
# ADVANCED PROJECT II

## Automatic measurements of Physical Traits of Fish

*By Gari Ciodaro Guerra*

Advisor: Prof. Dr. Agostino Merico

Dec 24, 2019

**Abstract**

This report contains the theoretical considerations that were used in building the Automatic Measurement tool *AMT*, which can be used to measure standard physical dimensions of a fish giving an image. The picture must be on black background and with good quality.

This document is the final derivable of the course *Advanced Project II* from *Jacobs University* master on *Data Engineering*. The purpose of the course as expressed in the program syllabus is *to provide the student with an in-depth understanding and command of one of the data analytics or data management techniques that are represented by the research groups of the faculty of Data Engineering.*

# 1 Introduction

Functional ecology relies on physical traits to understand food acquisition and locomotion of fishes [1]. Measuring physical traits can help scientists understand diversity as well as behavioral dependencies in ecosystems such as reefs [2]. However, tools to carried out this measurements are short in supply, and are not open source. Giving this scenario, the purpose of this project is to build a tool(python software) to measure these traits. The covered dimensions can be seen in Figure 1.

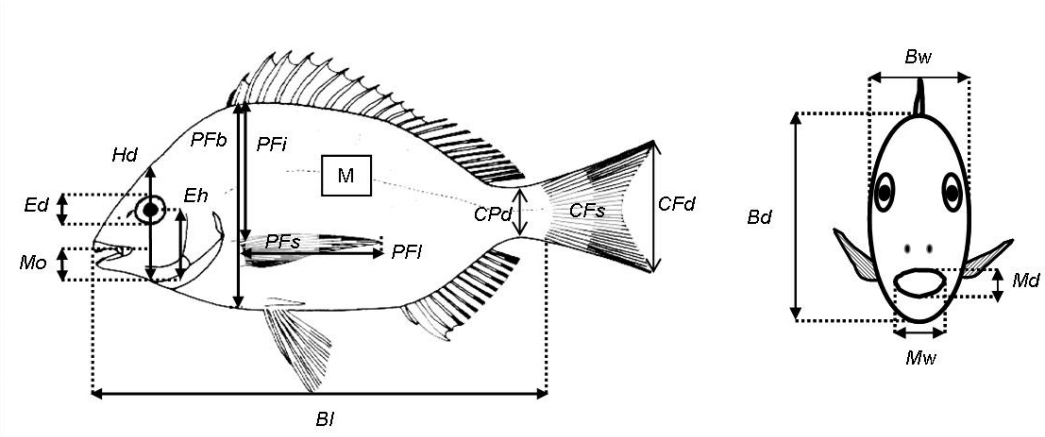


Figure 1: Morphological traits measured on digital pictures: **Bl** body standard length, **Bd** body depth, **CPd** caudal peduncle minimal depth, **CFd** caudal fin depth, **CFs** caudal fin surface, **PFi** distance between the insertion of the pectoral fin to the bottom of the body, **PFb** body depth at the level of the pectoral fin insertion, **PFl** pectoral fin length, **PFs** pectoral fin surface, **Hd** head depth along the vertical axis of the eye, **Ed** eye diameter, **Eh** distance between the center of the eye to the bottom of the head, **Mo** distance from the top of the mouth to the bottom of the head along the head depth axis. **Bw** body width, **Md** mouth depth, **Mw** mouth width.(taken from [1])

# 2 Image representation in Python

In *Python* an image is represented as a matrix of tree layers, where each layer is one of the primary color: Red, Green and Blue. Each entry in this matrix is therefore a tree dimensional vector where each component is an intensity value of R,G,B. The position (0,0) correspond to the top left entry of a the matrix.

# 3 Data exploration

We used 654 pictures random color-pictures of fishes and explore they general characteristic. In figure 2 we have the Height and Width distribution of the images. In order to apply K-means clustering to our set of images, we used the statistical mode of the dimensions of the images, that is: Height:360px (pixels) and Width:640px, as consequence each image is transformed into a row vector of 360x640 entries (interpolation using internal area was used). In other to asses which is the best number of clusters we implemented the *elbow* method, which is telling as tha $k = 5$ is good balance of the information gained by increasing the number of clusters and complexity reduction. By plotting the centroids of the selected $k = 5$ clusters, we decided to focus the development of the software on pictures that are assigned to cluster two. In principle the software can work with any fish image, but the result could be undesirable.

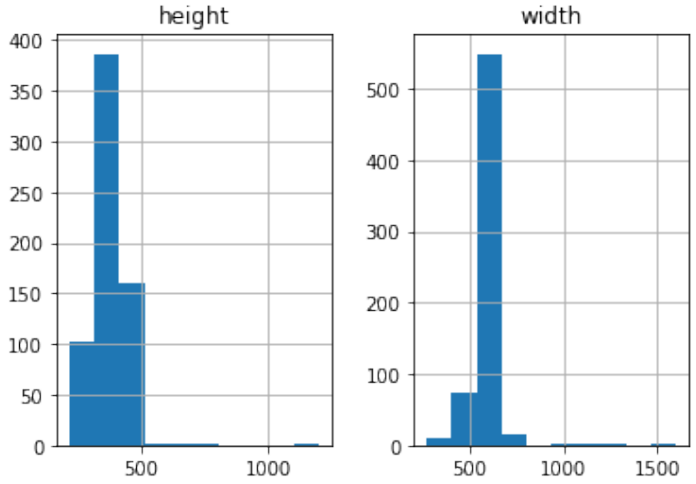


Figure 2: Dimension distribution in pixels of tested fish images.

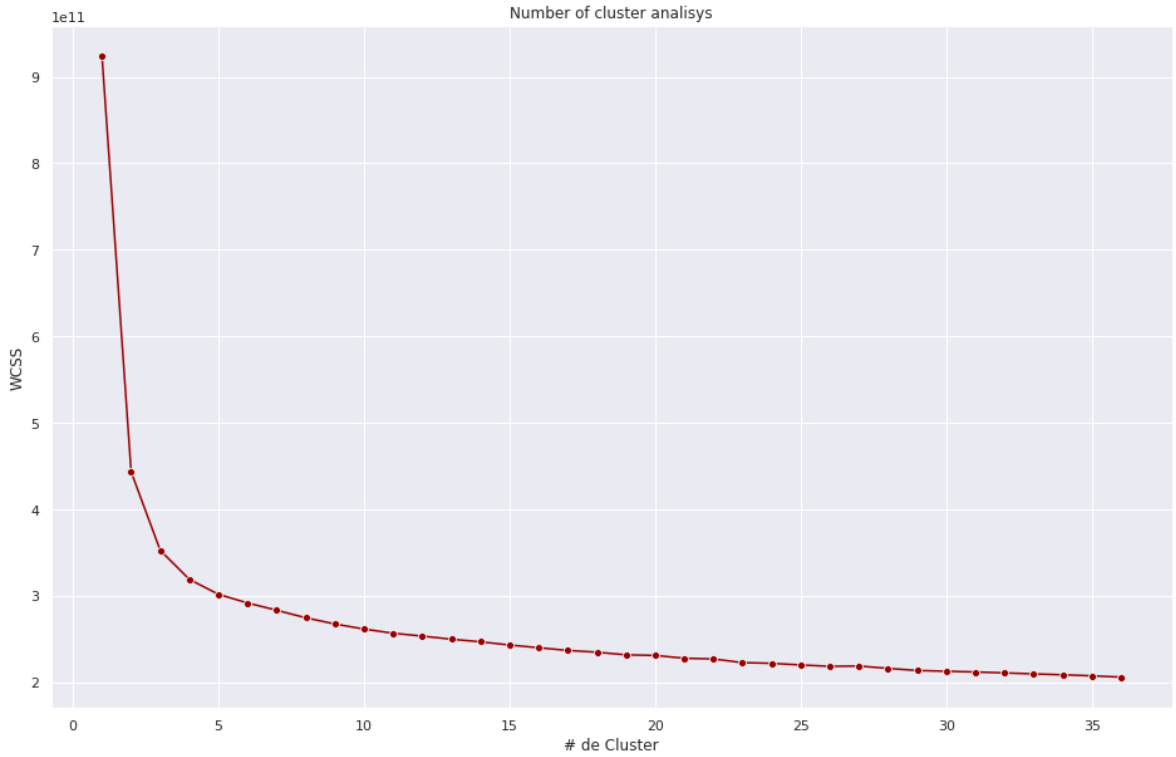


Figure 3: The elbow method for selecting a good balance between the loss of information and dimension reduction of the k means algorithm. $k = 5$ was selected. Within Cluster Sum of Squares (WCSS) [3]. Fewer clusters will have greater internal distances.

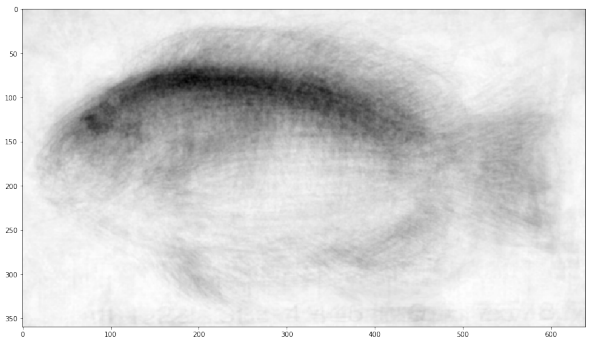

Figure 4: code vector, cluster 1 K-means 5

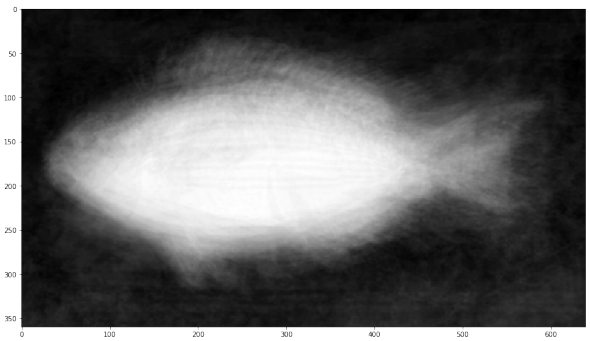

Figure 5: code vector, cluster 2 K-means 5

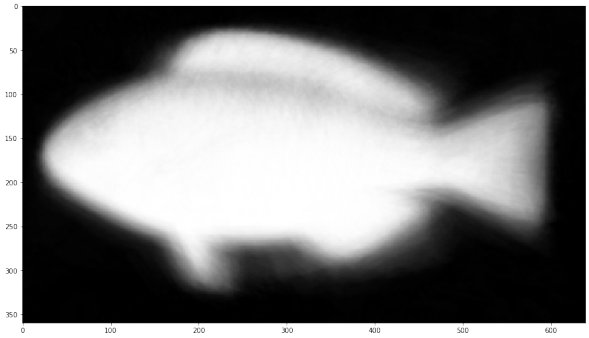

Figure 6: code vector, cluster 3 K-means 5

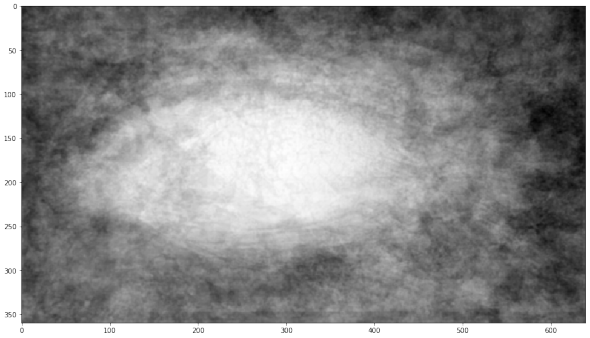

Figure 7: code vector, cluster 4 K-means 5

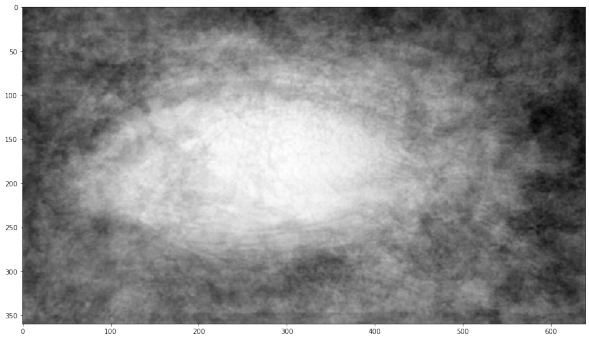

Figure 8: code vector, cluster 5 K-means 5

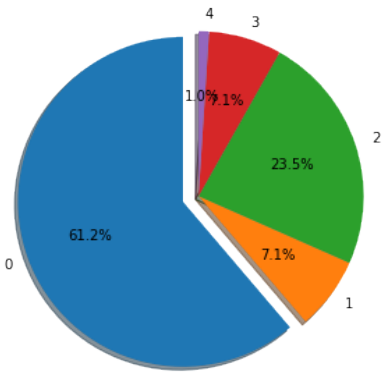


Figure 9: Distrubution of images over K-means 5 clusters

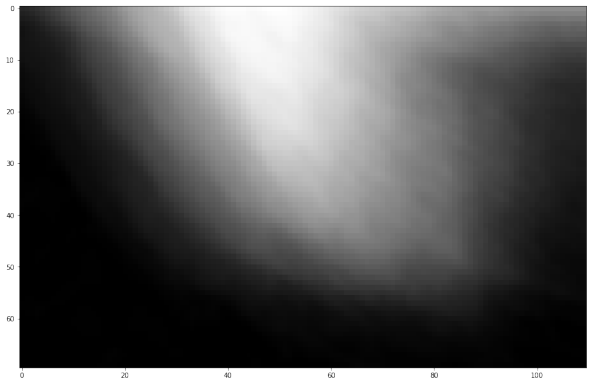

Figure 10: code vector lower fin, for template matching (cross correlation)

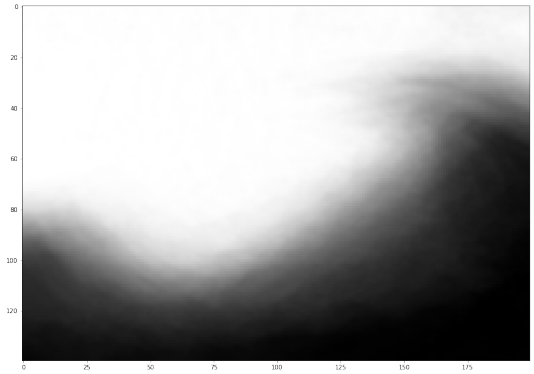

Figure 11: code vector longer lower fin, for template matching (cross correlation)

In later sections, cross correlation is used to template match the tail, and the outer body fins, we used the code vector of cluster 2 to get such templates.

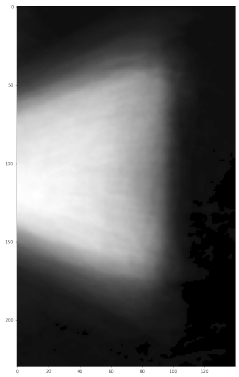

Figure 12: code vector tail, for template matching (cross correlation)

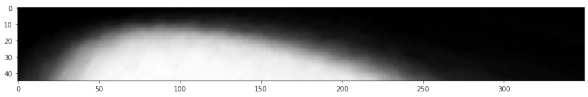

Figure 13: code vector upper fin, for template matching (cross correlation)

# 4 Image prepare

The pre processing pipe line goes as follows, take the tree channel image 14 and transform it into grey (one channel) 15, find the contour 16, multiply it by the original gray image 17, histogram equalized the result 18. This process will give us a enhanced features, and reduced noise from the background.

Figure 14: Raw input image. Recall that this image must be assigned to cluster 2, or cluster 1.

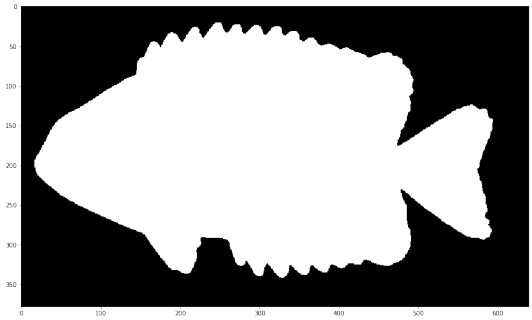

Figure 16: We find the contours using minimum energy curves (implemented in open CV) and the fill-up the recently found boundary

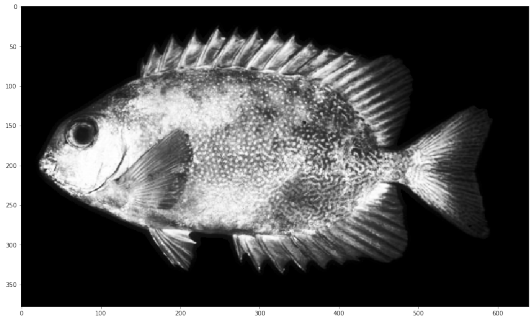

Figure 18: We again transform to gray, but now we histogram equalized the image. This will make use of the pixel space more effectively.

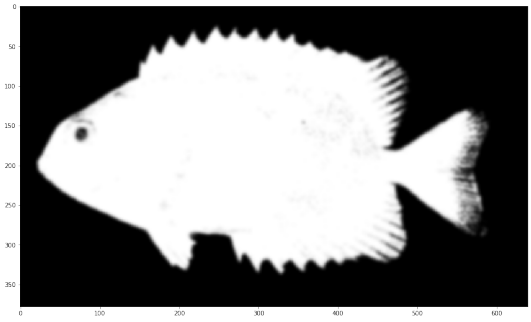

Figure 15: Converted to Grey and convoluted with a Gaussian kernel of 11x11 size. Here we implemented an average adjusted thresholding using a box of center pixels of the color image.

Figure 17: Multiplication of the filled contour with the original image. This will give a image with reduced information of the background

# 5 Getting the coordinates for CPd

We use the function *matchTemplate* from open CV to detect the tail of our image (recall that the template is 12). This function uses cross correlation to return the area of the image where the correlation coefficient is highest. The result can be seen in 19. The resulting cropped image is further process by calculating the shortest euclidean distance between none zeros vertical entries. the column of the cropped image with this minimum distance will be set as the coordinates for **CPd**. See function *tail_detection(masked_image,"CPd")* of the helper.py.

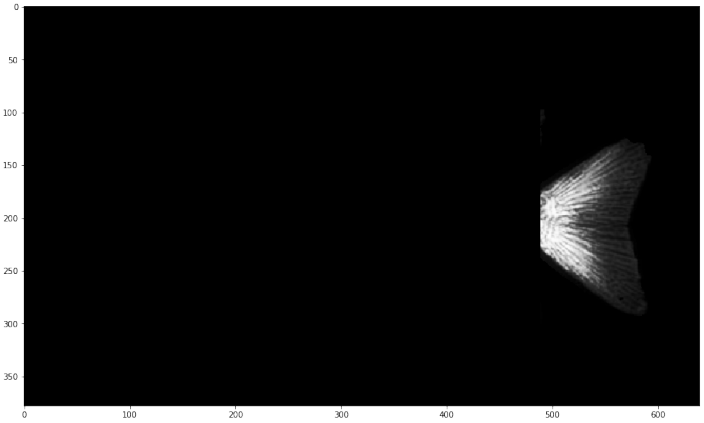


Figure 19: Cross correlation result on our sample image.

# 6 Getting the coordinates for CFd

To get this trait, the same process as in **CPd** holds, however, instead of finding the minimum distance between columns of the cropped images, we use the maximum distance. See function *tail_detection(masked_image,"CFd",CPd_coordinates[1])* of the helper.py.

# 7 Getting the coordinates for TL

To get **TL** we make use of the *Canny* algorithm. *Canny* is used for edge detection in computer vision. It roughly cover four steps: Smoothing (by Gaussian convolution), Finding gradients(by using Sobel filters (actually Sobel filters are used to Gaussian smooth and calculate an approximation of the partial derivatives in the same step)), Non-maximum suppression ( to avoid false positives and get smoothed curves), double thresholding and Edge tracking by hysteresis. The resulting image can be seen in figure 20. This image is binary, meaning is zero in all black entries and 1 in all white ones. Using this image we compute the sum of all horizontal row, and select the row with the maximum sum, using this row, calculate the first and last non zero column, which will complete the coordinates for TL. See function *TL(grey_image)* of the helper.py.

# 8 Getting the coordinates for Eh, Hd, Ed

Detecting circular shapes in images is a challenging task, here we used open cv function *HoughCircles*. We pass two different images to the *HoughCircles* function, one, a convolution using the Gabor filter 22 and another using the plain Grey image 18, we then count the number of detected circles, and select the biggest count if **confusion=True** or the smallest count if **confusion=False** ( on function

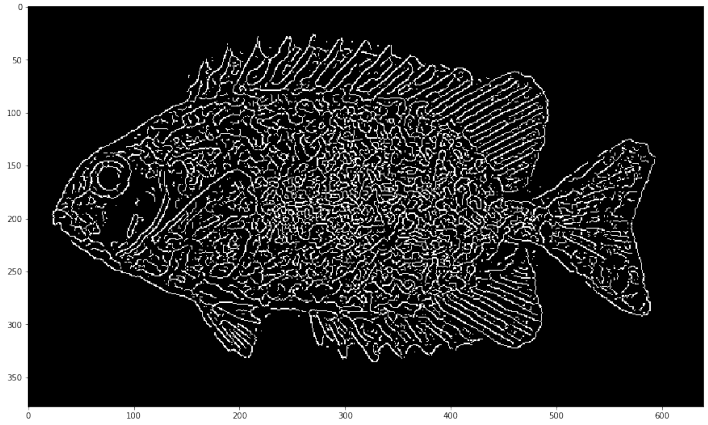

Figure 20: Canny output to calculate the **TL**

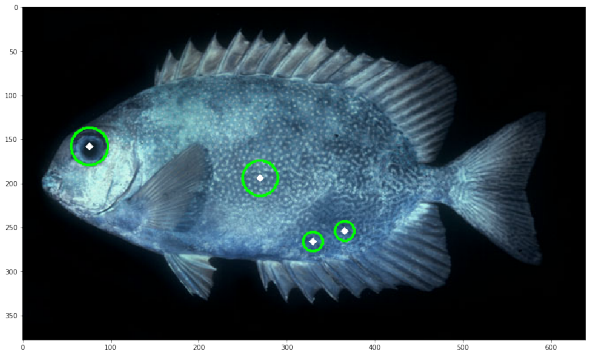

Figure 21: Circles detected by *HoughCircles* open cv function.

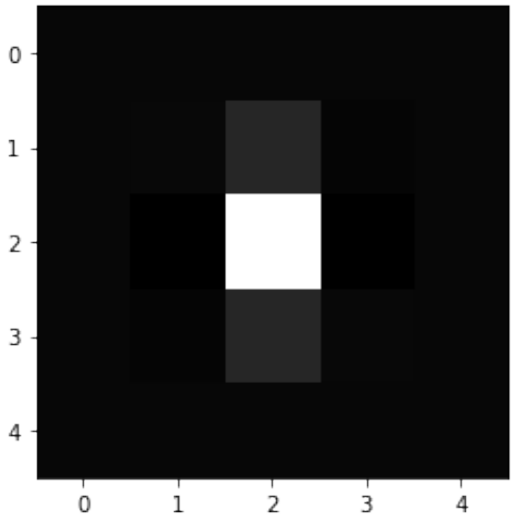

Figure 22: Gabor filter for eye detection.

*eye_detection* of the helper script). The resulting detection can be seen in 21. Once we have reduced number of candidates, we select the further left circle to be the eye. In case the software misses the proper eye, the user could change the parameter *minRadius* or **confusion=False**. In figure 21 we can see that an example of changing the parameter **confusion** which is True by default. Once we have the center of the eye, he coordinates of Eh, Hd, Ed are calculated by using the binary contour 16 in similar manner as TL.

# 9 Getting the coordinates for Bd

To get **Bd** coordinates we use cross correlation as in the tail detection section. We detect and subtract the code vector templates 10, 11 and 13 and then we apply the *Canny* edge detection algorithm. The result can be observed in 26. Additionally we restrict the search using the minimum column coordinate of Hd, and the minimum column coordinate coordinate of CPd. Using this reduced version

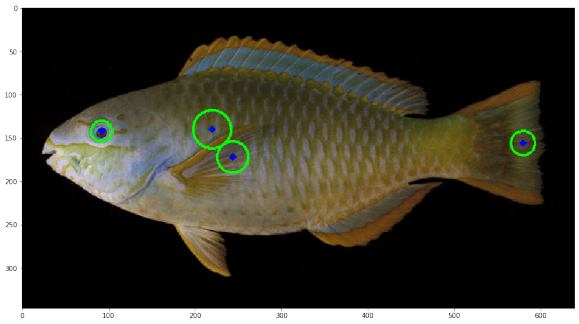

Figure 23: Eye detection **confusion=True**.

Figure 24: Eye detection **confusion=False**.

Figure 25: By adjusting the parameter **confusion** on the function *eye_detection* of the helper script we can increase the probability of getting the correct eye.

of the image, we proceed to calculate the vertical columns with the highest sum, and the assign leading and trailing coordinates to the first and last none zero entry of the columns.

# 10 Machine learning for Mouth and Pectoral Fin Detection

Detecting the Mouth and pectoral fin of the Fish using traditional techniques could not be achieved. As consequence, Machine learning was tried. Even though the fin was detected, further development is required to have a reliable measurement.

## 10.1 Feature extraction of images

Before applying machine learning, we constructed the class *BowAux* that extract features of the images in a standardized and reliable manner. The technical details are beyond the scope of the present report, but the idea is to find a representation of an image that is scale invariant and that can capture enough information of it, that can be used to characterized it. Feature extraction is carried out as follows (also see class *BowAux*).

- Split each image of a fish in your training set into smaller boxes of 50x50 Pixels (*image_splitter* function of the helper script).
- Extract the descriptors of each box using the SIFT (Scale invariant feature transform).
- Pass all the SIFT descriptors of each unlabeled box to the an open cv object called *BOWKMeansTrainer*.
- Once all the training boxes has been passed to *BOWKMeansTrainer*, call the cluster method on it, this will use histogram frequency of your descriptors to create relevant *image-vocabulary-dictionary* that will be later used to represent an incoming image based on the similarities to this image-vocabulary.

The previous process is known as the Bag of words approach in Natural language Processing. The details are highly technical so no further explanation is giving here. It is enough for this report to know, that every time a new image arrives, we can use this vocabulary to represent it in a scale invariant fashion and with consistent dimensions, as any machine learning algorithm requires.

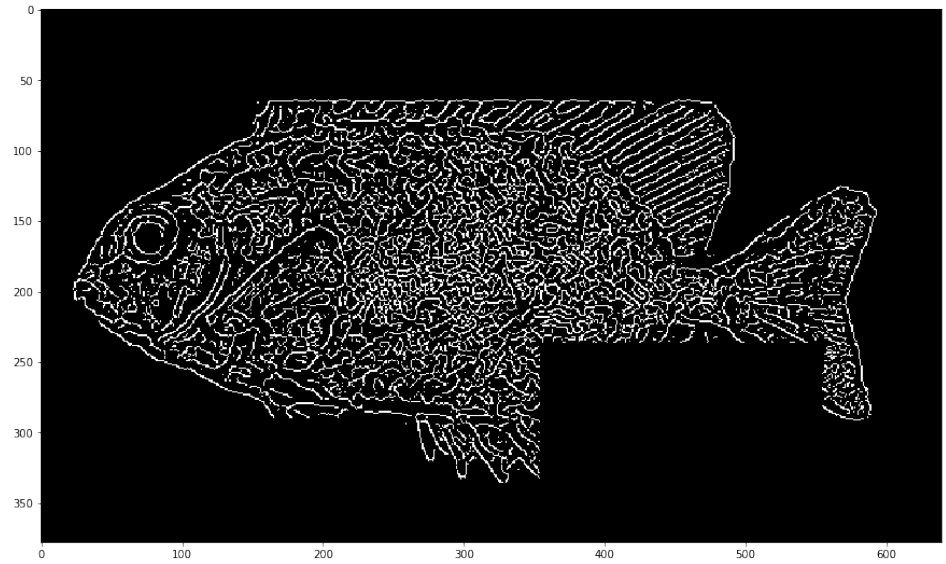


Figure 26: Reduced image for Bd coordinate calculation. We detect and subtract the code vector templates 10, 11 and 13 and then we apply the *Canny* edge detection algorithm.

## 10.2 Training

We manually labeled 204 images, and selected 183 for training and 21 for testing. This last 21 were never used to tune any hiperparameter. We labeled the eye, the mouth an the pectoral fin, however only the mouth and the eye were trained due to lack of computational resources. The problem proved to be very demanding, not only we have highly unbalanced classification task, but traditional metrics were taking carefully, since geometry could be use to further increase the probability of detection. *F1-Score* was the metric optimized but the highest $AUC$ of the $ROC$ curve was selected.

The training process consists in oneHot encoded the labels Mouth and Fin and tread the problem as independent binary classification. In both cases the following process was carried out.

1. Take the training set, and split randomly into hiper-train set and validation set, in 90-10% manner keeping the proportion of the target class the same in both sets.
2. scale the data, create tree synthetic sets: 1. no clustering, no synthetic polynomial degree two. 2. cluster and optimized k by elbow method. 3. no clustering, use synthetic polynomial degree 2, select 100 best features based in chi statistic test.
3. Fit Default parameters in the tree synthetic sets using 5-fold cross validation on the hiper-train set, to get a base line of: **Random Forest**, **Logistic Regression**, **Extreme Gradient Boosting**,**Multilayer Perceptron**,**Linear Discriminant Analysis**, **Quadratic Discriminant Analysis**, **K nearest neighbors** [4].
4. select the higest cv-score base line model of the tree synthetic sets. Also select the model with lowest standard divation on the cv test.
5. Perform 20 random hiperparameters fit evaluating each of them using cross validation, on both best two previous models.
6. select the best hiperparameters configuration, on both models.
7. perform a last fit on the on the validation set, select the higest score as your final model.

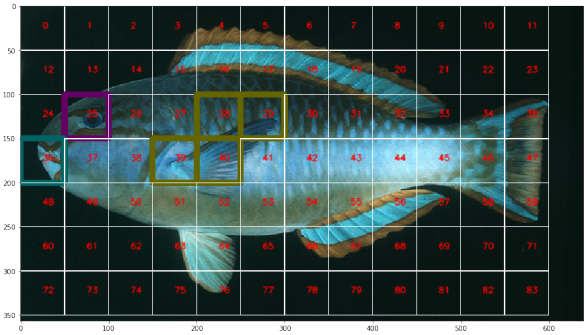

Figure 27: Example 1.

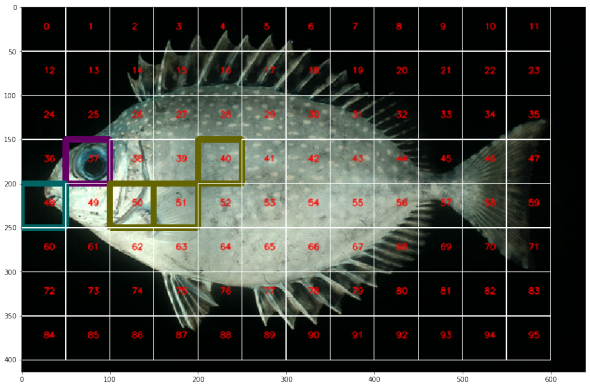

Figure 28: Example 2.

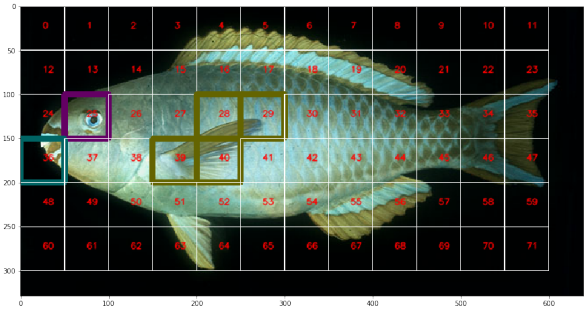

Figure 29: Example 3.

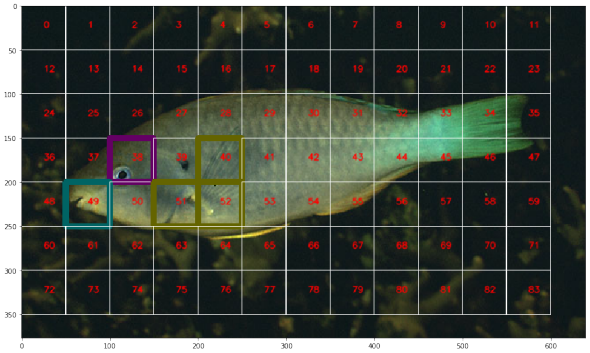

Figure 30: Example 4.

Figure 31: Example train images for eye, mouth and pectoral fin detection with machine learning. Purple for eye, blue for mouth, and gold for fins.

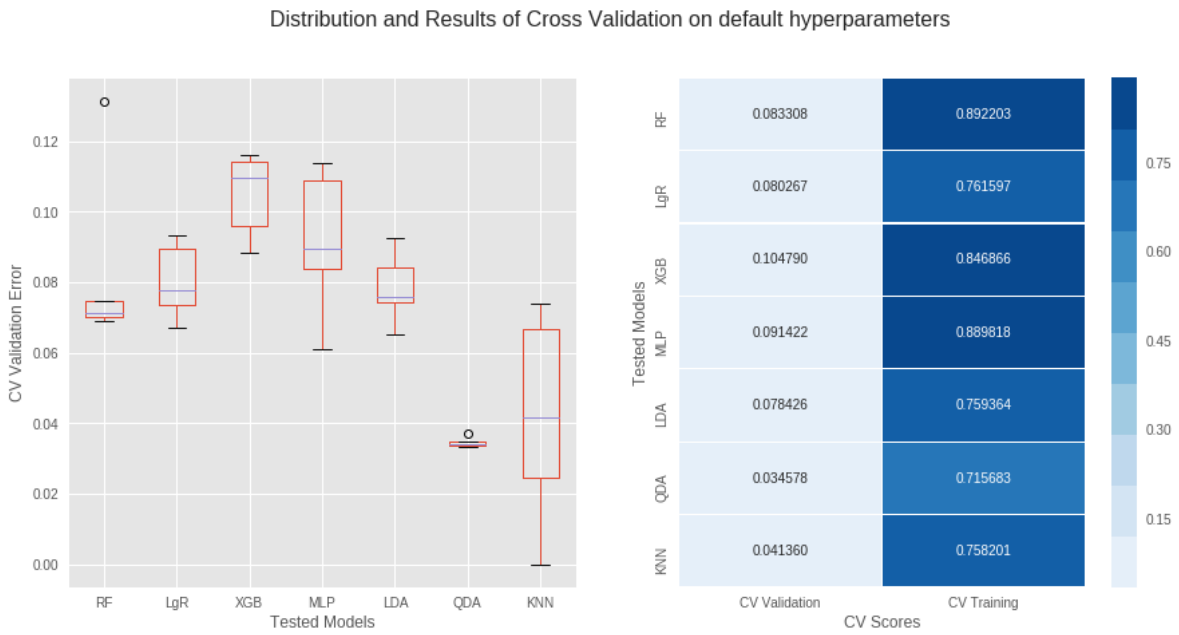


Figure 32: Base line score for Mouth class.

8. fit the best model on whole dataset (hiper-train set and validation set).

## 10.3 On Training for Mouth detection.

- Optimized metric: f1_score
- clustering: False
- polynomial: False
- balanced train: True
- Selected Machine Learning algorithm: XGB
- Selected Hiper-Parameters: 'min_child_weight': 1, 'max_depth': 3, 'learning_rate': 0.1, 'reg_alpha': 0.02, 'reg_lambda': 0.8, 'gamma': 0.002, 'subsample': 1.0, 'colsample_bytree': 1.0, 'objective': 'binary:logistic', 'nthread': -1, 'scale_pos_weight': 1, 'seed': 10, 'n_estimators': 200
- Final Training Score after Hiper-Parameters tunning or Further optimization: 0.8571
- Expected score on unseen data (10% validation set): 0.11
- Measured processing time: 1831.70 seconds.
- Training score for full data set: 0.8575

## 10.4 On Training for Fin detection.

- Optimized metric: f1_score
- clustering: False

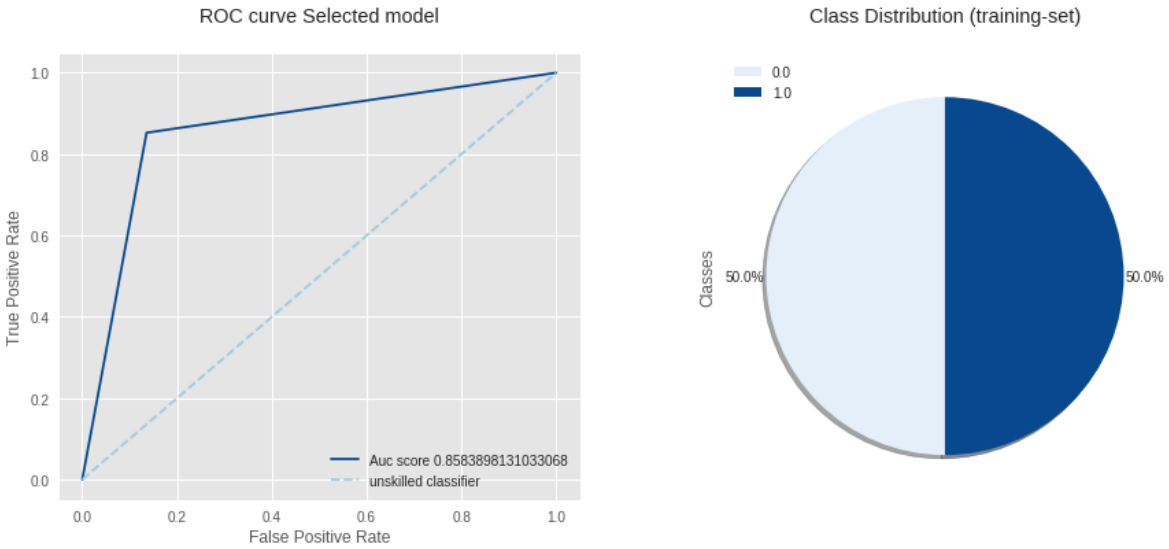


Figure 33: ROC plot for best Mouth classifier, XGB. Balanced by random oversampling

- polynomial: False
- balanced train: True
- Selected Machine Learning algorithm: XGB
- Selected Hiper-Parameters: 'min_child_weight': 1, 'max_depth': 3, 'learning_rate': 0.09, 'reg_alpha': 0.02, 'reg_lambda': 1, 'gamma': 0.03, 'subsample': 1.0, 'colsample_bytree': 0.7, 'objective': 'binary:logistic', 'nthread': -1, 'scale_pos_weight': 1, 'seed': 10, 'n_estimators': 100
- Final Training Score after Hiper-Parameters tunning or Further optimization: 0.69
- Expected score on unseen data: 0.16
- Measured processing time: 4039.09 seconds
- Training score for full data set: 0.70

# 11 Conclusions

The following traits were implemented: Cdp,Tl,Eh,Hd,Ed,Bd,CFs,CFd and Mo. For the pectoral Fin, the machine learning is need of more work, perhaps by incrementing the training set. As it now, the software will draw purple boxes with the probability of the fin. As can be observed, the classifier is still very uncertain. The software requires that Tl length of the fish is passed in the name of the image, so it can be used as the reference between pixel distance and physical distance.

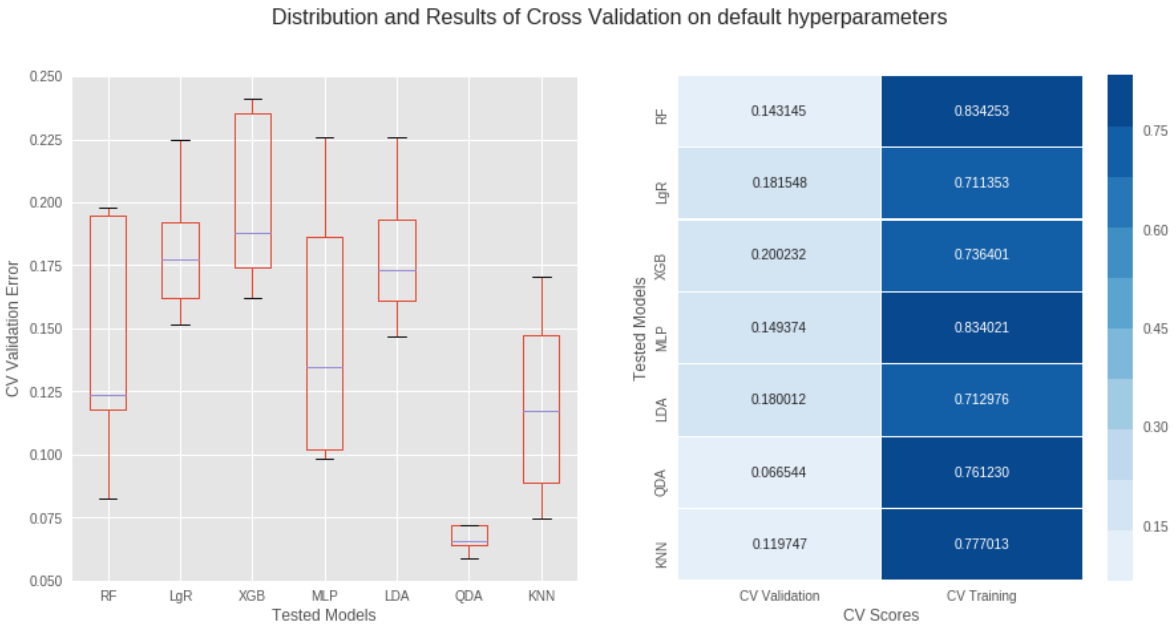


Figure 34: Base line score for Fin class.

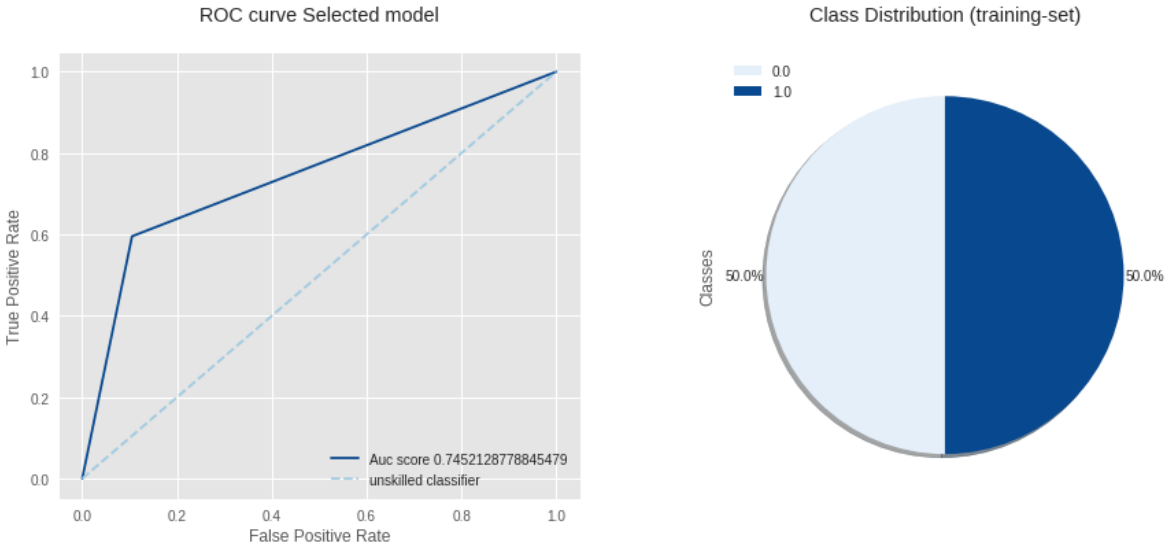


Figure 35: ROC plot for best Fin classifier, XGB. Balanced by random oversampling

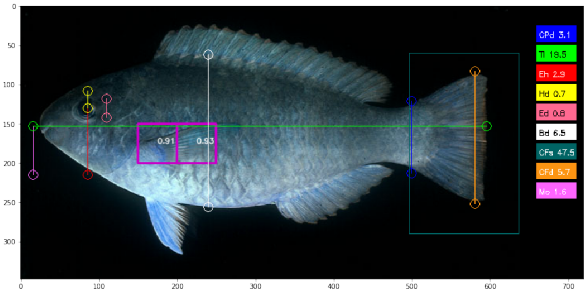


Figure 36: Demo 1.

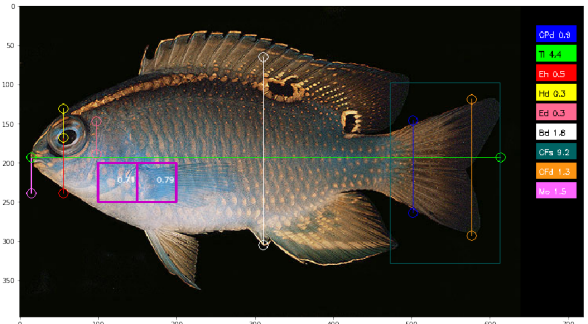


Figure 37: Demo 2.

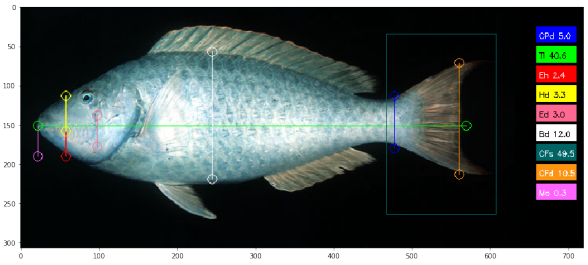

Figure 38: Demo 3.

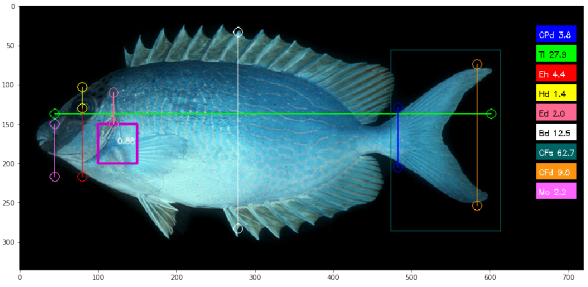

Figure 39: Demo 4.

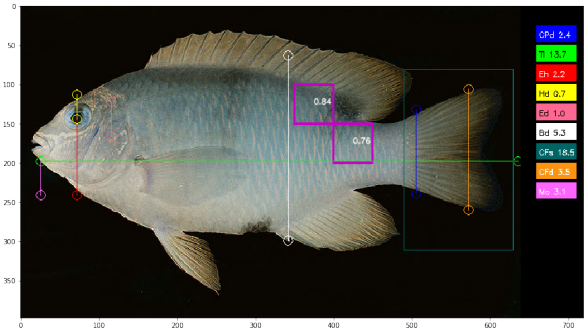


Figure 40: Demo 1.

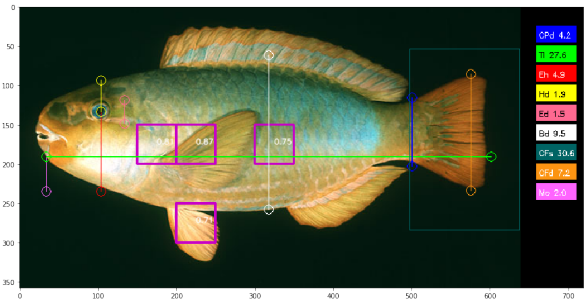


Figure 41: Demo 2.

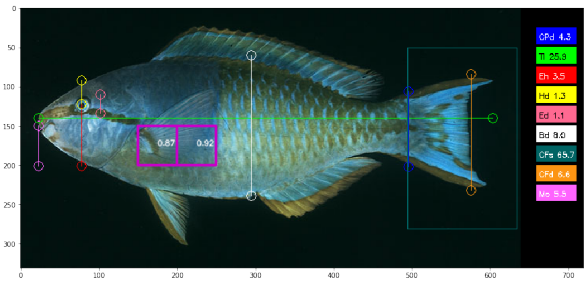


Figure 42: Demo 3.

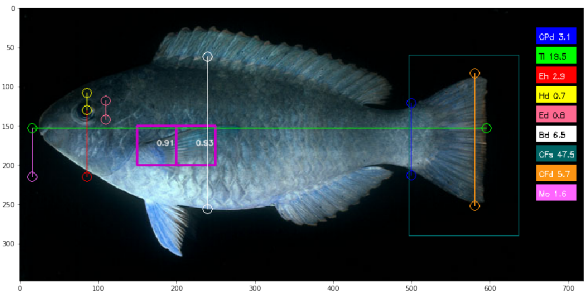


Figure 43: Demo 4.

# Appendix A Code Repository

All the code implemented and utilized during the execution of the data workflow described in this report is available at the GitHub repository https://github.com/gariciodaro/AMT.git.

# References


[1] Sébastien Villéger et al. "Contrasting changes in taxonomic vs. functional diversity of tropical fish communities after habitat degradation." In: *Ecological applications* 20.6 (2010), pp. 1512–1522.

[2] Simon J Brandl and David R Bellwood. "Morphology, sociality, and ecology: can morphology predict pairing behavior in coral reef fishes?" In: *Coral Reefs* 32.3 (2013), pp. 835–846.

[3] The Scikit-Learn community. *sklearn Gradient Boosting Regressor*. Apr. 2019. URL: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html.

[4] Andreas C Müller, Sarah Guido, et al. *Introduction to machine learning with Python: a guide for data scientists*. " O'Reilly Media, Inc.", 2016.