Computer vision in the new era of Artificial Intelligence and Deep Learning

Visión por computador en la nueva era de la Inteligencia Artificial y el Deep Learning

Rubén Usamentiaga*, Alberto Fernández°
*** University of Oviedo**
° TSK

Gijón (Spain)
5 – 16 April 2021

https://github.com/albertofernandezvillan/computer-vision-and-deep-learning-course

Scikit-learn

Introducing scikit-learn for classification, regression and clustering

- scikit_learn_introduction_classification.ipynb
- scikit_learn_introduction_regression.ipynb
- k_means_clustering_sklearn.ipynb

Open in Colab

- scikit_learn_introduction_classification.ipynb
- scikit_learn_introduction_regression.ipynb
- k_means_clustering_sklearn.ipynb

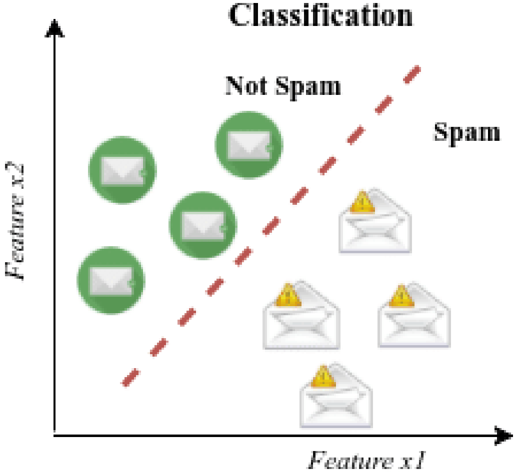
https://github.com/albertofernandezvillan/computer-vision-and-deep-learning-course

supervised vs unsupervised learning

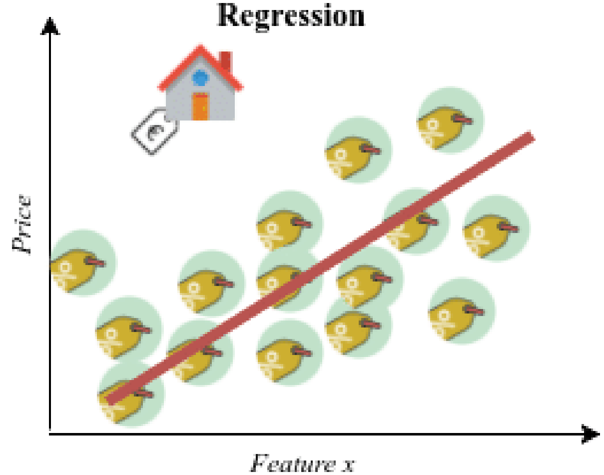
Supervised

Unsupervised

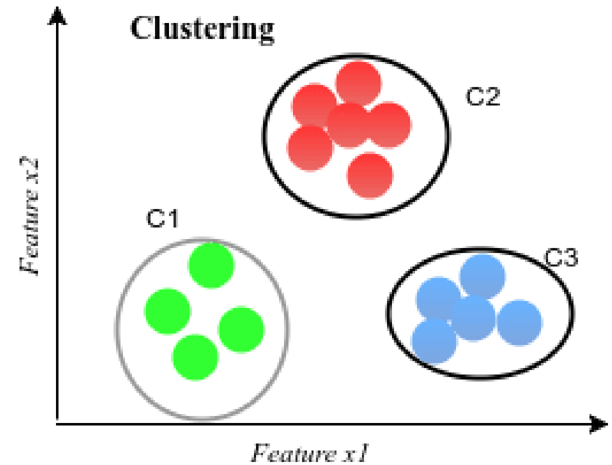
Spam filtering



House price estimation



Customers clustering



Classification: predicting a label
Classification uses supervised learning techniques to find the relationship between the features and the assigned label (e.g. spam/no spam)

Regression: predicting a quantity
Regression uses supervised learning techniques to learn a mapping function from the features to a continuous output variable. A regression problem requires the prediction of a quantity. All houses have a price.

Clustering: assigning a cluster
Customers are grouped into different categories based on their purchasing behaviour, but the unsupervised learning algorithm has no information about the labels (or classes) associated with each sample

Classification vs regression

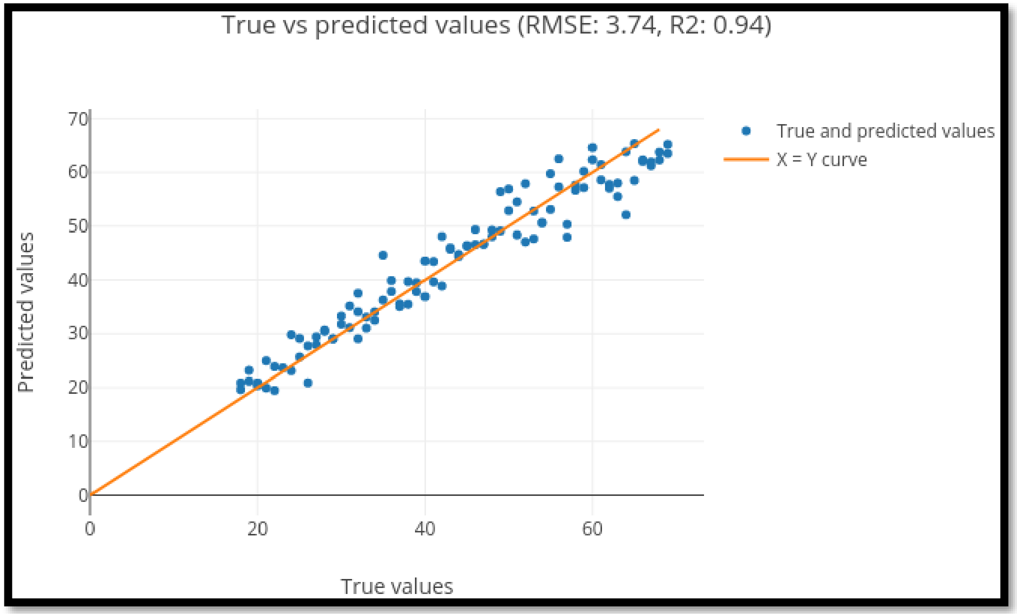
In some cases, it is possible to convert a regression problem to a classification problem. For example, the age to be predicted could be converted into discrete buckets. Age in a continuous range between [0 and 100] could be converted into:

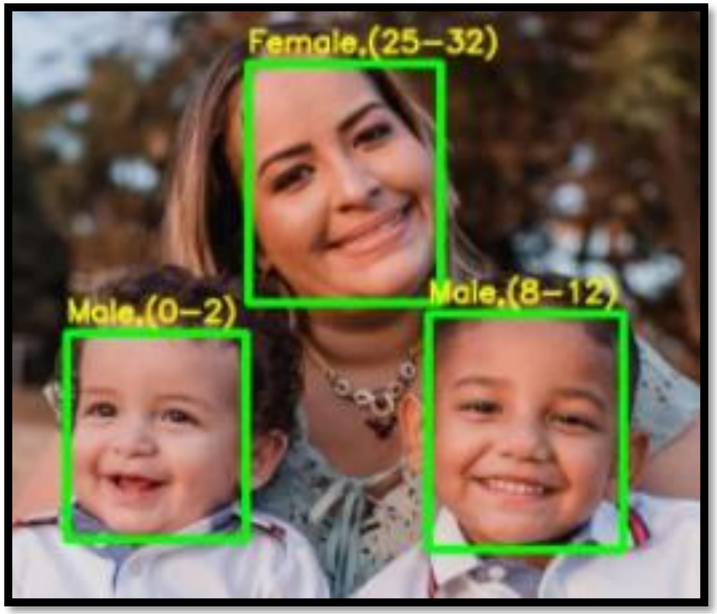
Class 0: 0 - 2

Class 1: 3 - 8

Class 2: 8 – 12

….

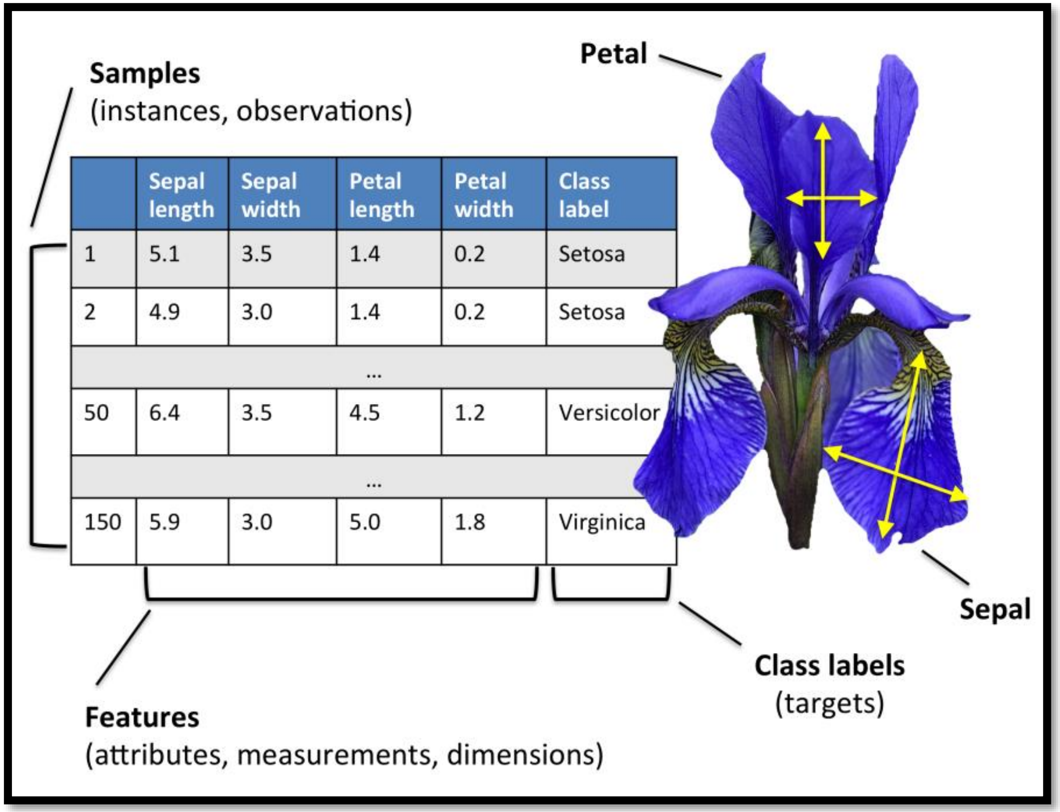




Scikit-learn for classification: dataset

```
from sklearn.datasets import load_iris

iris = load_iris()
x = iris.data
y = iris.target
feature_names = iris.feature_names
target_names = iris.target_names
```

```
x.shape: (150,4)
y.shape: (150,1)
```



	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

```
Feature names: '['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal
width (cm)']'
Target names: '['setosa' 'versicolor' 'virginica']'
```

As we have three classes, this is a three-class classification problem, because our task here is to classify each sample in one of the three classes (the species of Iris).

Scikit-learn for classification: training

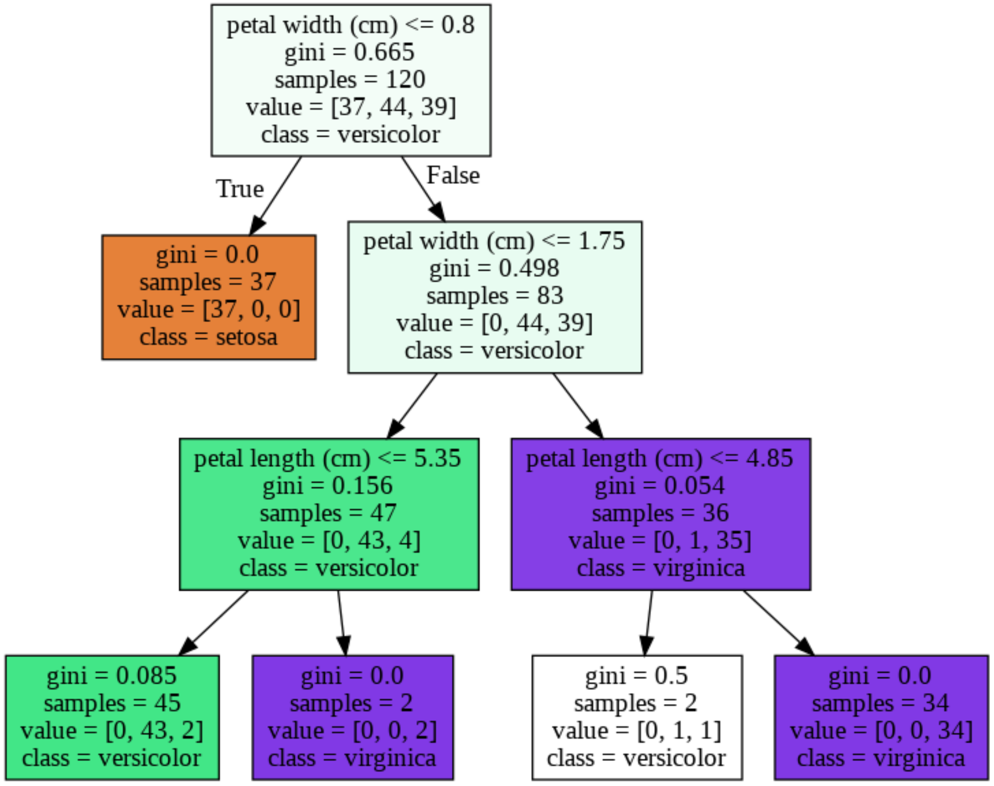
Split training and test sets

```
from sklearn.model_selection import train_test_split

# Split the dataset for training and testing:
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=123)
```

```
x train shape: '(120, 4)'
x test shape: '(30, 4)'
```

Training (fit)

```
from sklearn import tree

# Create the decision tree classifier
tree_classifier=tree.DecisionTreeClas
sifier(random_state=0, max_depth=3)

# We can train the model with fit
function:
tree_classifier.fit(x_train,y_train)
```

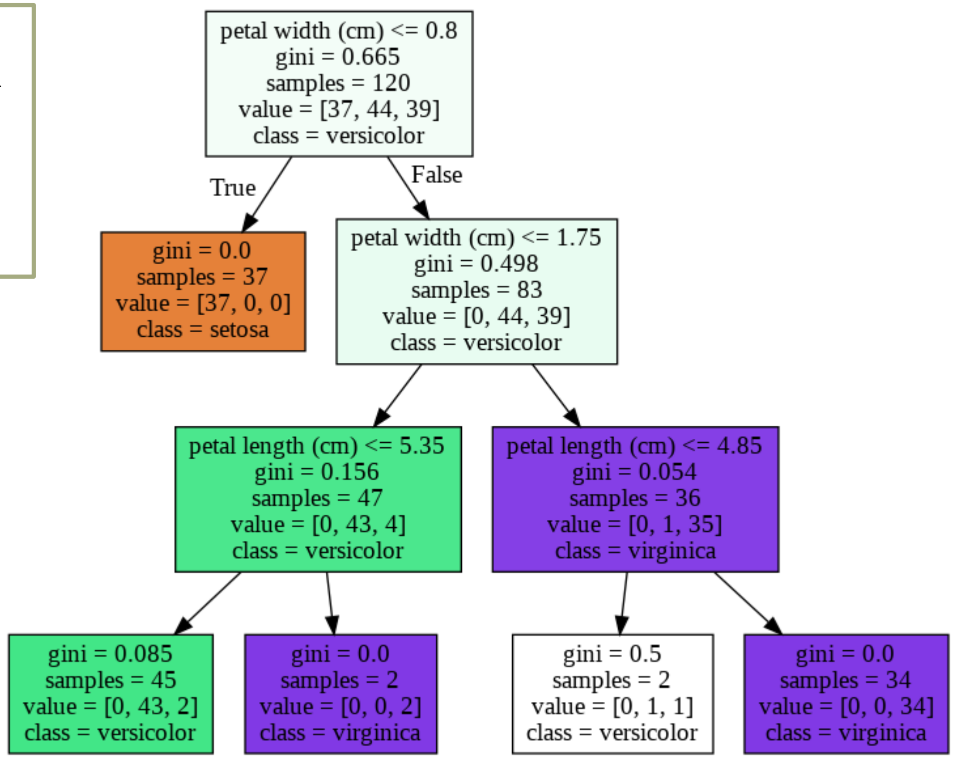


Making predictions using the trained model

```
sepal length of 5 cm
sepal width of 5 cm
petal length of 5 cm
petal width of 0.7999 cm.
```

```
new_sample = np.array([[5, 5, 5, 0.7999]])
```

```
# Predict the class for this new sample:
predicted_class = tree_classifier.predict(n
ew_sample)

pred_class = predicted_class[0]
class_name = iris.target_names[pred_class]
```



Model persistence

```
from joblib import dump

dump(tree_classifier,
'iris_tree_classifier.joblib')
```

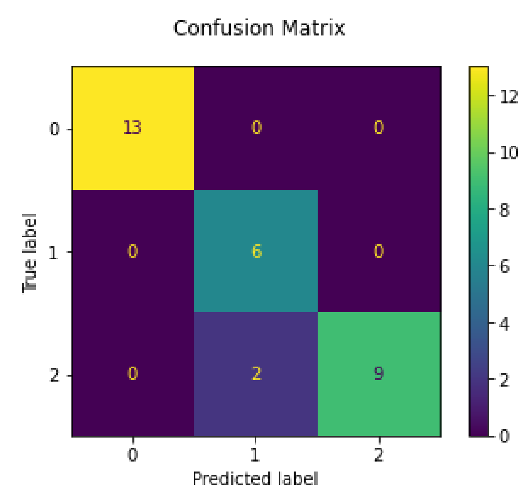
```
from joblib import load

tree_classifier_iris =
load('iris_tree_classifier.joblib')
```

Measuring the accuracy of the trained model

```
# 1. After training, the model is ready to make predictions:
tree_predictions = tree_classifier.predict(x_test)

# 2. Get the comparisons (True/False) for each one:
comparisons = (tree_predictions == y_test)
print("comparisons array: \n '{}'".format(comparisons))

# 3. Calculate the accuracy using np.mean():
my_accuracy = np.mean(comparisons)
print("The accuracy of the model is: '{}'".format(my_accuracy))
```

```
import sklearn.metrics as metrics

metrics.accuracy_score(y_test, tree_predictions)
metrics.classification_report(y_test, tree_predictions)
metrics.plot_confusion_matrix(tree_classifier, x_test, y_test)
```

```
Classification report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        13
           1       0.75      1.00      0.86         6
           2       1.00      0.82      0.90        11

    accuracy                           0.93        30
   macro avg       0.92      0.94      0.92        30
weighted avg       0.95      0.93      0.93        30
```



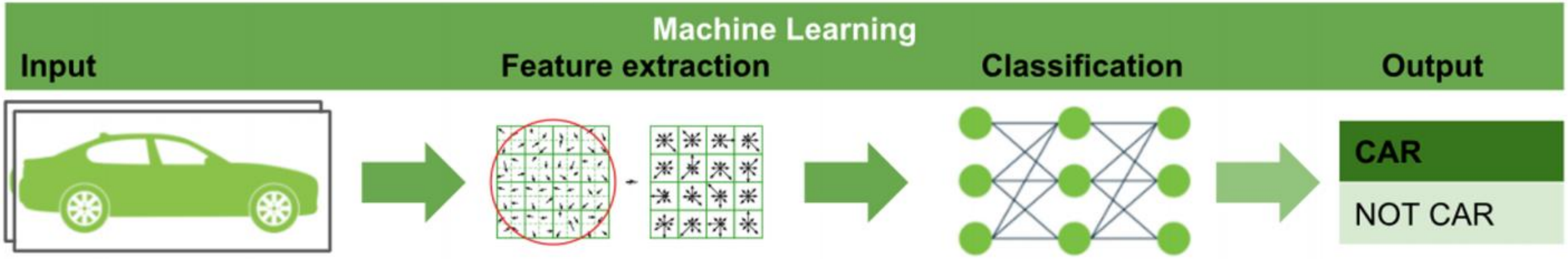
The importance of features in Machine Learning

Feature names: '['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']'
Target names: '['setosa' 'versicolor' 'virginica']'

- Based on the obtained results we can conclude that 4 features are enough to describe each sample
- This leads to the following question:
  - How to find “good features” describing my objects to be classified?

The importance of features in Machine Learning

- **Feature engineering** is the process of using domain knowledge to extract features from raw data. These features can be used to improve the performance of machine learning algorithms



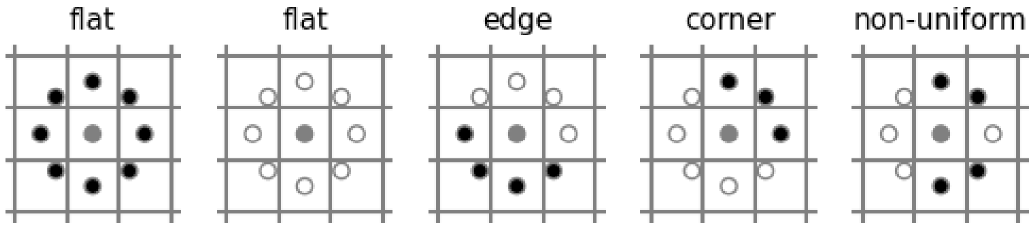
Example of feature extractors in computer vision

- Local binary patterns (LBP) is a feature extraction algorithm.
- LBP is a type of visual descriptor used for classification in computer vision
- The histogram of oriented gradients (HOG) is a feature descriptor used in computer vision and image processing for the purpose of object detection

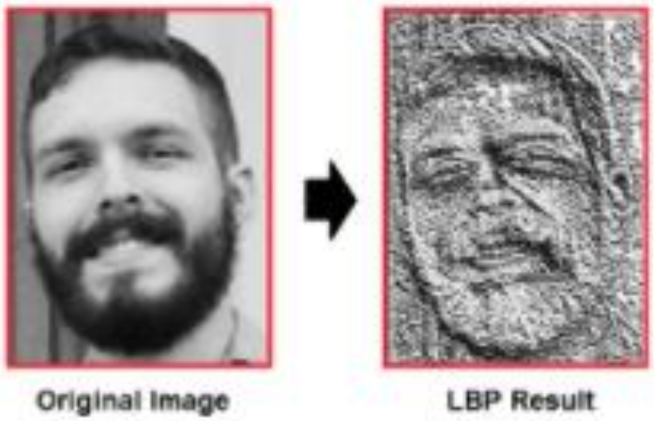
Local Binary Pattern (LBP)

https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_local_binary_pattern.html

LBP looks at points surrounding a central point and tests whether the surrounding points are greater than or less than the central point (i.e. gives a binary result)

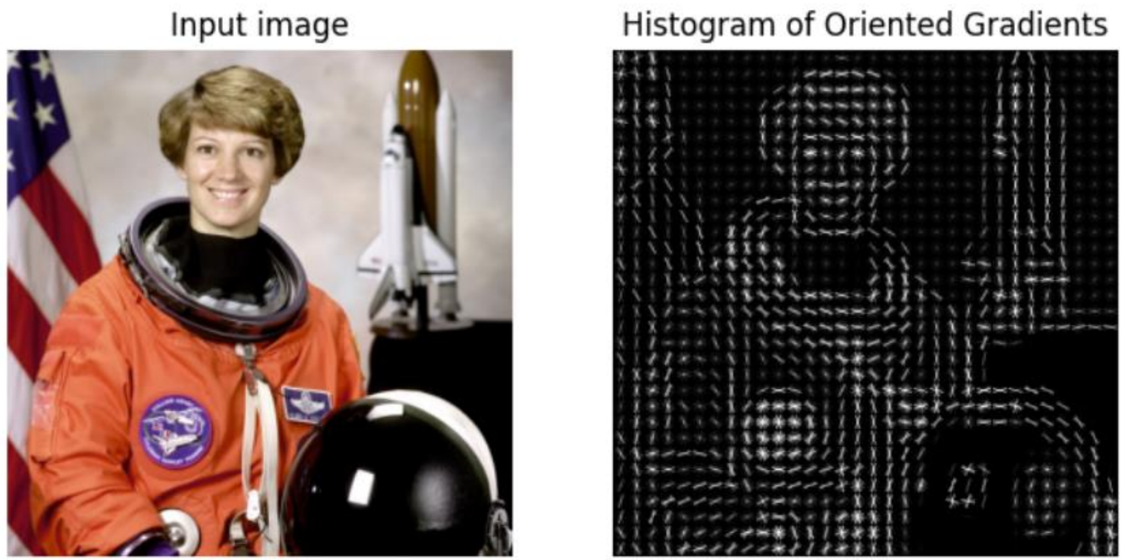



```
radius = 3
n_points = 8 * radius
METHOD = 'uniform`

lbp = local_binary_pattern(image, n_points, radius, METHOD)
```



Histogram of oriented gradients (HOG)

https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_hog.html




```
from skimage.feature import hog

fd, hog_image = hog(image, orientations=8, pixels_per_cell=(16, 16),
                    cells_per_block=(1, 1), visualize=True, multichannel=True)
```

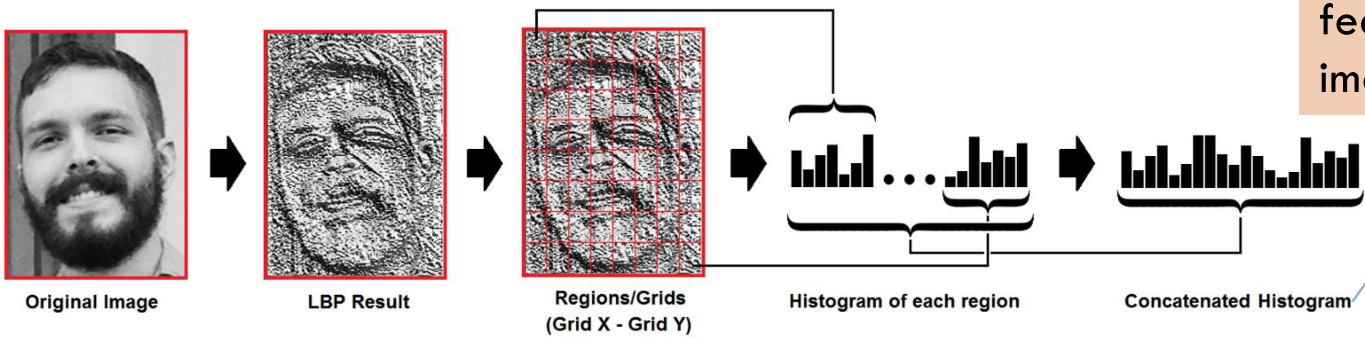
(n_blocks_row, n_blocks_col, n_cells_row, n_cells_col, n_orient) ndarray

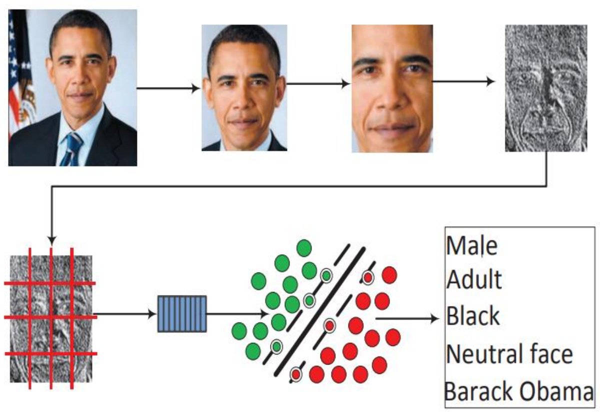
In this case: **8192 features**

Example: using LBP for face encoding

For efficient face representation, extracted features should retain also spatial information. Hence, the facial image is divided into m regions

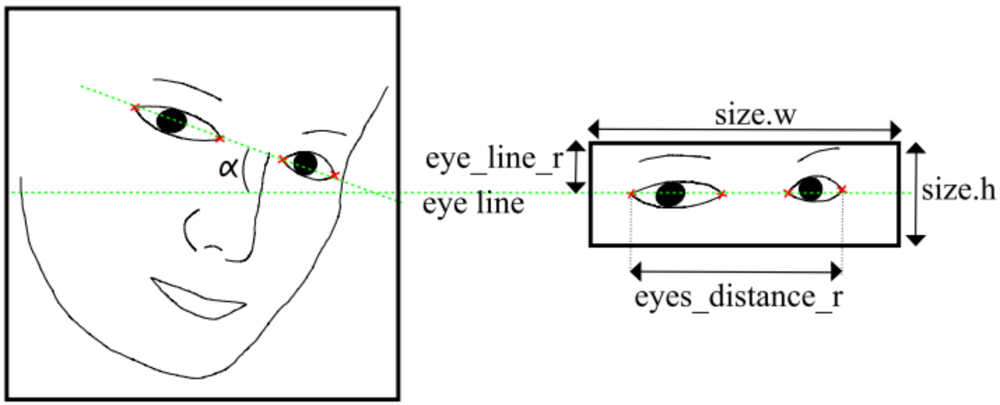
features for each image





- Each concatenated histogram (the features) is labelled according the purpose of the classifier:
  - For gender recognition: male (0)/female (1)
  - For emotion recognition: neutral (0)/happy (1)/sad (2)/…
- Therefore we have N samples in our dataset:
  - Each sample corresponds to an image
  - Each sample has features (histogram) and a label (class)
- Finally, a classifier (SVM, KNN,…) is trained/tested using all the training/testing instances

Example for glasses detection



This region is encoded using 56 values (if uniform LBP and a neighborhood of 8 pixels)

Uniform LBP patterns produces M values (labels):

- M=59 labels for a neighborhood of 8 pixels and produces M=256 labels for standard LBP. Additionally, for the 16 neighborhoods, the numbers are M=243 and M=65,536, respectively.

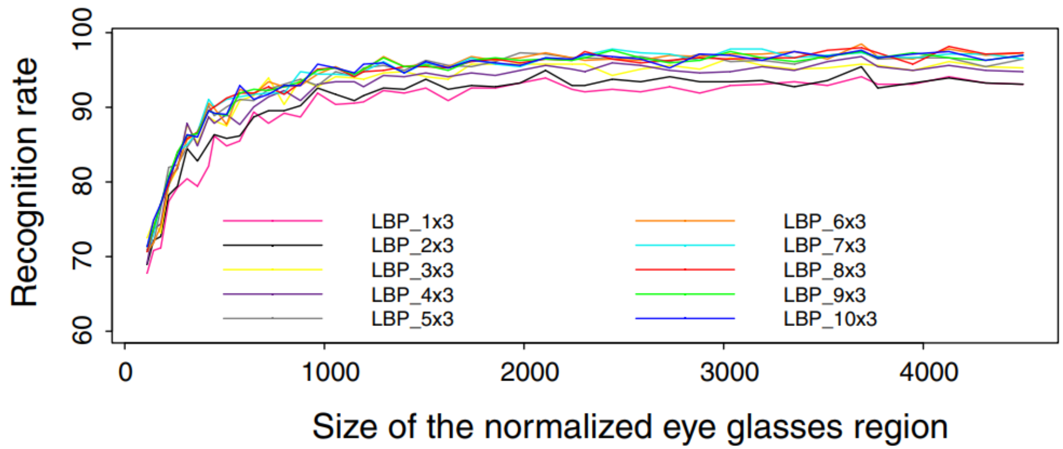
The total length of the histogram will be **XxYxM,** where X and Y are the number of division in both the width and the height of the input image (to retain also spatial information)

Example for glasses detection

Fernández, A., García, R., Usamentiaga, R., & Casado, R. (2015). Glasses detection on real images based on robust alignment. Machine Vision and Applications, 26(4), 519-531.

	1	2	3	4	5	6	7	8	9	10
1	88.36	88.53	93.42	91.57	93.76	94.10	93.59	94.60	93.09	94.77
2	91.23	91.40	93.76	92.58	94.10	94.94	94.27	94.94	94.44	94.60
3	91.74	93.59	95.11	94.77	96.96	96.80	96.29	96.63	96.29	97.13
4	92.41	93.25	95.45	95.11	95.78	95.78	95.78	96.96	96.46	96.29
5	92.92	93.59	95.45	94.94	96.46	96.80	96.80	97.30	96.29	97.64
6	93.59	93.59	96.29	95.28	96.63	97.13	96.63	97.64	96.12	96.46
7	93.42	94.10	96.12	95.28	96.80	96.96	96.80	96.80	96.80	97.30
8	92.75	94.60	96.29	94.60	96.80	96.63	96.63	96.46	96.63	96.63
9	93.09	94.27	95.95	95.28	96.29	96.63	96.46	95.95	96.46	96.46
10	92.92	93.09	95.95	94.60	96.63	97.13	95.78	96.12	96.80	96.80

Recognition rates as a function of the number of regions

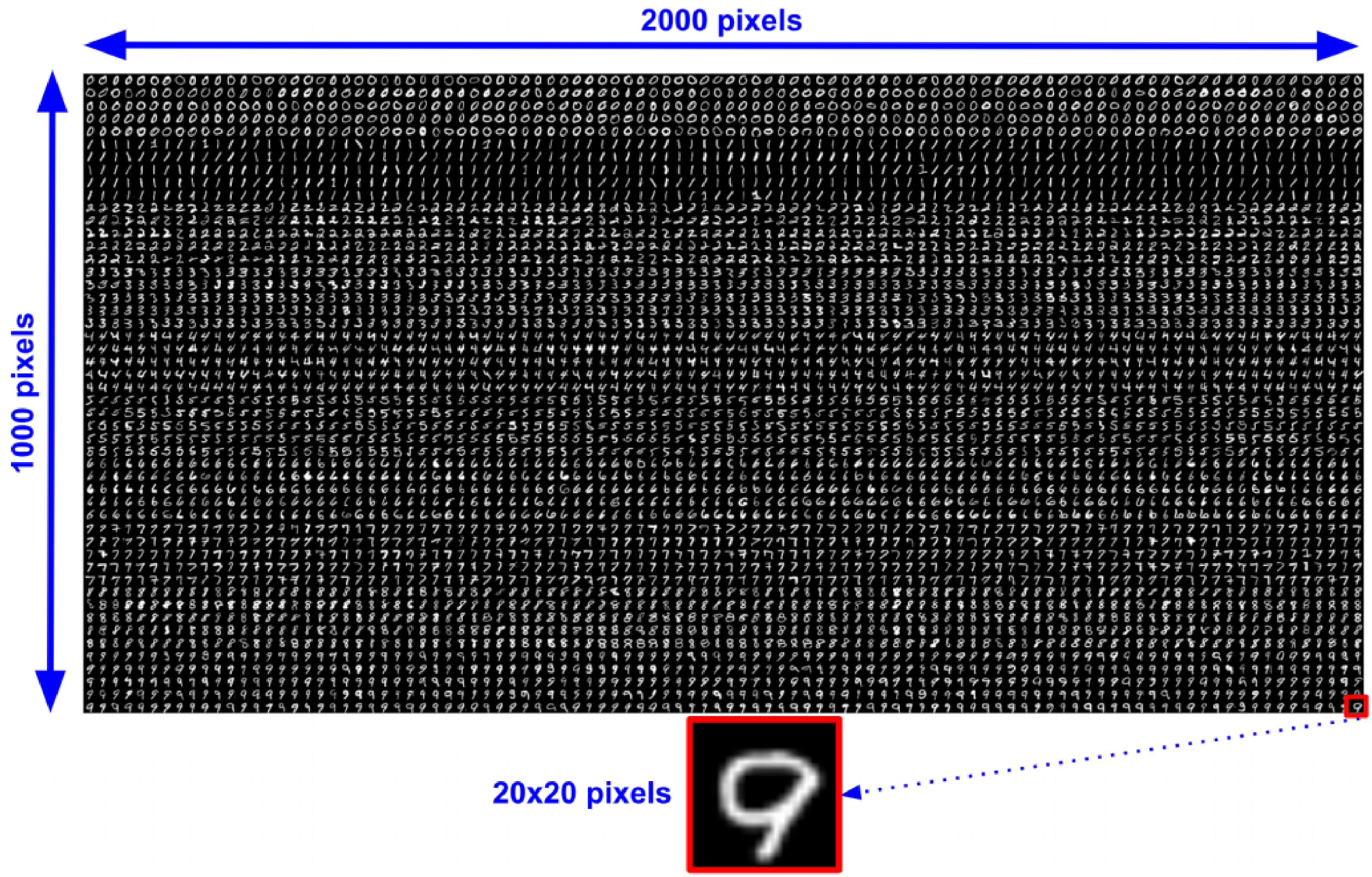


Results varying from the maximum to the minimum size of the normalized eye glasses region

When looking for the optimal number of regions, it is observed that the changes in the number of regions may cause big difference in the length of the feature vector, but the performance is not necessarily affected significantly.

HOG for handwritten digits recognition

Dataset: https://github.com/PacktPublishing/Mastering-OpenCV-4-with-Python/blob/master/Chapter10/01-chapter-content/digits.png



HOG for handwritten digits recognition

Handwritten digits recognition using SVM and HoG features:
https://github.com/PacktPublishing/Mastering-OpenCV-4-with-Python/blob/master/Chapter10/01-chapter-content/svm_handwritten_digits_recognition_preprocessing_hog.py

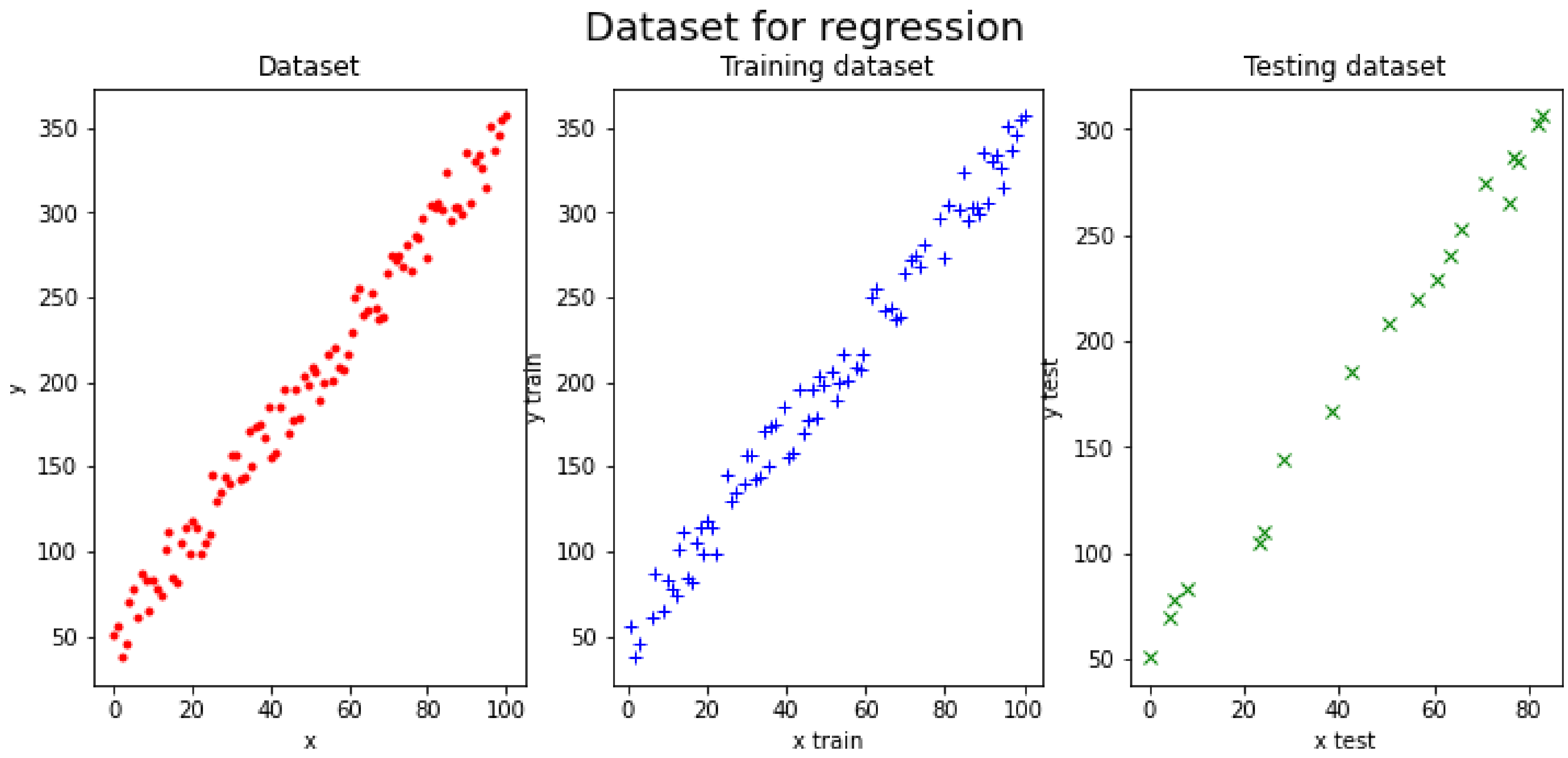
Handwritten digits recognition using SVM and HoG features with pre-processing of the images. A grid-search on C and gamma is also carried out.
https://github.com/PacktPublishing/Mastering-OpenCV-4-with-Python/blob/master/Chapter10/01-chapter-content/svm_handwritten_digits_recognition_preprocessing_hog_c_gamma.py

Handwritten digits recognition using KNN and HoG features and varying both k and the number of training/testing images with pre-processing of the images
https://github.com/PacktPublishing/Mastering-OpenCV-4-with-Python/blob/master/Chapter10/01-chapter-content/knn_handwritten_digits_recognition_k_training_testing_preprocessing_hog.py

Pre-processing = ***deskew*** (in this specific case)

Note: Indicated examples use OpenCV Machine Learning module (**cv2.ml**) and not scikit-learn

Scikit-learn for regression: dataset



Scikit-learn for regression: training and testing

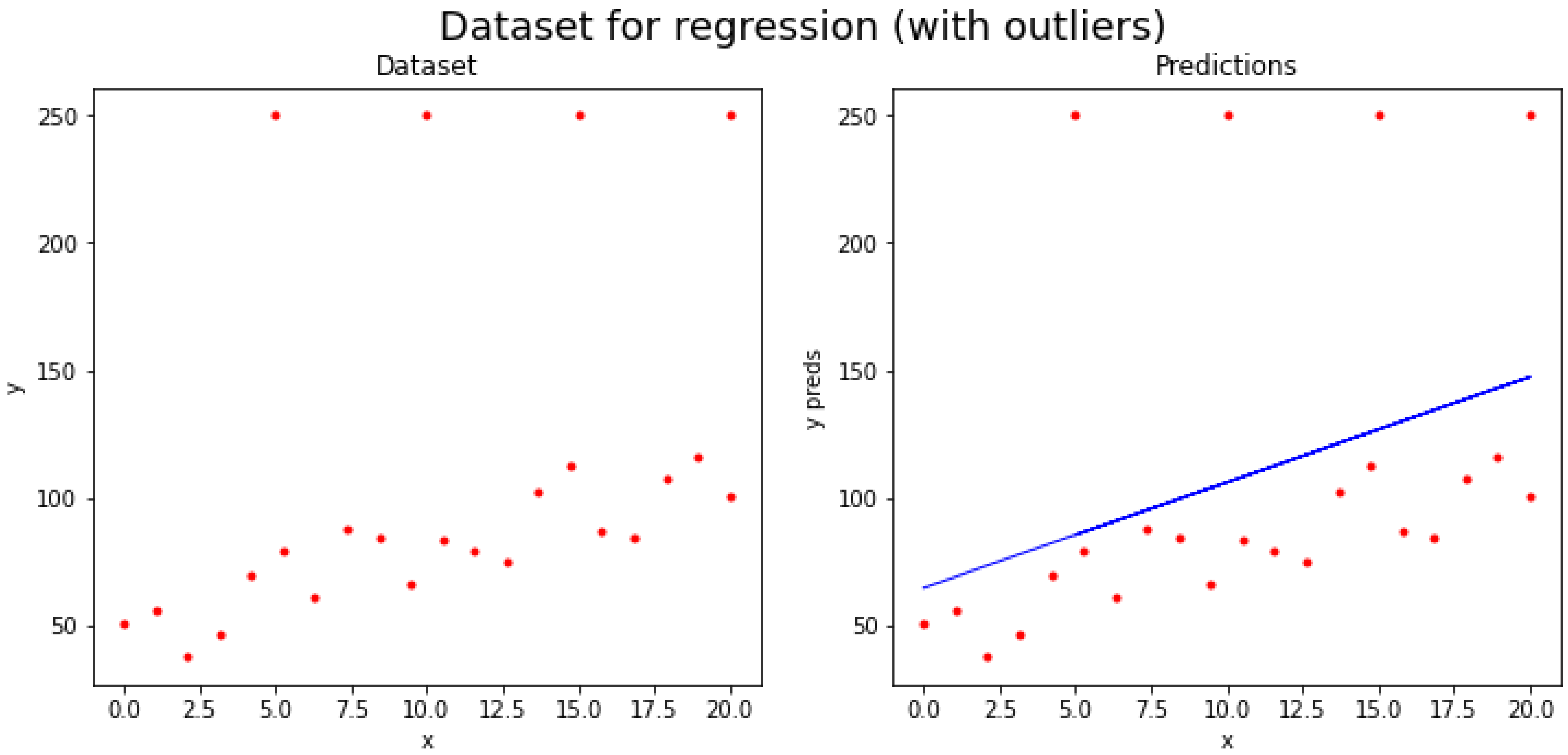
```
from sklearn import linear_model

# Create linear regression object:
model = linear_model.LinearRegression()

# Train the model using the training sets:
model.fit(x_train, y_train)

# The coefficients and the intercept factor:
print("Coefficients: {}".format(model.coef_))
print("Intercept: {}".format(model.intercept_))
```

```
Coefficients: [[2.99811861]]
Intercept: [49.14094108]
```

```
from sklearn.metrics import mean_squ
ared_error, r2_score

# Make predictions using the testing
 set
y_preds = model.predict(x_test)

# Model score:
model.score(x_test, y_test))

# The mean squared error (mse):
mean_squared_error(y_test, y_preds)

r2_score(y_test, y_preds)
```

Scikit-learn for regression: dataset with outliers

linear_model.LinearRegression estimator is heavily influenced by the outliers

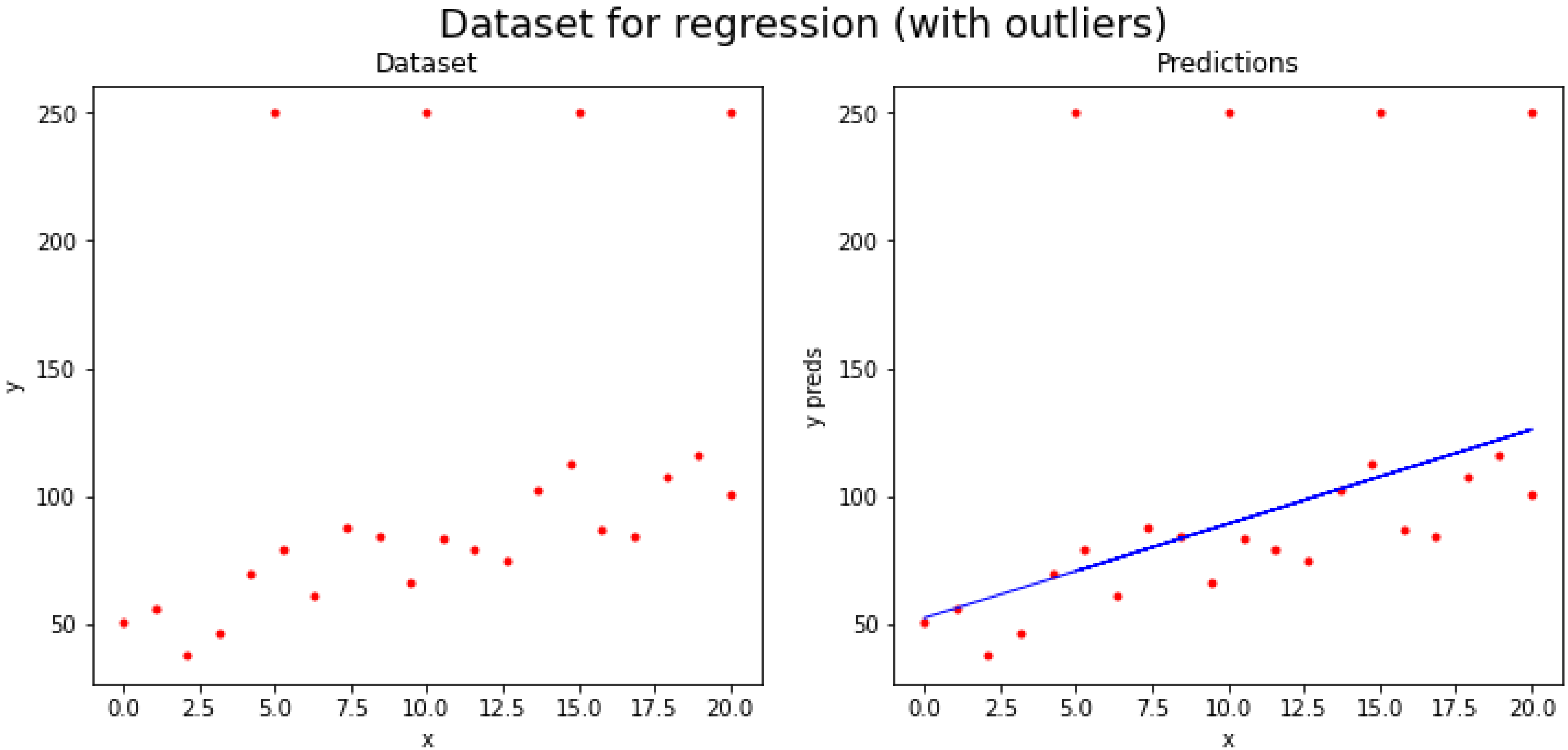


Scikit-learn for regression: dataset with outliers

linear_model.HuberRegressor is robust to outliers

The parameter epsilon controls the number of samples that should be classified as outliers. The smaller the epsilon, the more robust it is to outliers. greater than 1.0, default=1.35

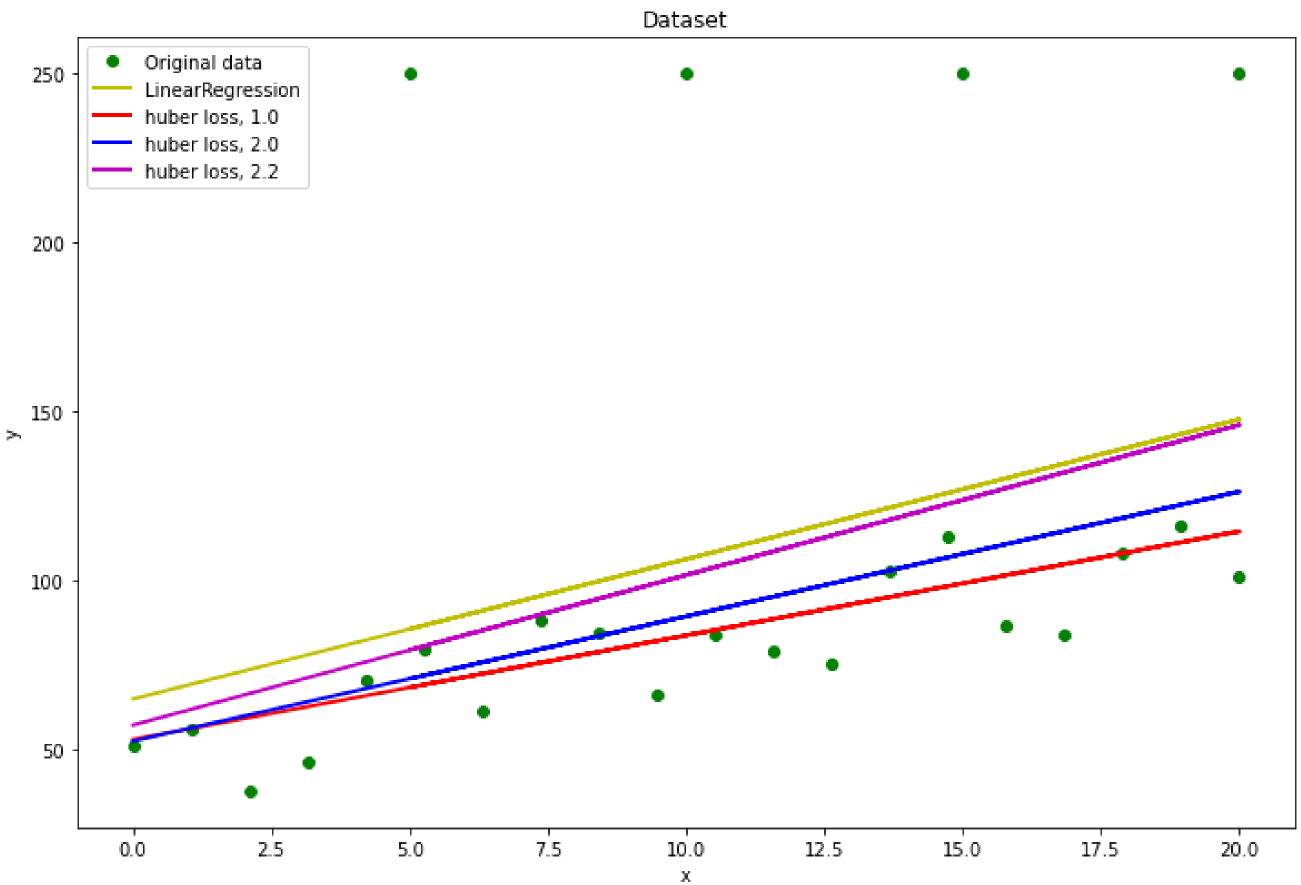
HuberRegressor(epsilon=2.0)



Scikit-learn for regression: dataset with outliers

HuberRegressor estimator is less influenced by the outliers. Moreover, as the parameter epsilon is increased for the HuberRegressor estimator, the obtained results get closer to the results obtained with the LinearRegression estimator.

Dataset for regression (with outliers)

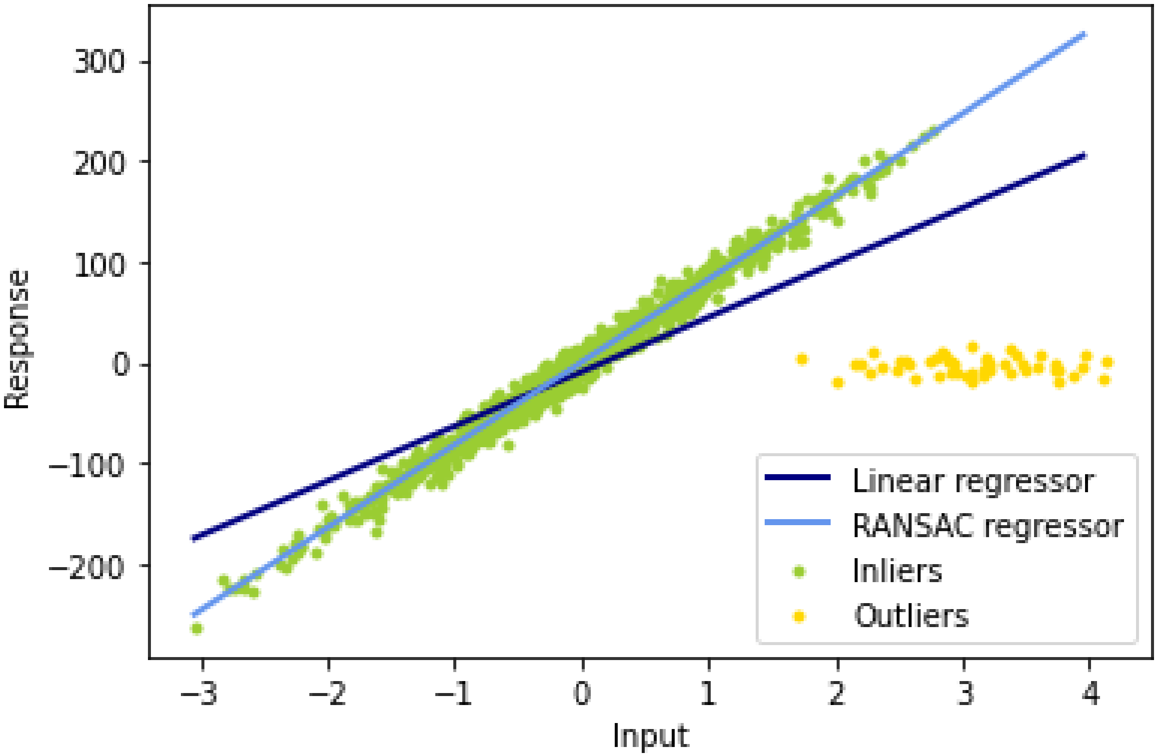


Scikit-learn for regression: dataset with outliers

Robust linear model estimation using RANSAC

```
# Robustly fit linear model with RANSAC algorithm
ransac = linear_model.RANSACRegressor()
ransac.fit(X, y)
inlier_mask = ransac.inlier_mask_
outlier_mask = np.logical_not(inlier_mask)
```

Boolean mask of inliers classified as True.




```
plt.scatter(X[inlier_mask],
y[inlier_mask],
color='yellowgreen',
marker='.', label='Inliers')
```

```
plt.scatter(X[outlier_mask],
y[outlier_mask],
color='gold', marker='.',
label='Outliers')
```

K-Means clustering for color quantization in scikit-learn

```
img = cv2.imread("landscape.jpg")

img = img.reshape((img.shape[1]*img.shape[0], 3))

kmeans = KMeans(n_clusters=NUMBER_CLUSTERS)
kmeans.fit(img)

centroids = kmeans.cluster_centers_
centroids = np.uint8(centroids)
labels = kmeans.labels_

result = centroids[labels]
result = result.reshape(img_shape)
```

```
[[ 17  44 125]
 [ 23  50 131]
 [ 31  59 143]
 [ 42  70 154]
 [ 13  42 127]
 [ 14  43 128]
 [  9  39 126]
 [ 10  40 127]
 [  7  39 128]
 [  7  39 128]]
```

```
0 0 0 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
3 3 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

[1 2 7]

[8 199 246]

[11 41 122]

[9 82 185]

Centroids are in the form of BGR triplets

Scikit-learn

Introducing scikit-learn for classification, regression and clustering

https://github.com/albertofernandezvillan/computer-vision-and-deep-learning-course