg	0.55	0.41	0.43	3953
weighted avg	0.54	0.54	0.49	3953

ÁRVORE DE DECISÃO

- Fácil de interpretar
- Rápido para treinar e prever
- Pode sofrer overfitting

DecisionTreeClassifier

▼ Parameters

criterion	'gini'
splitter	'best'
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_fraction_leaf	0.0
max_features	None
random_state	None
max_leaf_nodes	None
min_impurity_decrease	0.0
class_weight	None
ccp_alpha	0.0
monotonic_cst	None

```python
# Treinar modelo Decision Tree
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
```

✓ 5.4s

	precision	recall	f1-score	support
battery	0.39	0.44	0.41	164
biological	0.35	0.42	0.38	176
cardboard	0.53	0.45	0.49	395
clothes	0.67	0.67	0.67	1089
glass	0.50	0.51	0.51	596
metal	0.30	0.26	0.28	202
paper	0.39	0.42	0.41	344
plastic	0.45	0.43	0.44	413
shoes	0.33	0.35	0.34	390
trash	0.34	0.34	0.34	184
accuracy			0.49	3953
macro avg	0.43	0.43	0.43	3953
weighted avg	0.49	0.49	0.49	3953

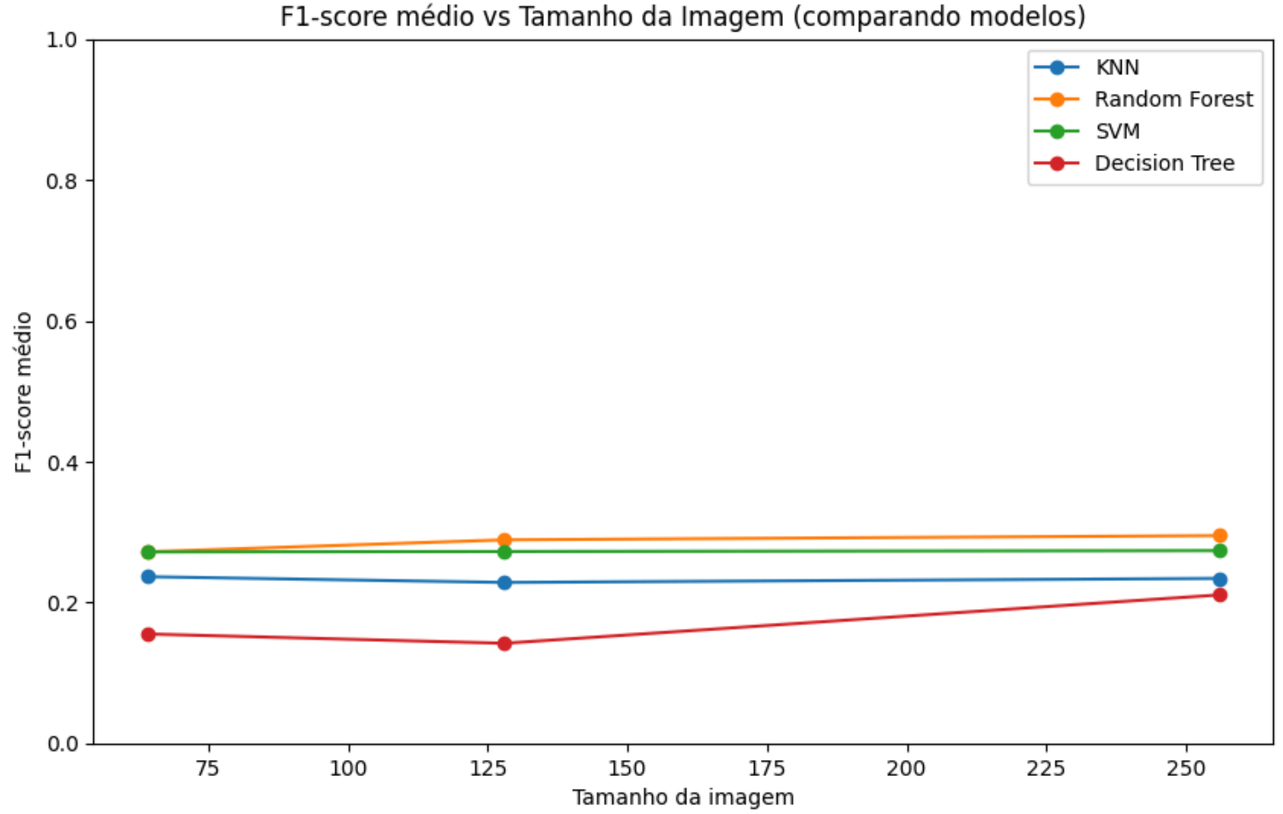
MUDANÇA DE PARÂMETROS

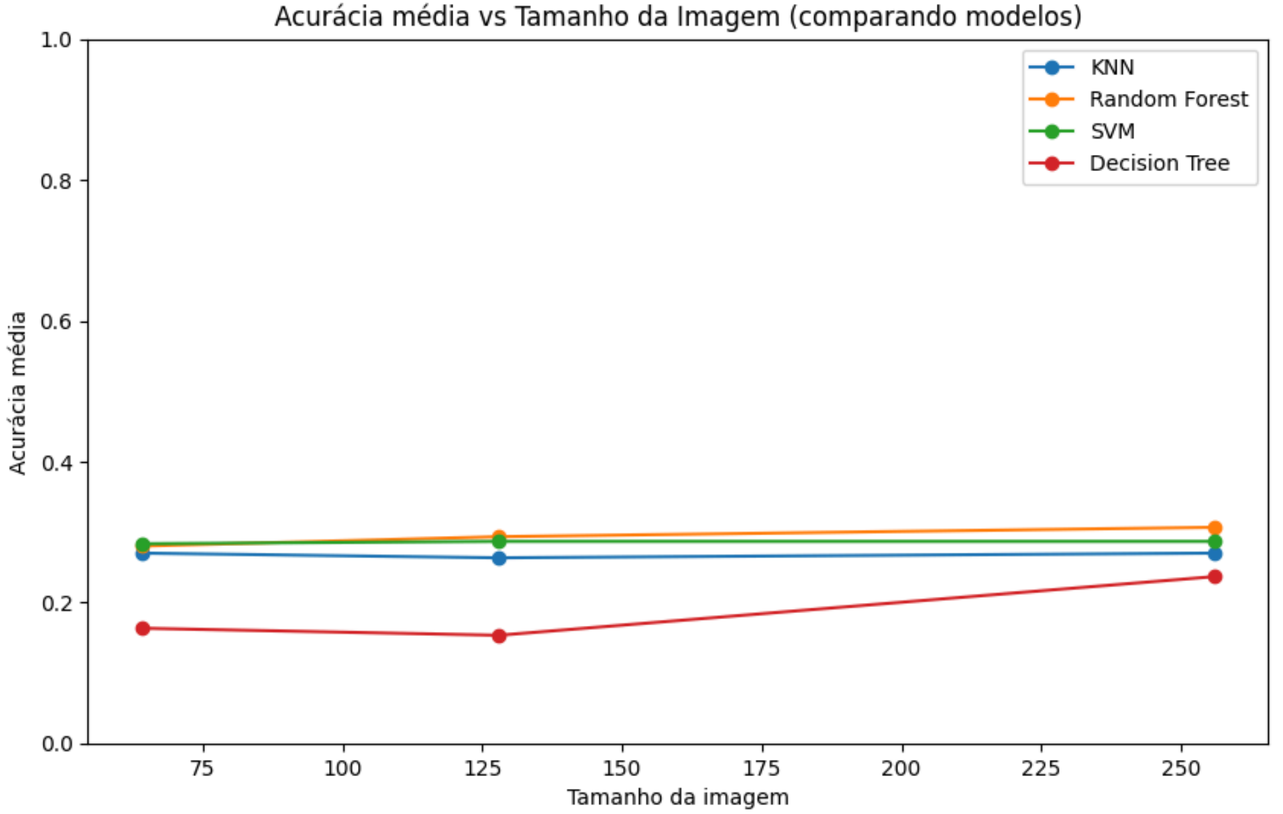
```python
results = []

# Loop para diferentes tamanhos de imagem e modelos
for img_size in [64, 128, 256]:
    X, y = [], []

    # Carregar dados reais (limitar para rodar mais rápido)
    for label in os.listdir(dataset_path):
        class_dir = os.path.join(dataset_path, label)
        if os.path.isdir(class_dir):
            # A quantidade de imagens a serem analisadas de cada classe foi reduzida para que o código consiga rodar o exemplo mais facilment
            for img_file in os.listdir(class_dir)[:50]:
                img_path = os.path.join(class_dir, img_file)
                features = extract_features(img_path, img_size)
                X.append(features)
                y.append(label)

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # KNN
    for k in [3,5,7]:
        clf = KNeighborsClassifier(n_neighbors=k)
        clf.fit(X_train, y_train)
        preds = clf.predict(X_test)
        report = classification_report(y_test, preds, output_dict=True, zero_division=0)
        results.append({
            'model': 'KNN', 'img_size': img_size, 'param': f'n_neighbors={k}',
            'param_value': k,
            'accuracy': report['accuracy'],
            'f1_score': report['macro avg']['f1-score']
        })
```

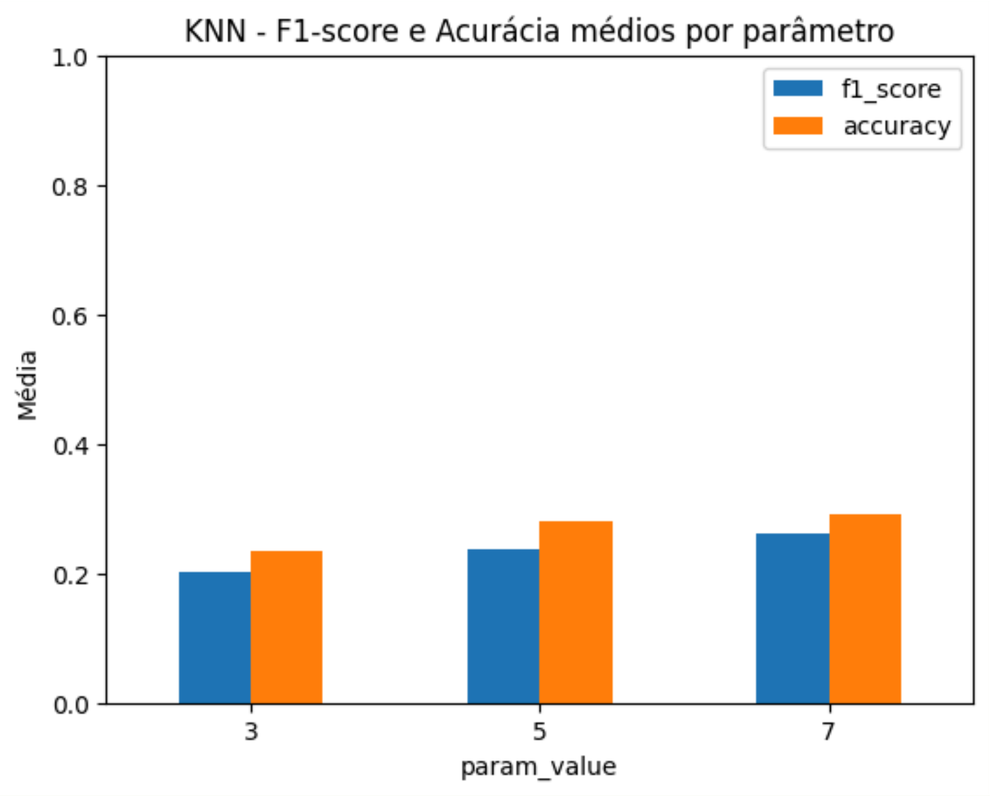
```python
    # Random Forest
    for n in [50,100,150]:
        clf = RandomForestClassifier(n_estimators=n, random_state=42)
        clf.fit(X_train, y_train)
        preds = clf.predict(X_test)
        report = classification_report(y_test, preds, output_dict=True, zero_division=0)
        results.append({
            'model': 'Random Forest', 'img_size': img_size, 'param': f'n_estimators={n}',
            'param_value': n,
            'accuracy': report['accuracy'],
            'f1_score': report['macro avg']['f1-score']
        })

    # SVM
    for kernel in ['linear','rbf','poly']:
        clf = SVC(kernel=kernel)
        clf.fit(X_train, y_train)
        preds = clf.predict(X_test)
        report = classification_report(y_test, preds, output_dict=True, zero_division=0)
        results.append({
            'model': 'SVM', 'img_size': img_size, 'param': f'kernel={kernel}',
            'param_value': kernel,
            'accuracy': report['accuracy'],
            'f1_score': report['macro avg']['f1-score']
```

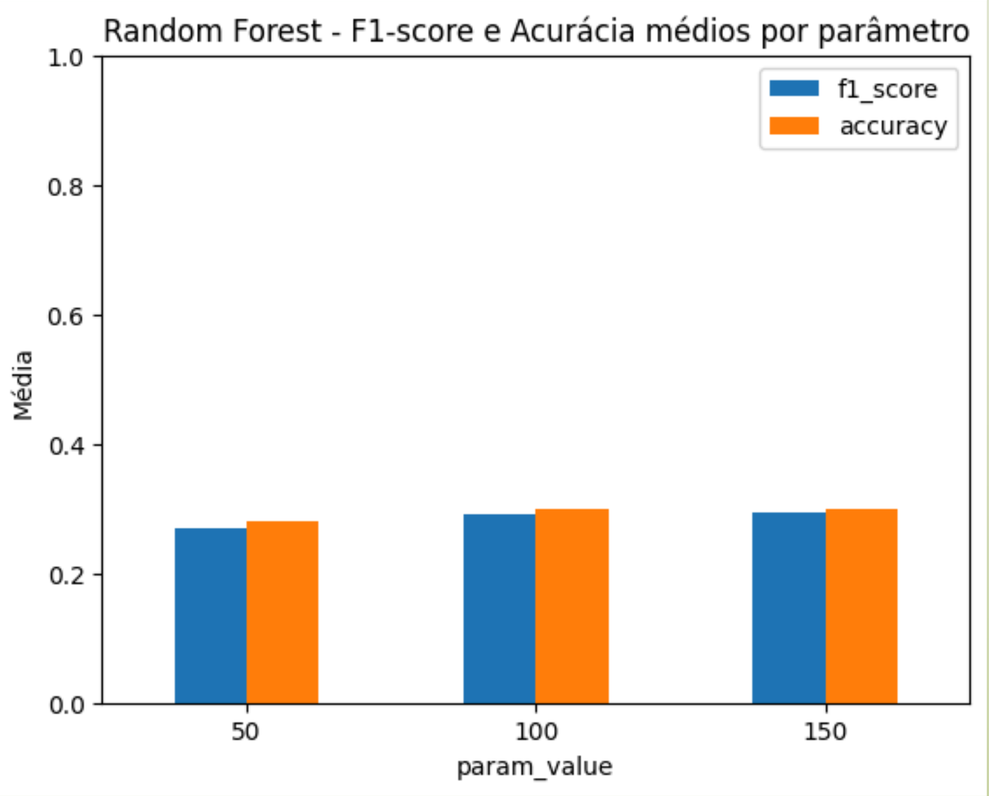
```python
# Decision Tree
for depth in [5,10,20]:
    clf = DecisionTreeClassifier(max_depth=depth, random_state=42)
    clf.fit(X_train, y_train)
    preds = clf.predict(X_test)
    report = classification_report(y_test, preds, output_dict=True, zero_division=0)
    results.append({
        'model': 'Decision Tree', 'img_size': img_size, 'param': f'max_depth={depth}',
        'param_value': depth,
        'accuracy': report['accuracy'],
        'f1_score': report['macro avg']['f1-score']
    })
```

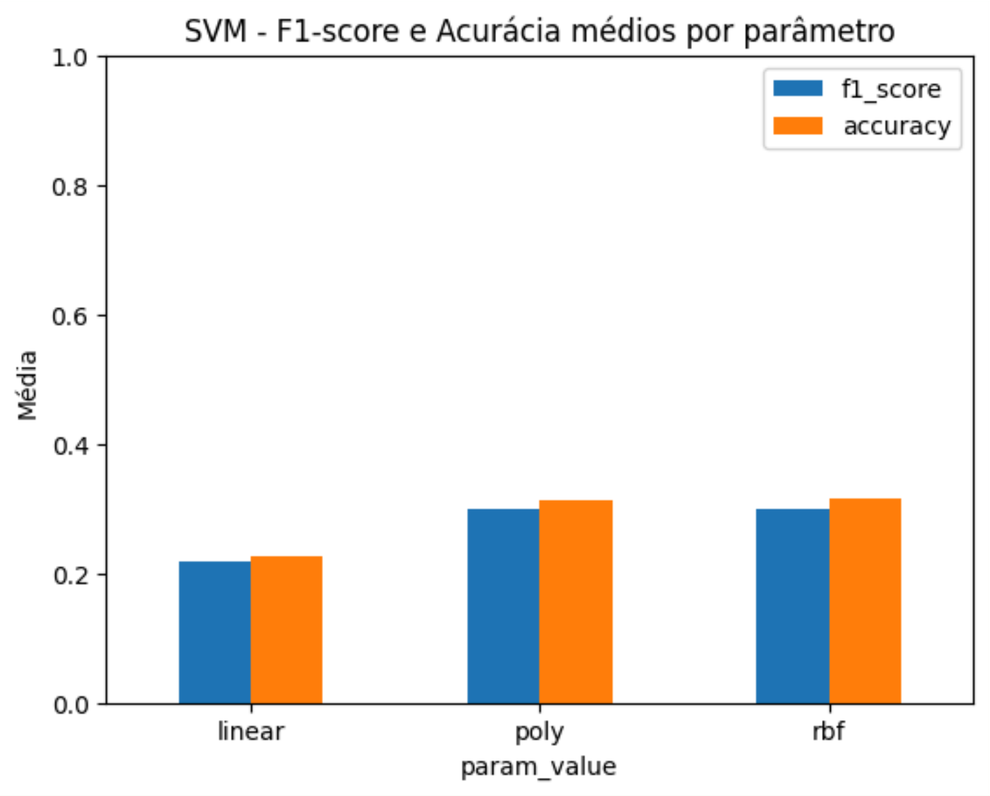
RESULTADOS

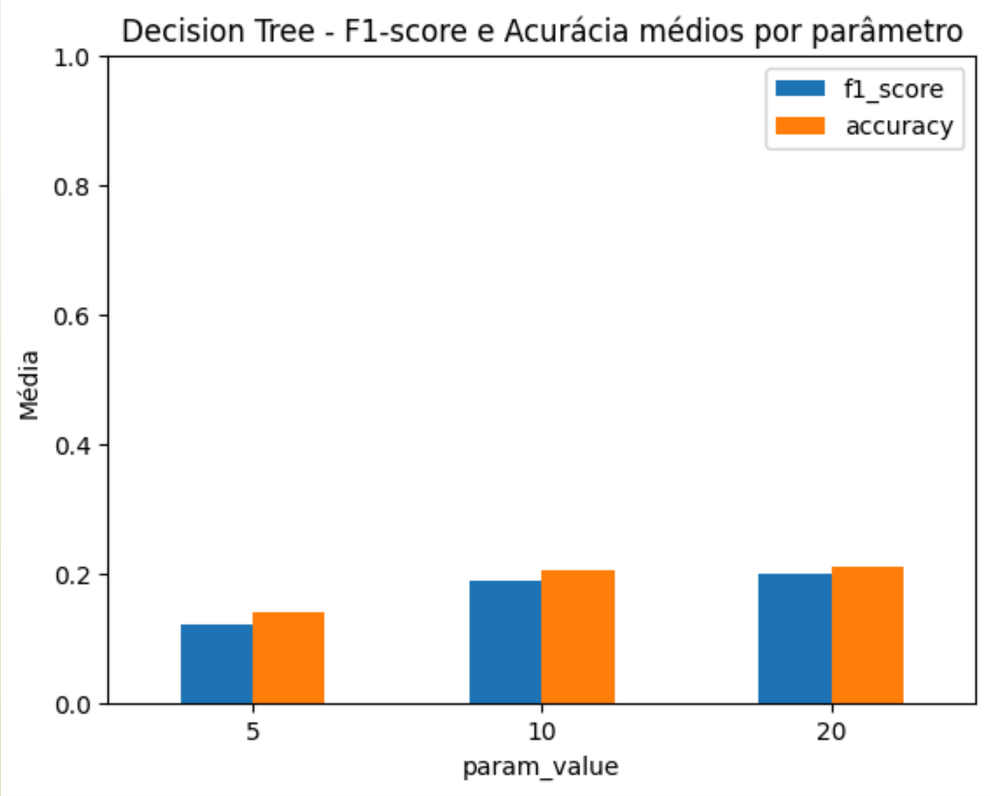

F1-score médio vs Tamanho da Imagem (comparando modelos)
KNN
Random Forest
SVM
Decision Tree
F1-score médio
Tamanho da imagem
1.0
0.8
0.6
0.4
0.2
0.0
75
100
125
150
175
200
225
250

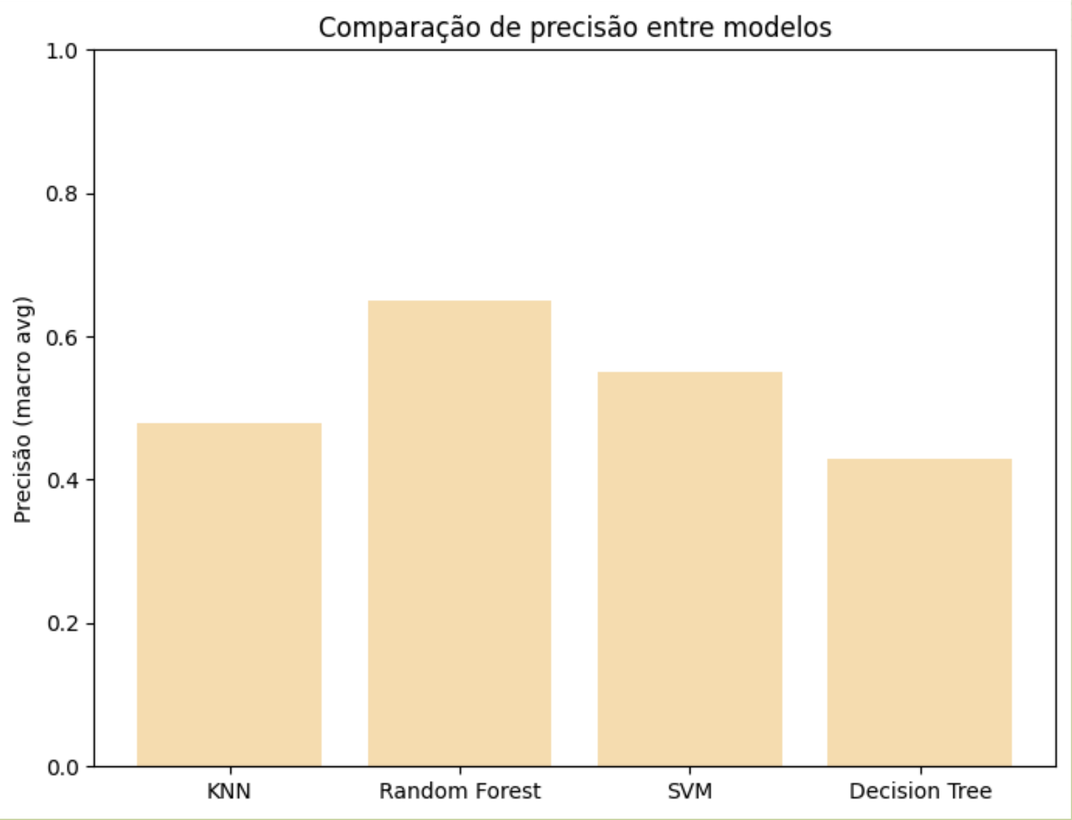

Acurácia média vs Tamanho da Imagem (comparando modelos)
KNN
Random Forest
SVM
Decision Tree
Acurácia média
Tamanho da imagem
1.0
0.8
0.6
0.4
0.2
0.0
75
100
125
150
175
200
225
250

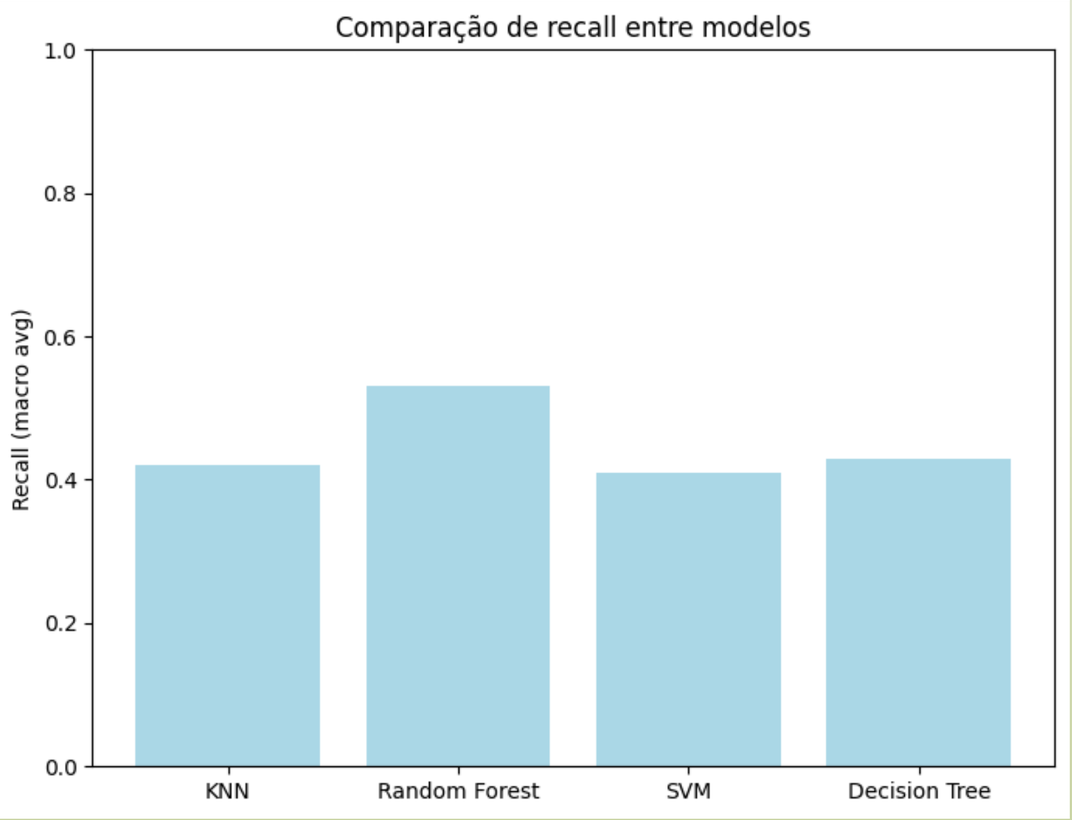

KNN - F1-score e Acurácia médios por parâmetro
f1_score
accuracy
Média
param_value
1.0
0.8
0.6
0.4
0.2
0.0
3
5
7

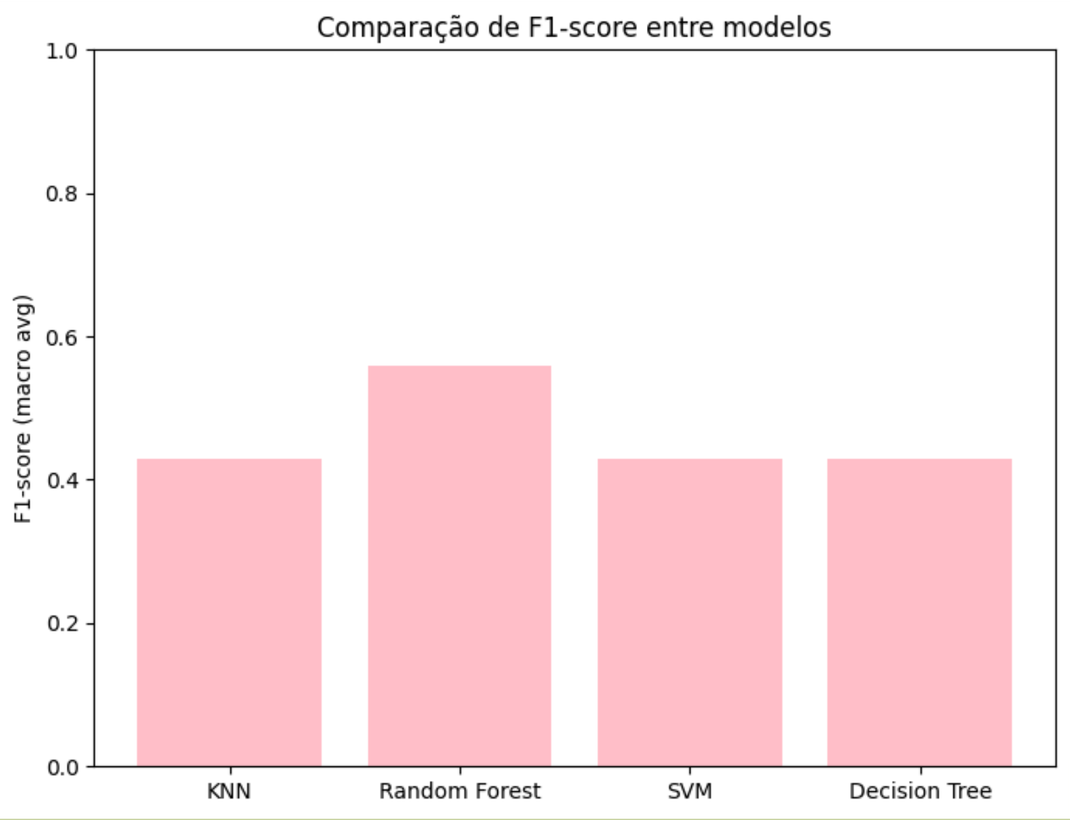

Random Forest - F1-score e Acurácia médios por parâmetro
f1_score
accuracy
Média
param_value
1.0
0.8
0.6
0.4
0.2
0.0
50
100
150

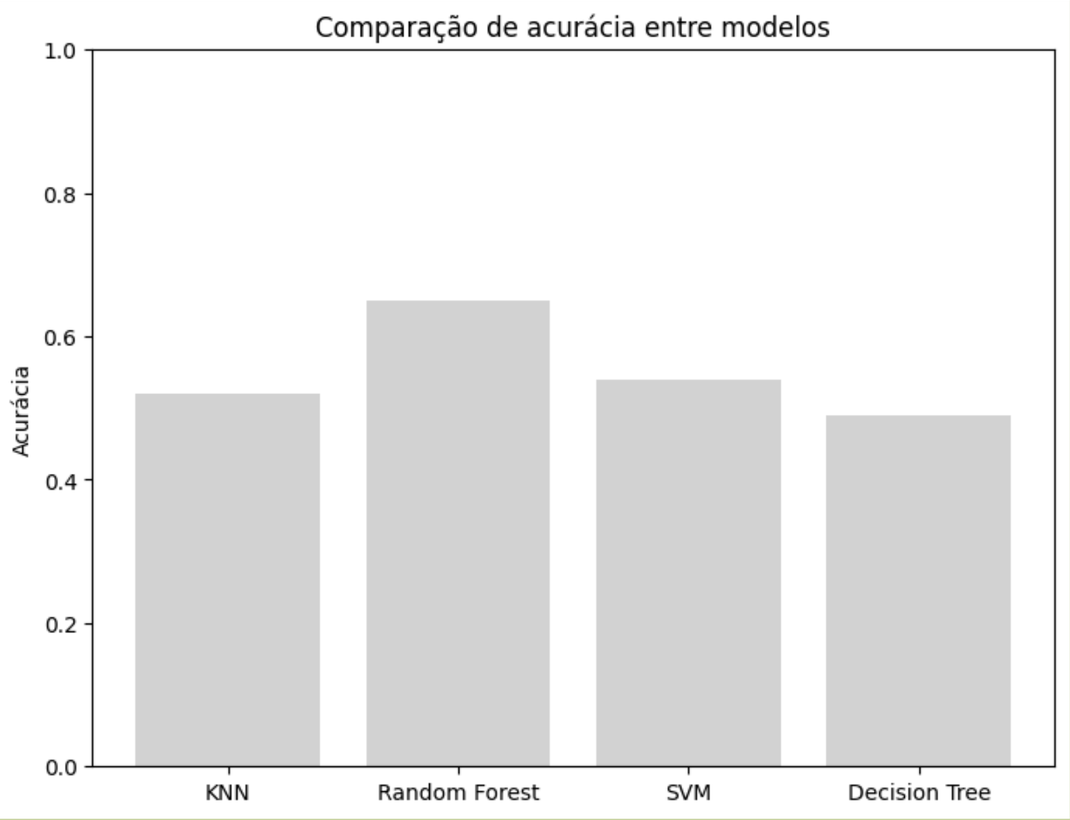

SVM - F1-score e Acurácia médios por parâmetro
f1_score
accuracy
Média
1.0
0.8
0.6
0.4
0.2
0.0
linear
poly
rbf
param_value


Decision Tree - F1-score e Acurácia médios por parâmetro
f1_score
accuracy
Média
1.0
0.8
0.6
0.4
0.2
0.0
5
10
20
param_value


Comparação de precisão entre modelos
Precisão (macro avg)
1.0
0.8
0.6
0.4
0.2
0.0
KNN
Random Forest
SVM
Decision Tree


Comparação de recall entre modelos
Recall (macro avg)
1.0
0.8
0.6
0.4
0.2
0.0
KNN
Random Forest
SVM
Decision Tree


Comparação de F1-score entre modelos
F1-score (macro avg)
1.0
0.8
0.6
0.4
0.2
0.0
KNN
Random Forest
SVM
Decision Tree


Comparação de acurácia entre modelos
Acurácia
1.0
0.8
0.6
0.4
0.2
0.0
KNN
Random Forest
SVM
Decision Tree

OBRIGADO!