|  |  |
| --- | --- |
| **Breed Locator Machine Learning Module**  **Machine Learning** | May 3rd, 2019 |

# **Definition**

### **Project Overview**

Animals, like humans, have unique personality traits, and these can be generalized to the specific breeds. At the very least, it would be desirable for a animal owner to know what type of personality to expect from a animal breed before bringing one into their farm. This is easy enough to do if a animal is a purebreed—simply ask the owner or breeder what type of animal it is. For hybrids, determination of the breed becomes much more difficult.

Wikipedia has a more comprehensive list of breeds, since they include more international breeds. Wikipedia also provides a list of hybrid breeds. Other online resources provide lists and pictures of breeds and hybrids.

### **Problem Statement**

The goal of this project is to create a machine learning application, which reports the animal breed that most closely resembles an input image. The breeds listed will be used to classify animal breeds by photo.

To create this application, training data must first be mined since a dedicated animal image database does not currently exist. Features need to be extracted from the training set; these could be texture, key points, and histograms of oriented gradients (HOG), and color information. A machine learning algorithm needs to then be trained and cross-validated on portions of the training set. The machine learning algorithms tested here will be support vector machine (SVM), K-Nearest-Neighbors (KNN), and Random forest decision tree classifiers.

### **Metrics**

Accuracy (fraction of correct predictions out of total predictions) will be used as a metric in this project. Recall, precision, and F1 scores would be difficult to use as a metric because there are so many class breeds, and the accuracy of the solution will likely be low (due to the large number of class breeds). The goal is to beat random guessing on the test data set (used as the benchmark).

# **Analysis**

### **Data Exploration**

Intuitively, there are some features about pictures of animals we use to discern their breed: color, fur texture, body shape, face shape, and size. Of these, we can easily calculate of all of these features except size. For this project, I chose to use fur texture as the main classification item, because the texture of animal coats tends to differ greatly between many breeds. Texture was measured with Haralick texture, which is a common way to compute texture of an object in computer vision.

To better facilitate classification, am going to manually draw rectangles around the animals and their faces. Using the cv2.grabCut() function, I separated the foreground (animal) from the background of the image, with the rectangle as a starting point for the k-Means algorithm behind grabCut. I hypothesized this would allow for better feature extraction since the background noise would be reduced, and it would also allow for a comparison of the features of the animal and the backgrounds.

I then use the grabCut mask to break up the foreground and background into rectangles, so Haralick features can be calculated for each. Haralick features will then be used to calculate for each 10px by 10px rectangle, and averaged over the foreground and background to get a single set of Haralick features for each foreground and background from each image.

Haralick texture outputs 14 features across 4 directions. Only the first 13 features are used in the mahotas Python implementation of Haralick (the 14th is considered to be unstable). The 4 directions are then averaged to arrive at a 13-dimension vector.

Examples of the Haralick features stored in a pandas DataFrame follow:

**animal:** cow

**breed**: Fresian

**filename**: ef144fab7422d2cd75cea6f09c3936d2813e7ed4.png

**foreground Haralick**:

[ 8.57381478e-03 4.10863443e+02 5.87781055e-01 5.65823223e+02

1.21756636e-01 1.36821823e+02 1.85242945e+03 5.59116418e+00

7.13270498e+00 8.16455498e-04 4.44472415e+00 -6.28489862e-01

9.94489378e-01]

**background Haralick**:

[ 7.55733120e-03 2.25157047e+02 5.39230190e-01 2.34382991e+02

1.24854741e-01 2.01441970e+02 7.12374918e+02 5.45011728e+00

7.16052523e+00 4.89178675e-04 4.19504514e+00 -5.70747974e-01

9.96462007e-01]

**animal:** cow

**breed**: Jersey

**filename**: e324673247810093289fds8231cas723s32312.png

**foreground Haralick**:

[ 4.63559378e-02 2.35314004e+02 7.53242876e-01 6.28036192e+02

3.35673427e-01 1.42878603e+02 2.27683077e+03 4.81972944e+00

6.00407352e+00 5.46744200e-03 3.33448890e+00 -5.67899390e-01

9.63606223e-01]

**background Haralick**:

[ 3.40171120e-02 4.37131707e+02 7.06957257e-01 8.20488436e+02

2.41225782e-01 1.92114355e+02 2.84482204e+03 5.21970324e+00

6.48837378e+00 3.50651503e-03 3.89854009e+00 -6.14770211e-01

9.73879275e-01]

Feature number 12 is negative, and the features span 4 orders of magnitude. The 2nd, 4th, 7th, and 10th Haralick features seem to differ the most in the chosen examples.

I then reduce the dimensionality of the Haralick features to 3 using PCA. Then, using quartile outlier analysis, where outliers are said to be 1.5 times outside the interquartile range (Q3-Q1), I find out that 75 total outliers in the reduced Haralick features. I found 12 entries that were outliers in more than one dimension, and dropped these outliers from the dataset.

**Exploratory Visualization**

The Haralick features of the foreground (breed) seem to differ most for features 1, 2, 4, 5,

6, 7, and 10, as can be seen from a subset of the data:

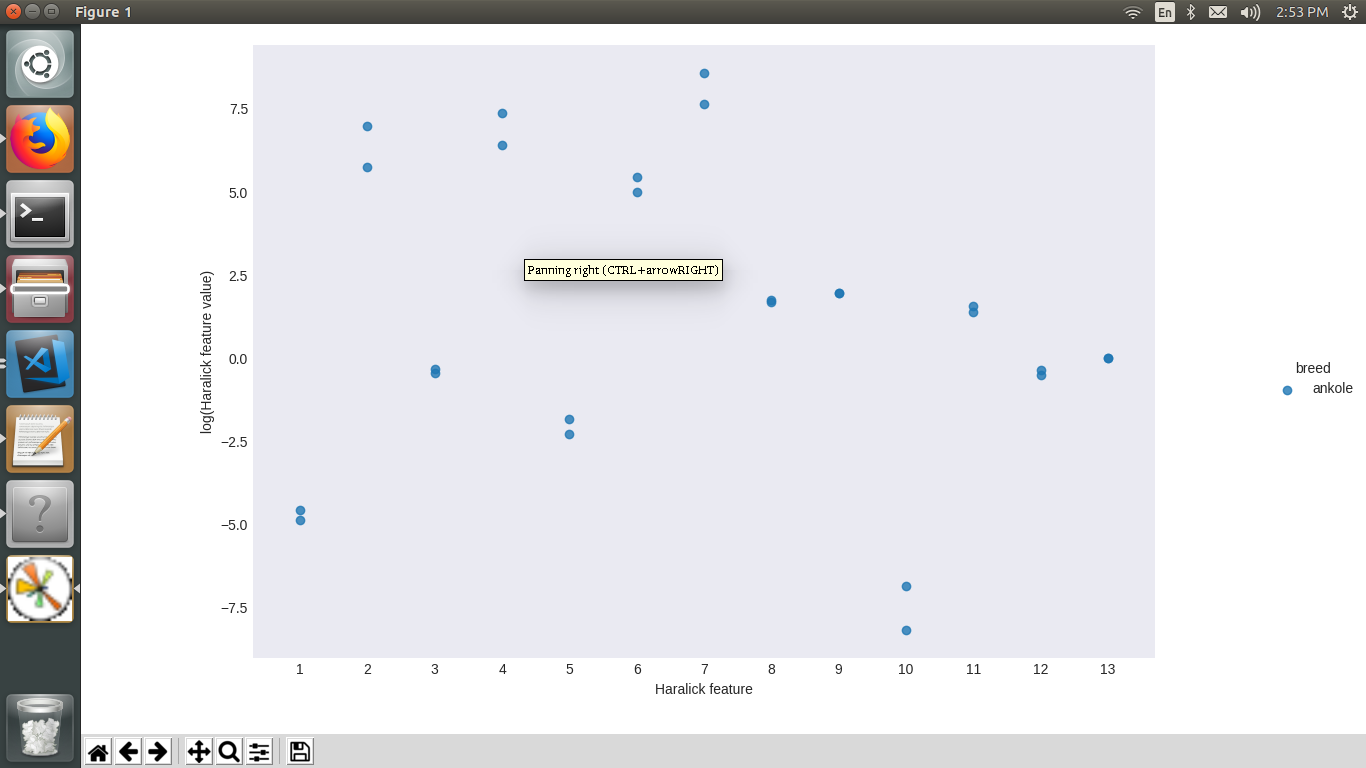
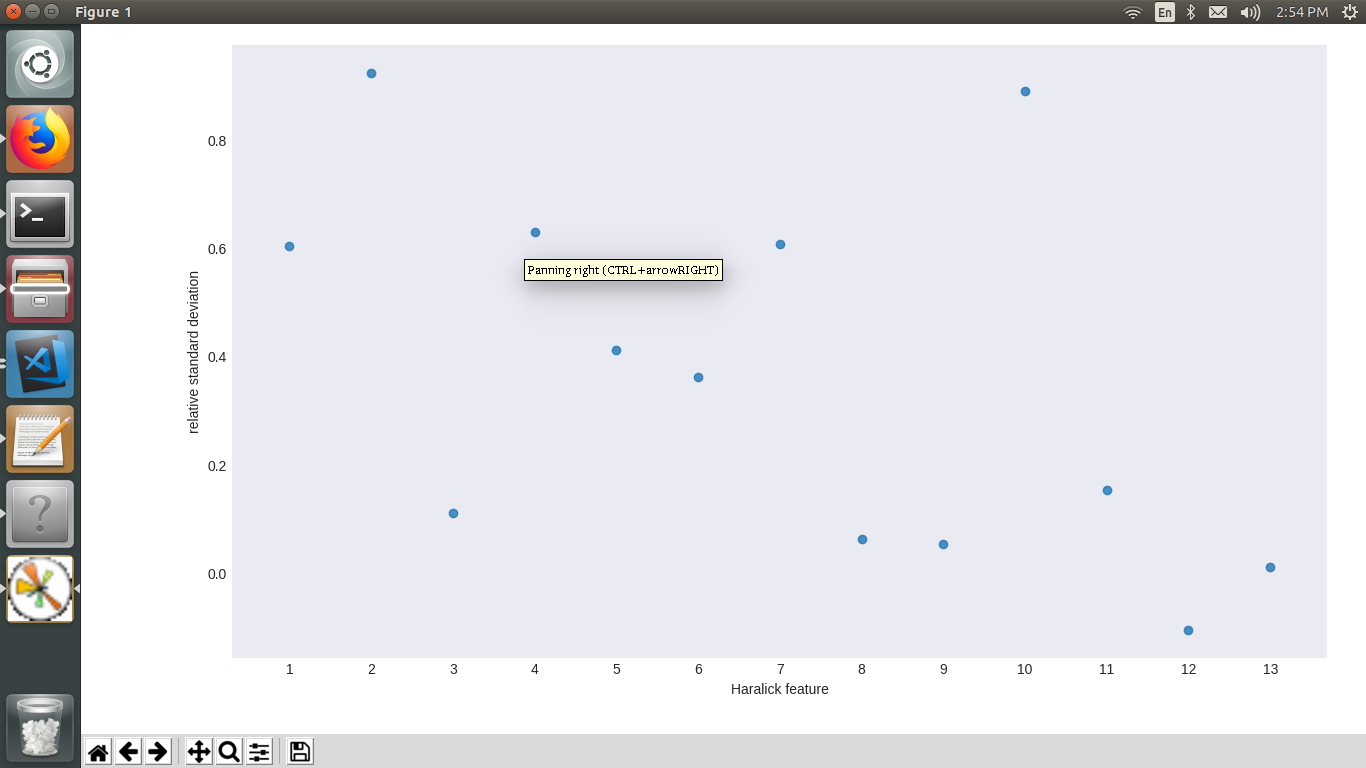


Figure 1: Distribution of the log(abs(x)) transformed foreground Haralick textures for a subset of animal breeds.

Overall, the standard deviations of the foreground Haralick features follow this trend:

Figure 2: Relative standard deviation of the 13 dimensions of the foreground Haralick texture.

with feature # 10 being a clear outlier in standard deviation.

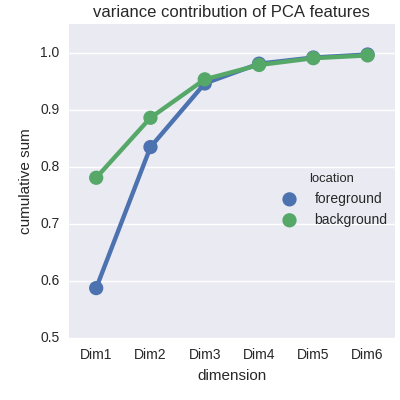


Figure 3: Cumulative sum of the variance captured by the first 6 Haralick PCA components.

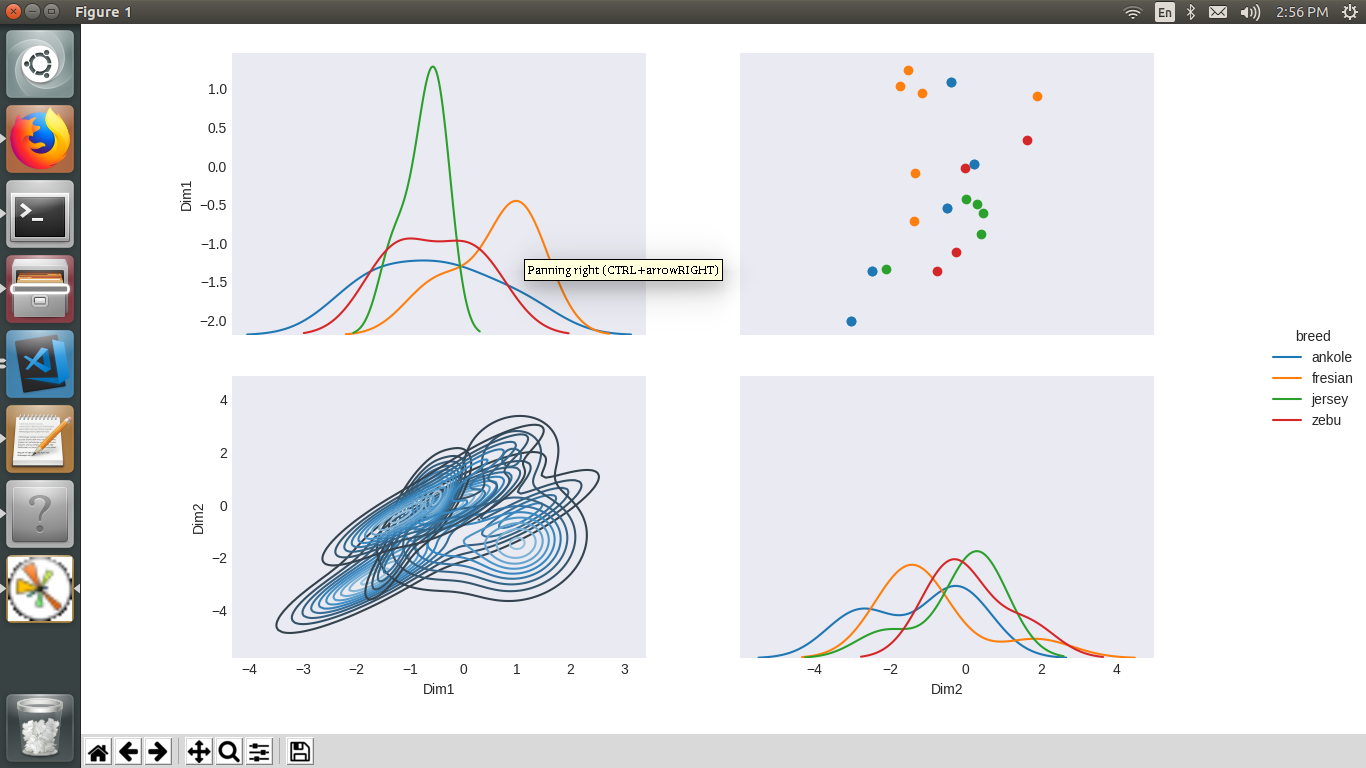
Once the 13-dimension Haralick features were calculated, I transformed them by taking

the log of the absolute value of each feature, since the features span multiple decades

and some are negative. Principal component analysis was then performed; it was found

the first 3 components of the PCA make up about 95% of the variation in all parts of the

images, and the top 4 components about 98%.

 Figure 4: Pair plot of the foregrounds' first 3 Haralick PCA dimensions.

We can see from a subset of breeds that there is some clustering within breeds in the first

3 dimensions of the foreground Haralick texture PCA, but there is a lot of overlap. This

does not bode well for classification based on these features.

Finally, a quick look at the grabCut() function, which separates foreground from

background actually uses a k-means algorithm behind the scenes. We can see from

comparing foreground and background RGB histograms that if there is some separability

in color from foreground to background, grabCut() does much better than if the

foreground and background color distributions are similar.

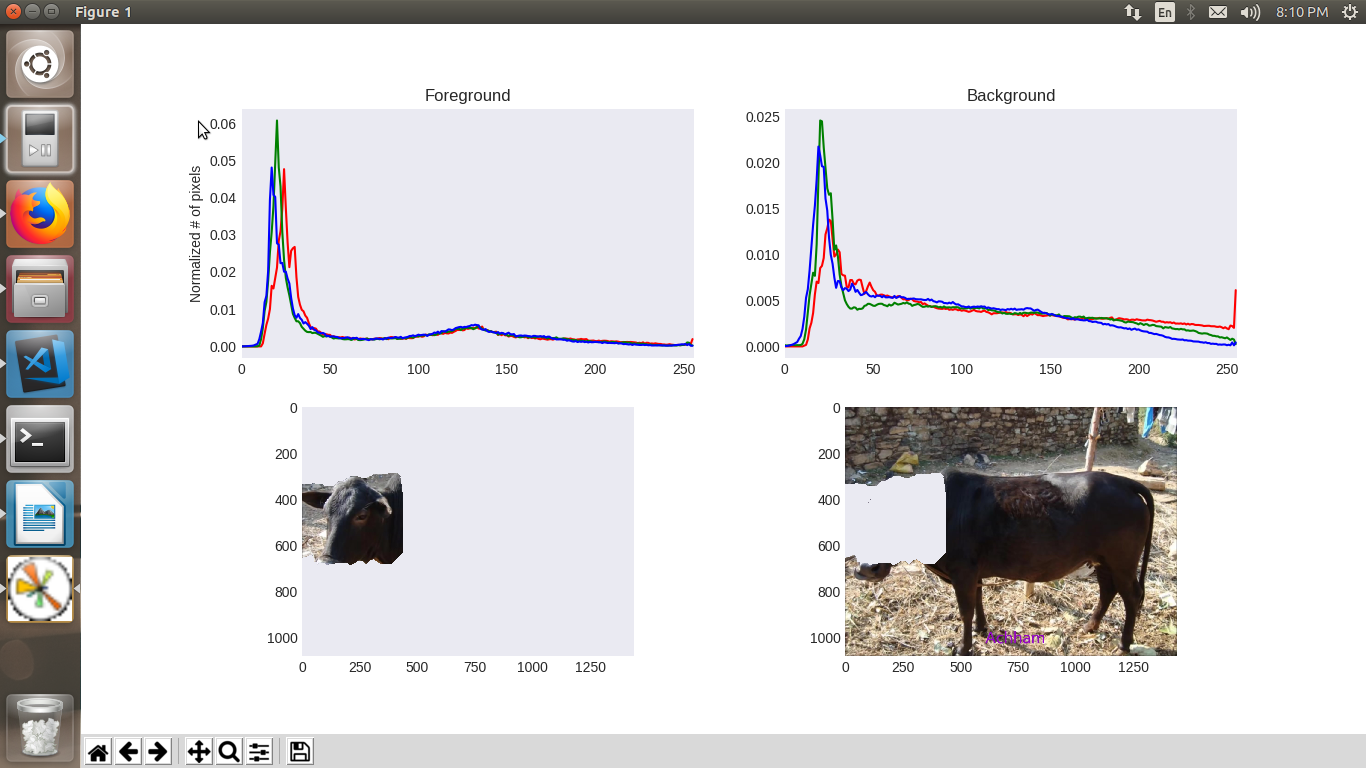


Figure 5: (Top) RGB color histograms of the foreground and background images after grabCut() has been applied.

**Algorithms and Techniques**

At this point, we’ve provided training data by scraping breeds and images, drawing rectangles around animal in images, using the OpenCV grabCut() function to separate the breed from the background, and extracting Haralick features and color histograms from the breed images (possibly transforming features via PCA). Machine learning algorithms (SVM, kNN, and RandomForest) are trained on the training data, using GridSearch to optimize the C and gamma parameters, using three folds for cross-validation. Typically, at least six images from each breed will be used for testing the classifier. For un-segmented test images, we will scale the image to a standard size if it is too large, then calculate Haralick features of center 60% of the image. If the algorithm was trained on PCA-reduced data, the features of the test images also must be transformed. Once we have the features ready, we can easily use the ‘predict’ method of the machine learning algorithm to classify the breed of the breed in the image.

One of the defining features of a animal breed is how it’s coat looks. Part of this is color, and part of it is texture. Here, we are using the texture as an indicator of breed. An SVM for our classifier is appropriate here because we are doing supervised learning with multiple classes. Using the RBF kernel allows us to split the data non-linearly, which should be helpful here, due to the overlap in the feature data. kNN also could be appropriate, because we would expect the training data to cluster around centers.

Finally, random forests are also appropriate, because the data can be separated based on many splits. Due to the large number of classes, we’d expect some overlap in many of the features. Some classes may have many overlapping features, with only a few small distinguishing traits. Random forests are a great choice for this scenario, because they can split these classes with mostly overlapping features near the bottom of decision trees.

**Improvement**

The color properties in the features could have spatial encodings embedded in them. For example, breaking up the image into subsections, and extracting the color histogram principal components from those areas, or just average RGB (or other color space) values may help discriminate animals with spots from those without.

Another improvement could be using a bag-of-words type model with the texture features, as well as color. The training images of animals could be broken up into subsections, and features extracted on each subsection. Then, kMeans classification would be performed with the subsection features. For an un-classified image, the image would be broken up into subsections, and features extracted from each subsection. Then, the kMeans classifier would find the closest breed for each subsection, ordering or weighting the subsections by their Euclidean distance to the nearest breed in the kMeans clusters.

However, I think with complex data and a plethora of classes, a neural network may be best suited for the problem. In the future, I would like to utilize a neural network for this problem, and utilize key point detection with a bag-of-words-based model.