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

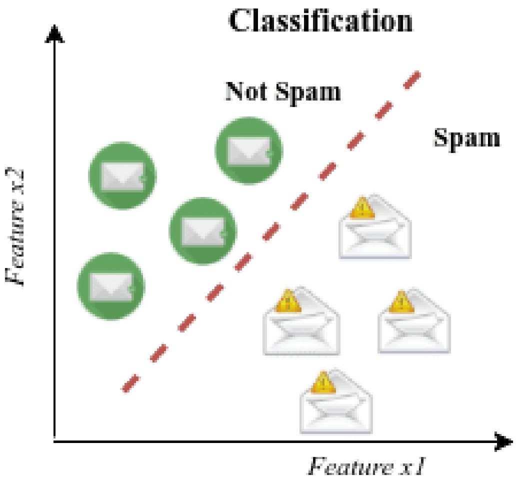
- scikit_learn_introduction_classification.ipynb
- scikit_learn_introduction_regression.ipynb
- k_means_clustering_sklearn.ipynb

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

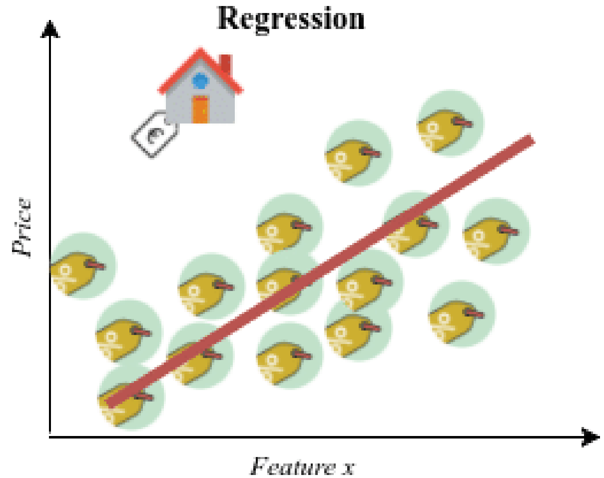
supervised vs unsupervised learning

Supervised

Spam filtering

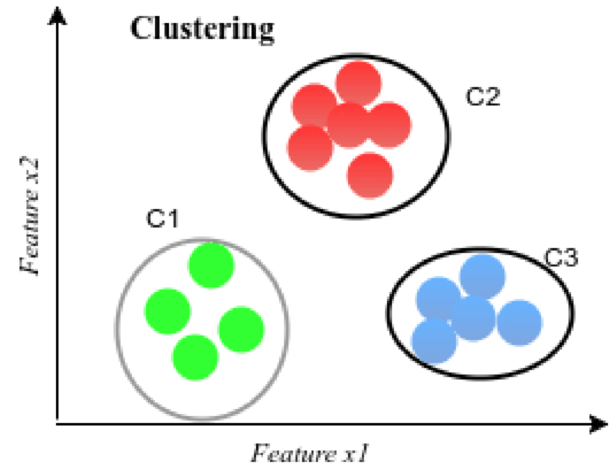


House price estimation



Unsupervised

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

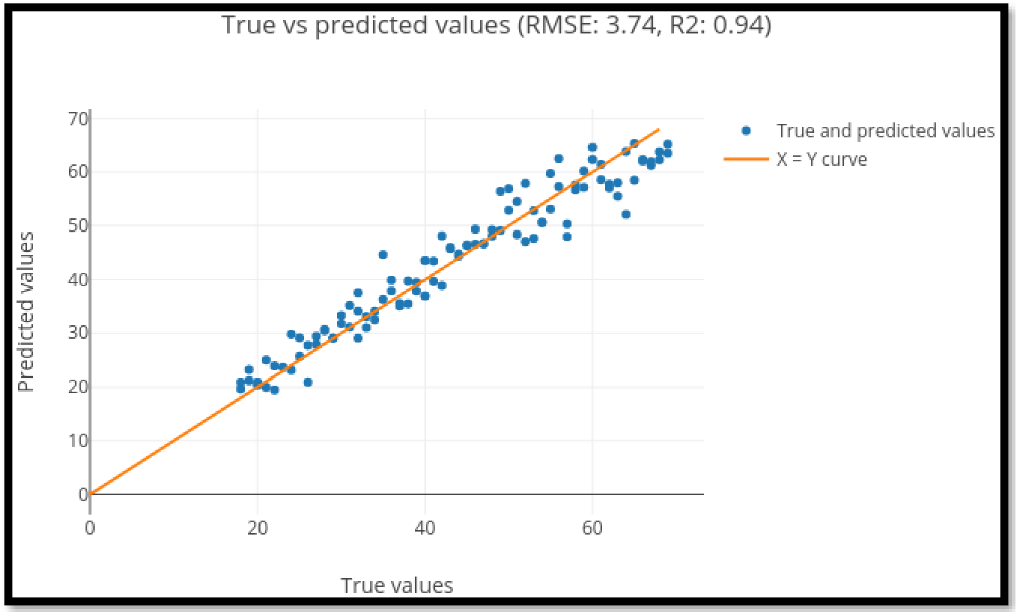
In some cases, it is possible to convert a regression problem to a classification problem. For example, the age to be predicted could be converted into discrete buckets. Age in a continuous range between [0 and 100] could be converted into:

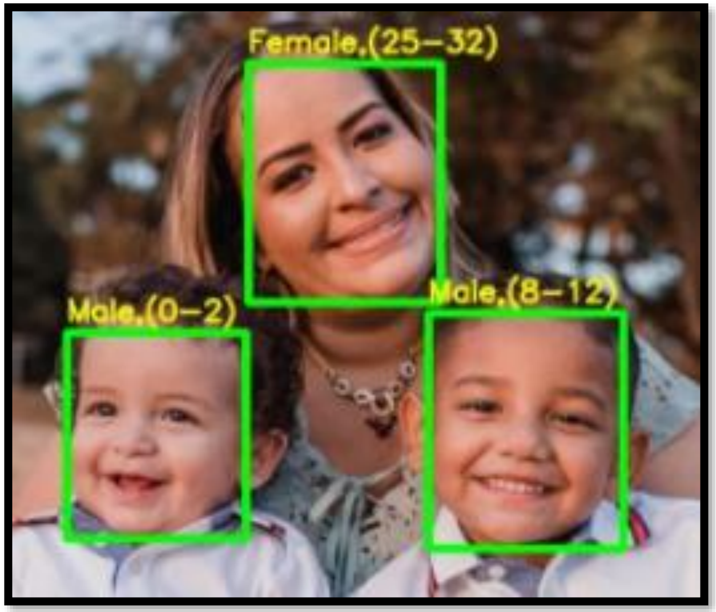
Class 0: 0 - 2

Class 1: 3 - 8

Class 2: 8 – 12

….

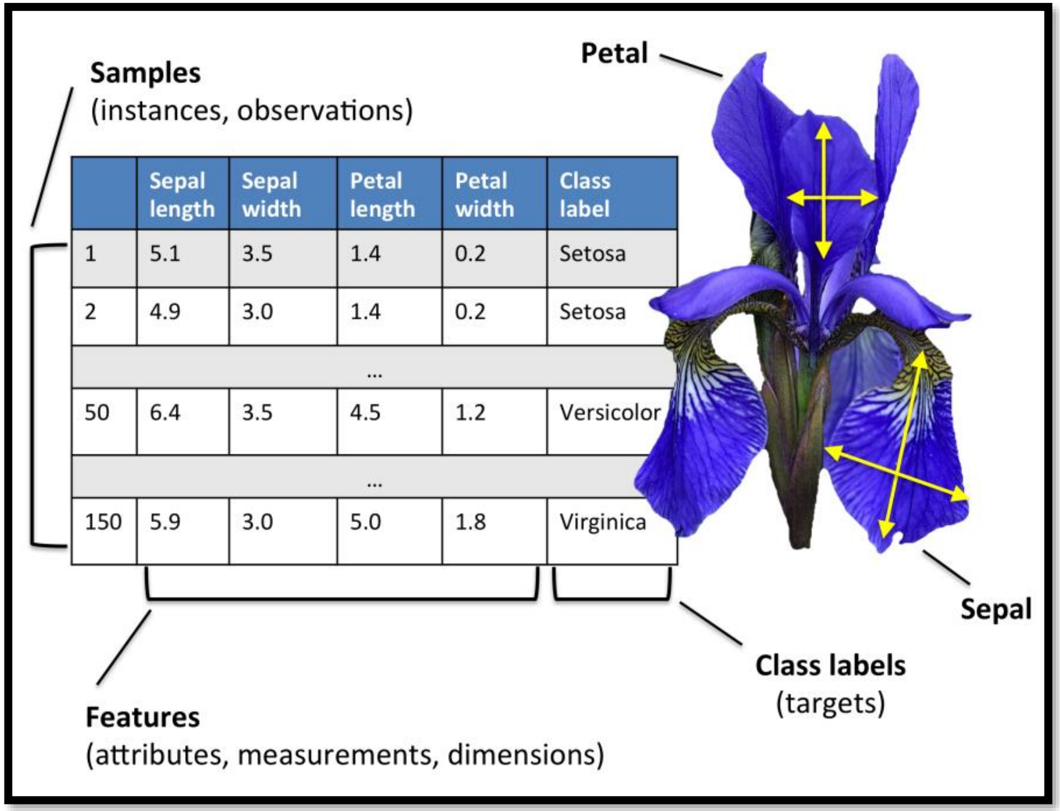




Scikit-learn for classification: dataset

```
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

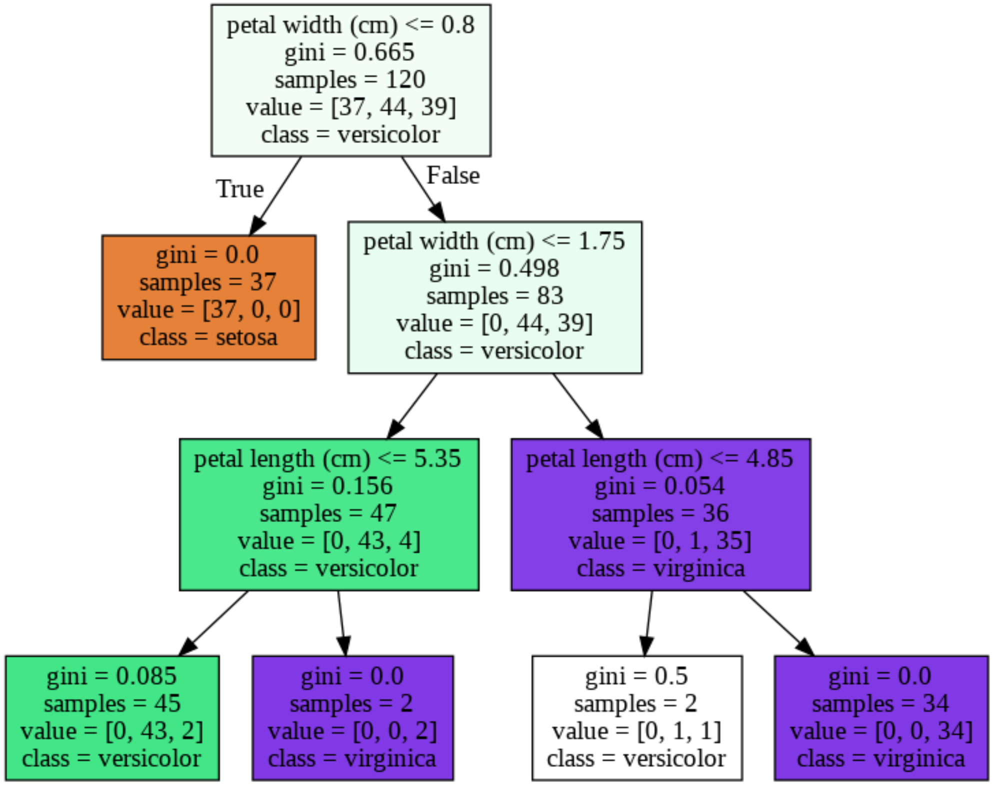
Split training and test sets

```
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)
```

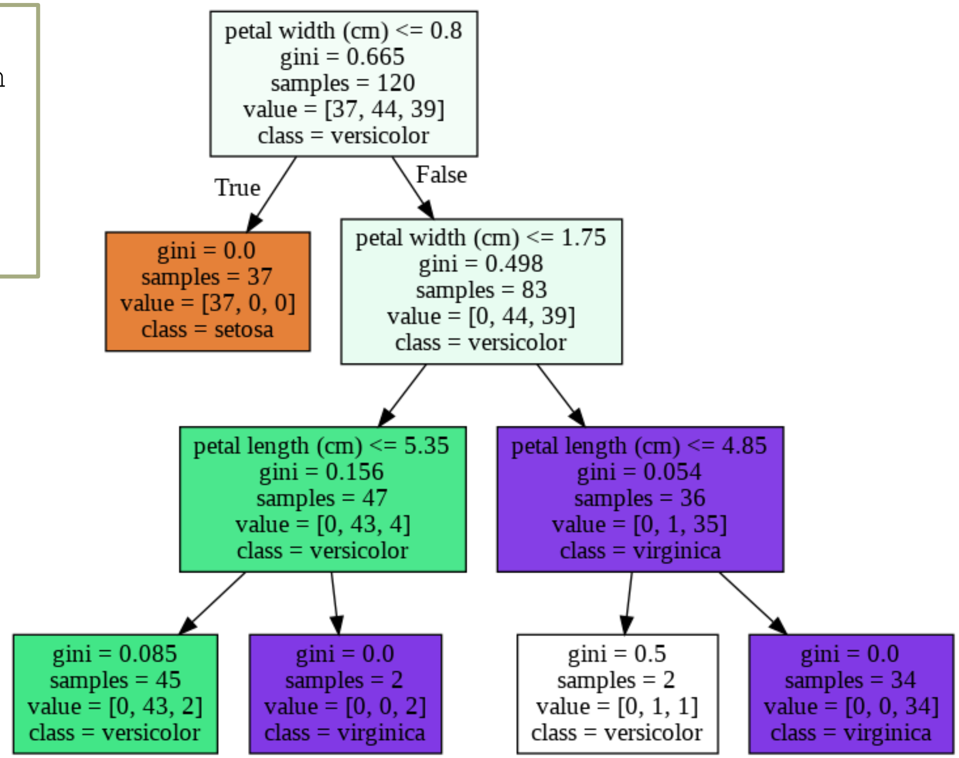


Making predictions using the trained model

```
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')
```

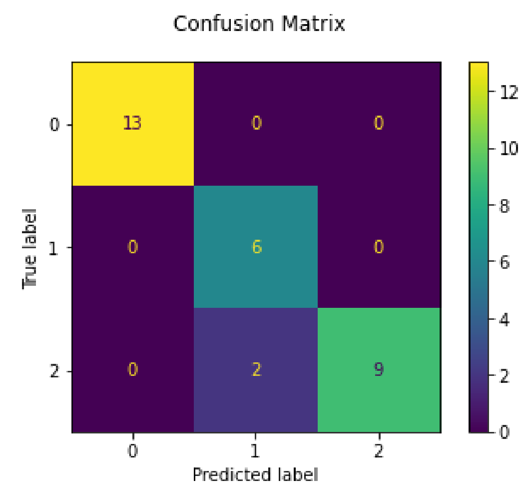
```
from joblib import load

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

Measuring the accuracy of the trained model

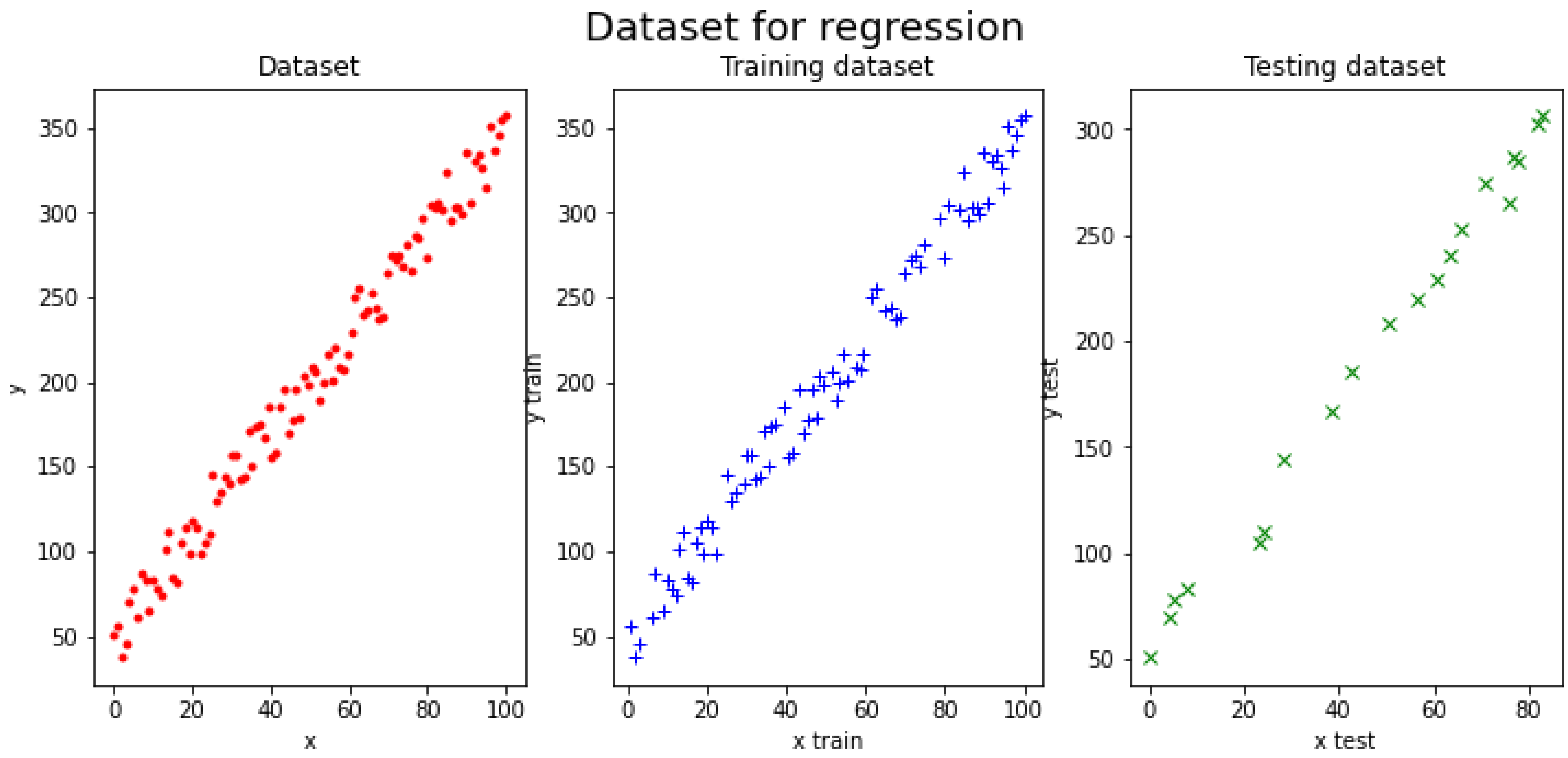
```
# 1. After training, the model is ready to make predictions:
tree_predictions = tree_classifier.predict(x_test)

# 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
```



Scikit-learn for regression: dataset

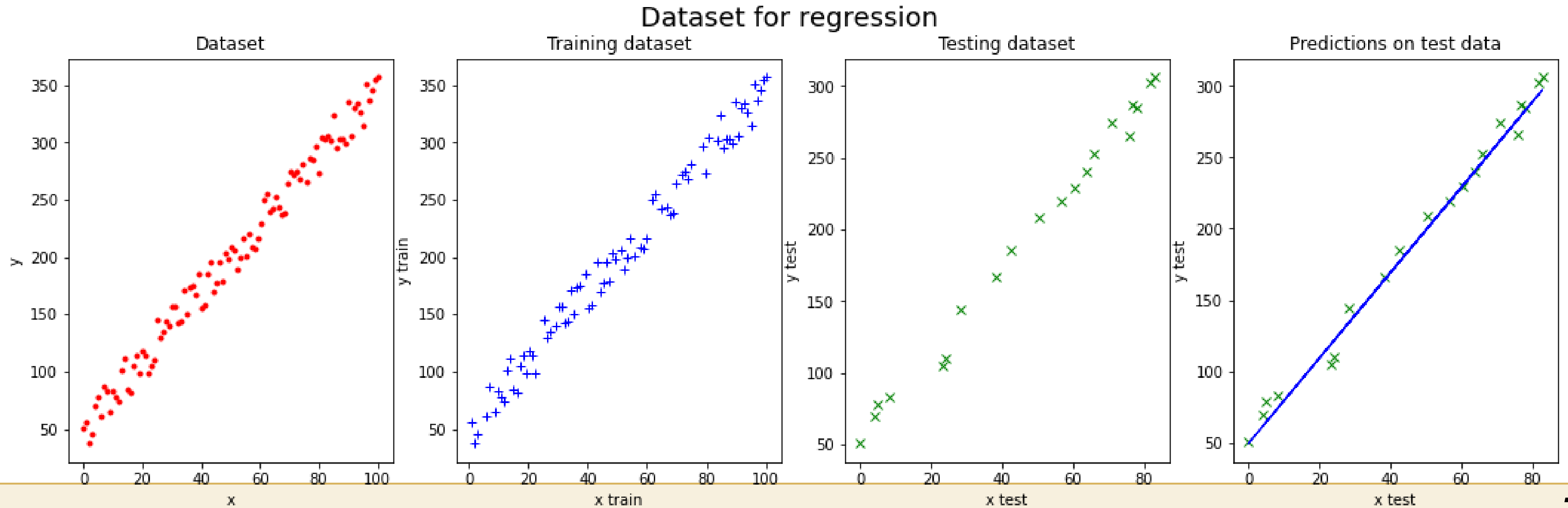


Scikit-learn for regression: training and testing

```
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_))
```

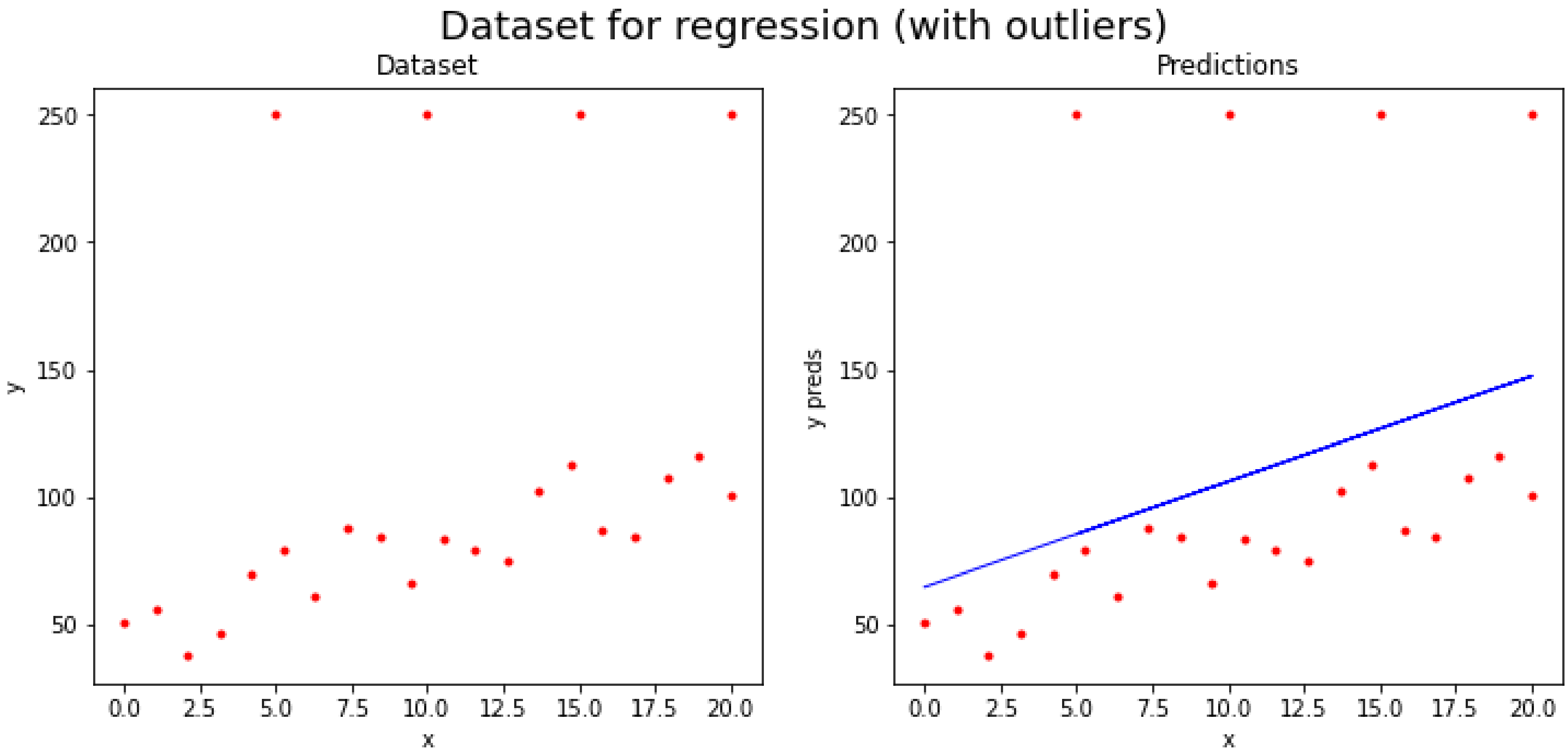
```
Coefficients: [[2.99811861]]
Intercept: [49.14094108]
```

```
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

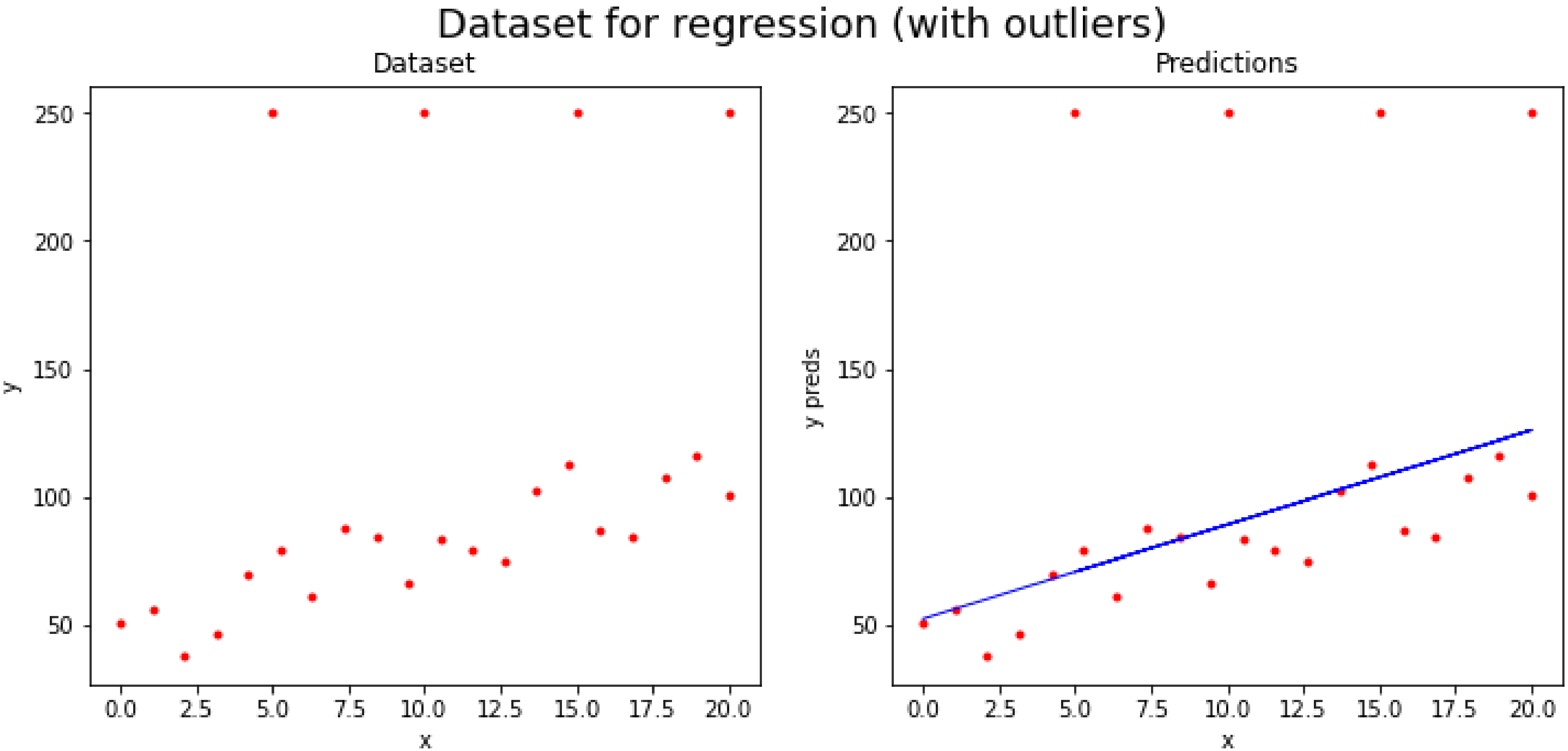


Scikit-learn for regression: dataset with outliers

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

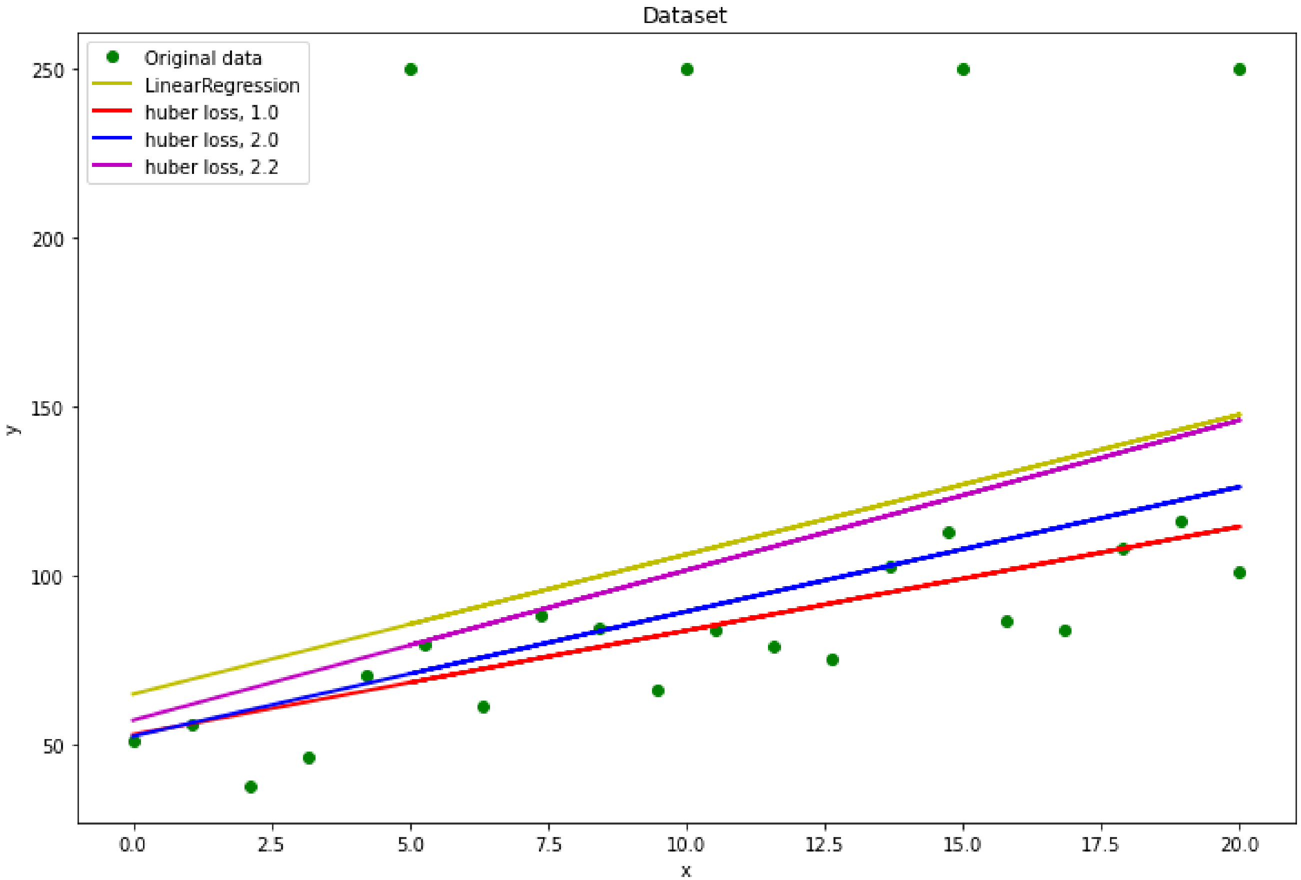
HuberRegressor(epsilon=2.0)



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)

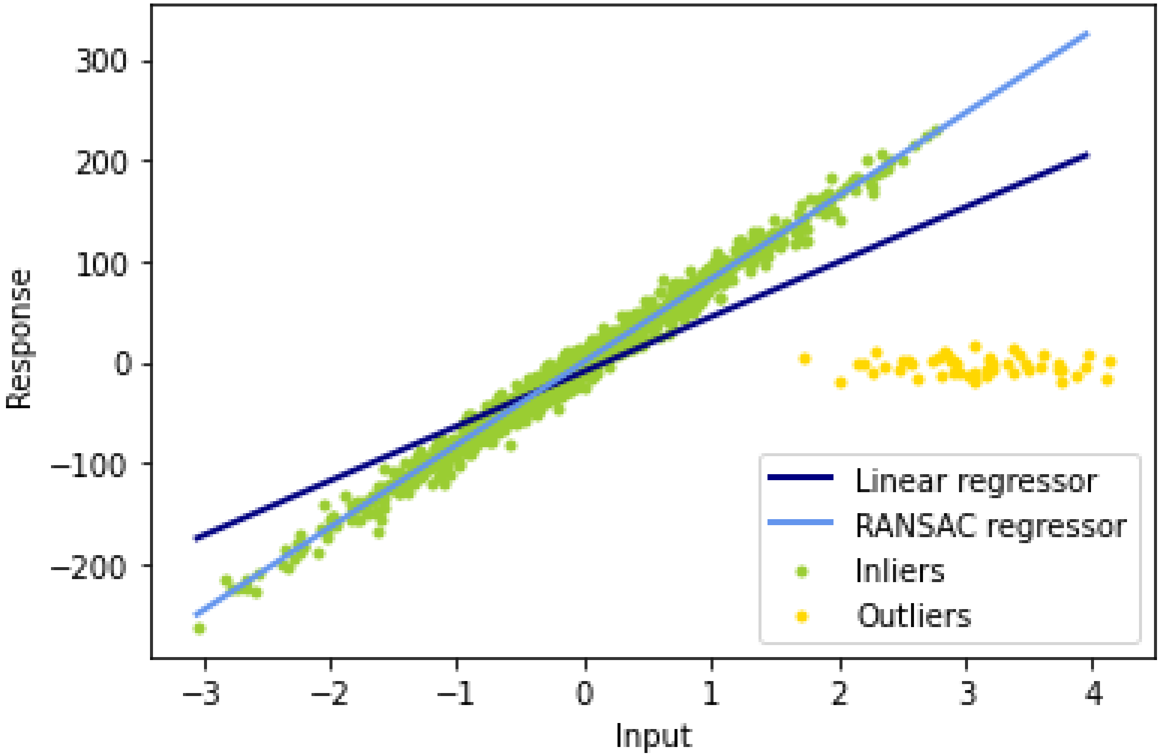


Scikit-learn for regression: dataset with outliers

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
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
```

[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