Inteligencja obliczeniowa
Przetwarzanie obrazu

Jakub Ronkiewicz, 238155

22 stycznia 2019

1 Baza danych

Wybrana przeze mnie **baza** zawiera zdjęcia dotyczące dzieł sztuki, które zostały podzielone na 5 kategorii:

- drawings
- engraving
- iconography
- painting
- sculpture

2 Preprocessing

Obrazki w bazie danych miały różne rozmiary, aby móc je porównywać i klasyfikować, muszą mieć taki sam rozmiar. Wszystkim zmieniam rozmiar na 200x200. Teraz następuję wyodrębnienie ze zdjęć pewnych własności, które zostaną później przekazane do klasyfikatorów. Te globalne własności opisze poniżej.

1. **Kolor**
   Moim celem jest uzyskanie histogramu kolorów. Za pomocą specjalnej funkcji `calcHist` z paczki `cv2` dostępnej dla języka `Python` obliczam histogram. Następnie normalizuje uzyskaną listę liczb.

   ```
   def fd_histogram(image):
       image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
       hist = cv2.calcHist([image], [0, 1, 2], None, [bins, bins, bins],
       [0, 256, 0, 256, 0, 256])
       cv2.normalize(hist, hist)
   ```

2. **Kształty**
   Dąże do wydobycia danych na temat kształtów znajdujących się na obrazku, najpierw konwertuje obrazek do skali szarości, aby algorytm lepiej wyłapywał krawędzie. Następnie korzystam z funkcji **HuMoments**, która potrafi rozpoznawać kształty.

   ```
   def fd_hu_moments(image)
       image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
       feature = cv2.HuMoments(cv2.moments(image)).flatten()
       return feature
   ```

3. **Tekstura**
   Myślę, że uzyskanie danych na temat tekstury jest nieocenione przy klasyfikowaniu dzieł sztuki. Wtedy np. odróżnienie obrazu od rzeźby powinno być dużo łatwiejsze. Korzystam z paczki `mahotas`, w której znajduję się funkcja `haralick`. Funkcja ta oczekuje obrazu w skali szarości, dlatego wcześniej go konwertuje

   ```
   def fd_haralick(image):
       gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
       haralick = mahotas.features.haralick(gray).mean(axis=0)
       return haralick
   ```

Następnie za pomocą funkcji `hstack` z paczki `numpy` łączę te 3 wektory w jeden.

3 Klasyfikacja

Na początku dziele zbiór na treningowy i testowy w stosunku 90% do 10%. Zadeklarowałem listę klasyfikatorów, z których będę korzystał.

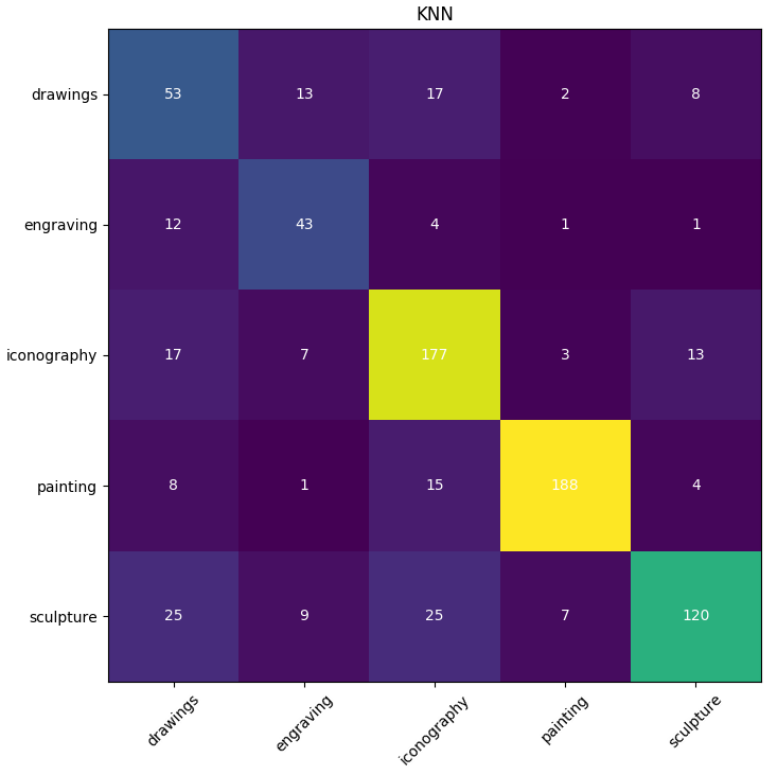
```
models = [
    ('KNN', KNeighborsClassifier()),
    ('NB', GaussianNB()),
    ('LR', LogisticRegression()),
    ('RF', RandomForestClassifier(n_estimators=num_trees, random_state=seed)),
    ('SVM', SVC(random_state=seed))
]
```

Dzięki temu za pomocą pętli wyliczam macierz błędu i dokładność dla wszystkich klasyfikatorów.

```
for name, model in models:
    classifier = model.fit(trainDataGlobal, trainLabelsGlobal)
    y_pred = classifier.predict(testDataGlobal)
    confusion_matrices[name] = confusion_matrix(testLabelsGlobal, y_pred)
    results[name] = classifier.score(testDataGlobal, testLabelsGlobal)
    draw_confusion(confusion_matrices[name], name, train_labels)
```

KNN

Sprawdźmy jak poradził sobie znany już z poprzednich zajęć klasyfikator k najbliższych sąsiadów.

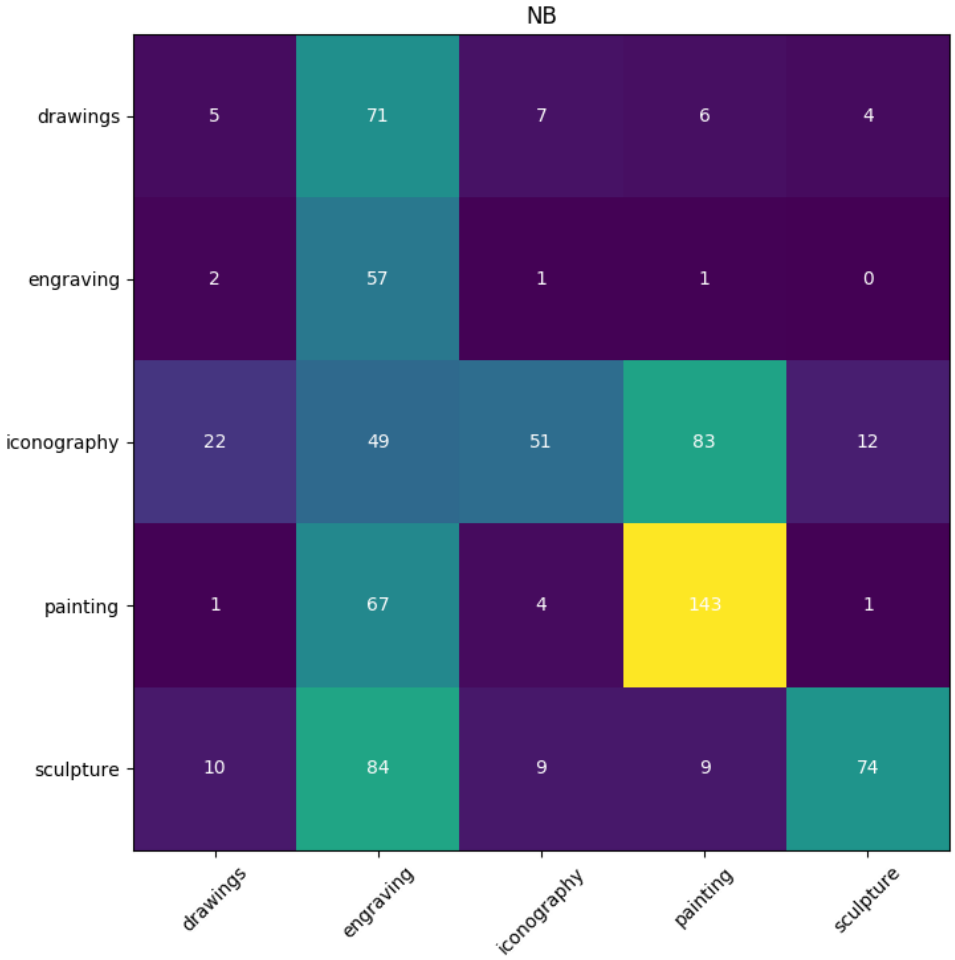


Rysunek 1: Macierz błędu dla KNN

Jak widać dosyć dobrze, najtrudniej było sklasyfikować rysunki, często były mylone z innymi kategoriami.

Gaussian Naive Bayes

Odmiana Gaussa klasyfikatora Naive Bayes.

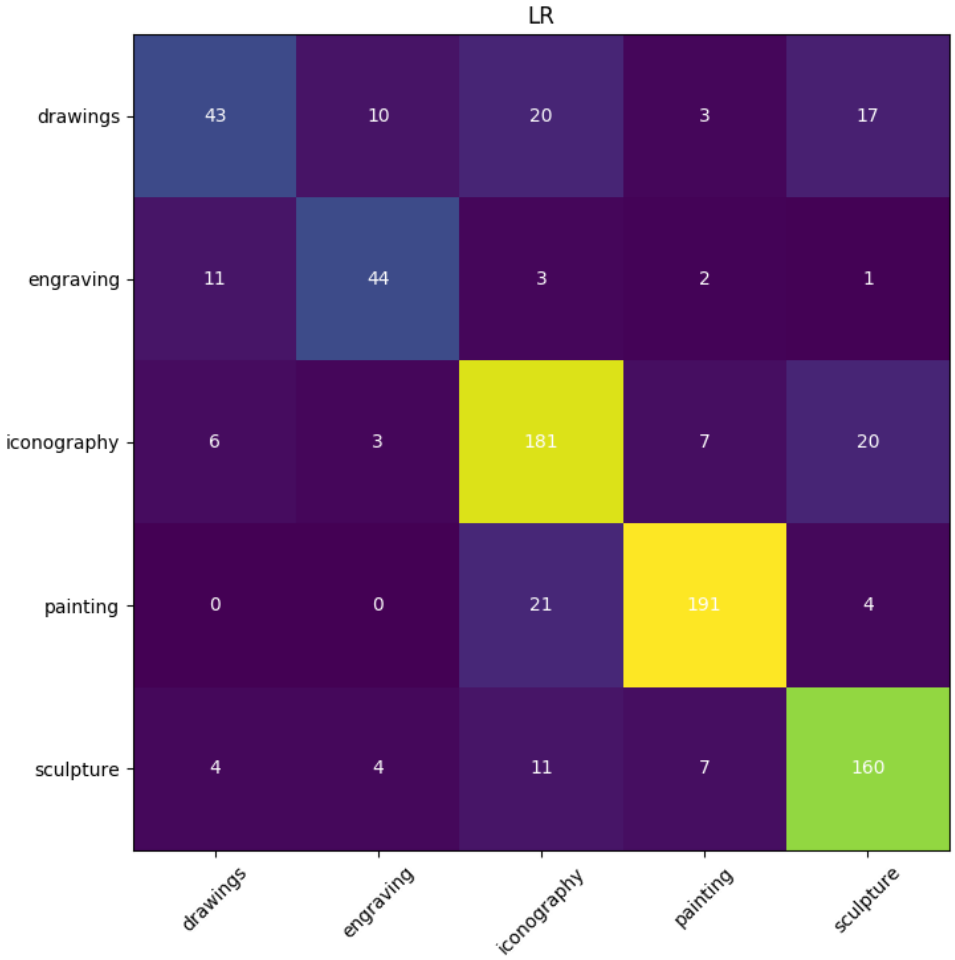


Rysunek 2: Macierz błędu dla NB

Niestety ten klasyfikator kompletnie sobie nie poradził, szczególnie z kategorią `engraving`.

Logistic Regression

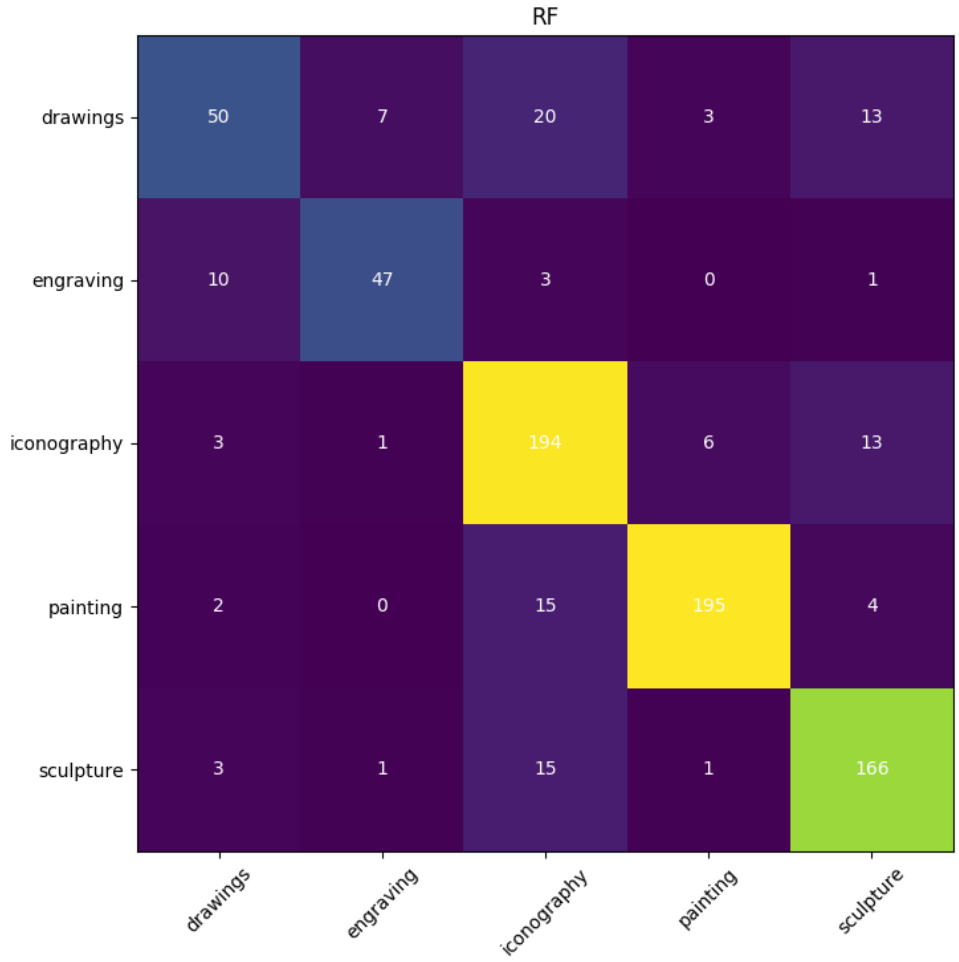
Regresja logistyczna, nie wykorzystywany przeze mnie wcześniej metoda.



Rysunek 3: Macierz błędu dla Logistic Regression

Dała bardzo dobre rezultaty, porównywalne do `KNN`.

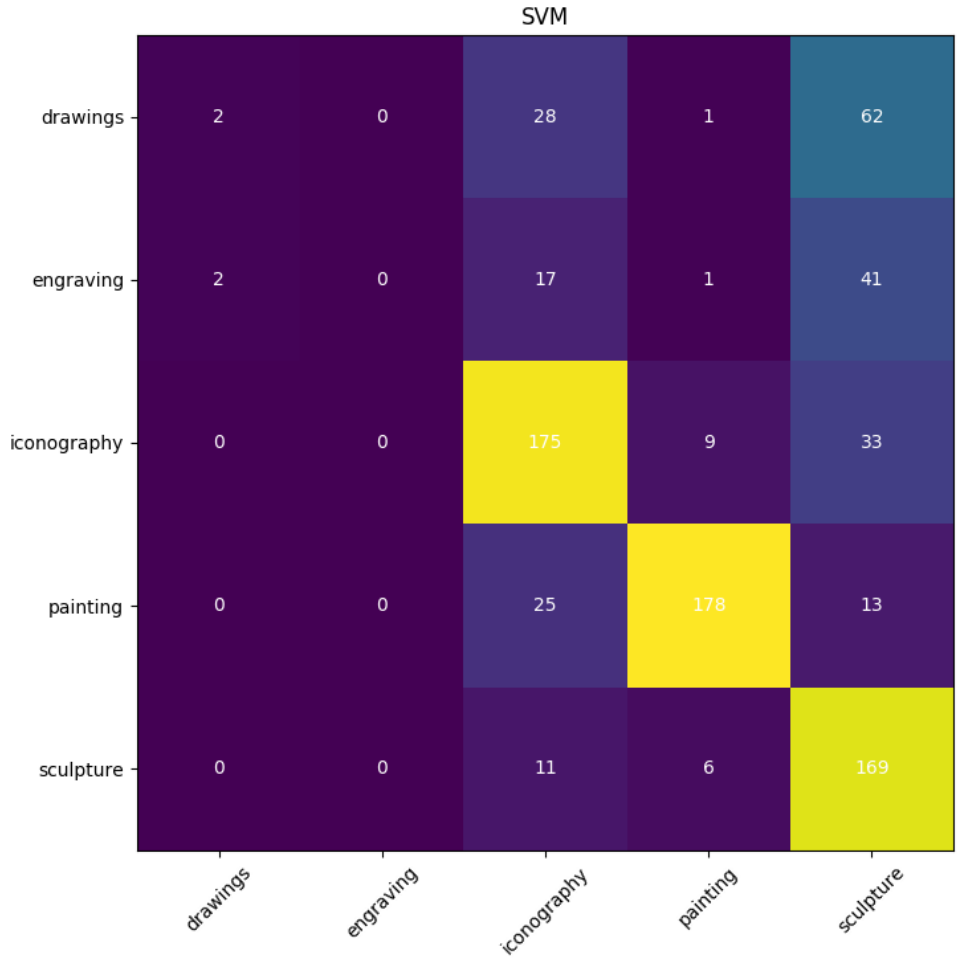
Random Forest



Rysunek 4: Macierz błędu dla Random Forest

Chyba mamy zwycięzcę, na przekątnej znajdują się najwyższe wartości z dotąd testowanych klasyfikatorów.

SVM

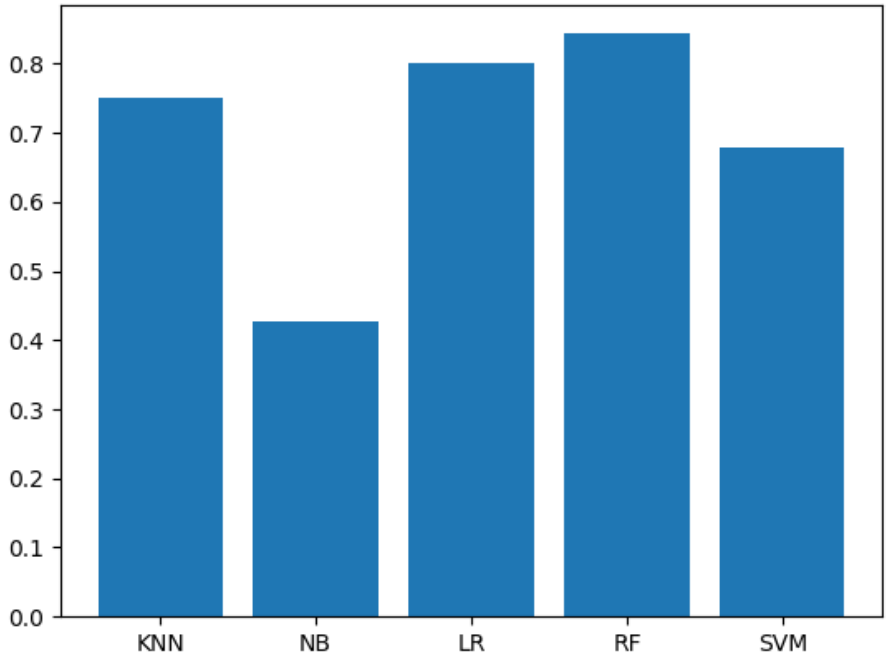


Rysunek 5: Macierz błędu dla SVM

Wszystko było by pięknie tylko kompletnie nie poradził sobie z rozpoznawaniem rysunków i drzeworytów.

4 Podsumowanie

Na podsumowanie zostawiam wykres dokładności klasyfikatorów.



Rysunek 6: Dokładność klasyfikatorów

`Random Forest` okazał się najlepszym klasyfikatorem, a najgorzej wypadł `Naive Bayes`. Kod źródłowy dostępny **tutaj.**