Hotdog Or Not

Mostafa Atmar, Patrick Cockril, Riley Russell

May 9, 2023

`https://github.com/riley76865/cs482-final-project`

Motivation

Based on the famous skit from the HBO comedy series *Silicon Valley* where a gag app was used to classify objects as either being a hot dog or not a hot dog, we took this approach into our own hands and attempted to create a binary classifier that would emulate this and then compared it to the results of an another already built high-efficency classifier that made extensive use of machine learning libraries. Those who have watched the series may get a kick out of this, but to those whom the novelty falls short, this is an excellent way of demonstrating SIFT[1], a widely-used and robust feature extraction algorithm as well as experience in working with Bag of Visual Words (BoVW)[2], which is a popular approach for representing images in a fixed-size feature vector by creating a visual vocabulary. In addition to the sources above, some more examples of these same technologies being applied to image classification can be found here[3], here[4], and here[5].

Theory, Metric Definition, and Background Information

Within our codebase, there are a few classifers that are involved. The first of which (`bovw_svc.ipynb`) we built ourselves with minor guidance from the professor's lectures. We used support vector machines (SVM) as our method of performing image classification based on the Bag of Visual Words model. SIFT (Scale-Invariant Feature Transform) is an algorithm used to detect and describe local features in

[5] `https://www.cs.ubc.ca/~lowe/papers/iccv99.pdf`

[5] `https://openaccess.thecvf.com/content/CVPR2021/papers/Gidaris_OBoW_Online_Bag-of-Visual-Words_Generation_for_Self-Supervised_Learning_CVPR_2021_paper.pdf`

[5] `https://tinyurl.com/2rsm4c5u`

[5] `https://www.researchgate.net/profile/Jasmin-Velagic/publication/321412919_Aerial_image_mosaicing_approach_based_on_feature_matching/links/6167fd813851f9599400d03a/Aerial-image-mosaicing-approach-based-on-feature-matching.pdf`

[5] `https://www.researchgate.net/profile/Aini-Hussain/publication/229051043_Feature_Extraction_Technique_Using_SIFT_Keypoint_Descriptors/links/55e0874d08aede0b572e7404/Feature-Extraction-Technique-Using-SIFT-Keypoint-Descriptors.pdf`

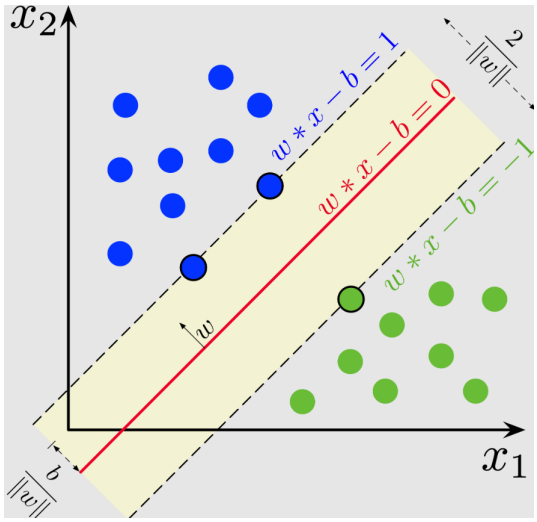


Figure 1: SVC

images, in this case, what makes a hotdog a ...hotdog. Finally, we used K-Nearest Neighbors (KNN) in the form of K-Means clustering to build a visual vocabulary.

The metric that will be used to determine the classification is accuracy via Support Vector Classifier (SVC). Accuracy is the ratio of the number of correct predictions to the total number of predictions. The equation for accuracy is as follows:

Accuracy = (Number of correct predictions) / (Total number of predictions)

In Figure 1, we see an example of a SVM trained with samples from two classes. The hyperplane splits them down the middle, with samples on either side of the margin being the support vectors. This mirrors our approach since we are using a binary classifier to filter the images.
We obtained our image dataset, which consists of 4000 hotdogs and other objects from Kaggle[6], and the dataset actually had this exact goal of identifying hotdogs as we did referencing the show.

Approach, Implementation, and Results

To take out the complexity of the tutorial, we will present the code in pseudocode, so it can be done in any language as well as being easier to read. The full python code can be found in the GitHub link at the top under `bovw_svc.ipynb`

Before we begin, you will have to install several libraries, namely `numpy, opencv-python,` and `scikit-learn`. We figured these are one of the best general-use libraries for machine learning.

1. Install the required libraries and import them into your code. Here's a simple

[6] `https://www.kaggle.com/datasets/thedatasith/hotdog-nothotdog`

command that can do that:
`pip install numpy opencv-python scikit-learn`

2. Load the dataset and extract SIFT features: This step defines two functions:
a. `load_images_labels(image_paths, categories)`: This function takes in a dictionary of image paths organized by category and loads the images into memory. It also assigns labels to the images based on their category.
b. `extract_sift_features(images)`: This function takes in the loaded images and uses the SIFT library to extract patches from them. It returns the keypoints and descriptors for all images.

```
function load_images_labels(image_paths, categories):
    for each category in categories:
        load images from image_paths[category] and append to images list
        append category as label to labels list
    return images, labels

function extract_sift_features(images):
    initialize SIFT detector
    for each image in images:
        compute SIFT keypoints and descriptors
        append keypoints to keypoints list
        append descriptors to descriptors list if not None
    return keypoints, descriptors
```

3. Build a visual vocabulary using KMeans: `The build_vocab(descriptors, k)` function takes in the SIFT descriptors and the desired number of clusters `k` (You will want to make this a lot smaller than 100). It applies the KMeans clustering algorithm to create the visual vocabulary by grouping similar features together.

```
    def build_visual_vocabulary(descriptors, k=100):
    kmeans = KMeans(n_clusters=k)
    kmeans.fit(descriptors)
    return kmeans
```

4. Extract Bag of Words features: The `extract_bovw_features(images, keypoints, visual_vocabulary)` function computes the Bag of Words representation for each image. It takes in the images, their keypoints, and the visual vocabulary generated in the previous step. For each image, it calculates the histogram of visual words, which serves as the Bag of Words feature representation.

```
function extract_bow_features(images, keypoints, visual_vocabulary):
    for each image and its corresponding keypoints:
        compute descriptor for the keypoints
        predict cluster labels for descriptors using visual_vocabulary
        compute histogram of cluster labels
        append histogram to bow_features list
    return bow_features
```

5. Train and evaluate an SVM classifier: The `train(bow_features, labels)` function takes in the Bag of Words features and their corresponding labels. It splits the dataset into training and testing subsets, scales the features using StandardScaler, and trains an SVM classifier with a linear kernel. After training, it evaluates the classifier on the test set and computes the accuracy.

```
function train_svm_classifier(bow_features, labels):
    split bow_features and labels into train and test sets
    fit and transform StandardScaler on train set
    transform test set with StandardScaler
    initialize and train SVC with linear kernel and C=1 on train set
    predict labels for test set
    compute accuracy between true and predicted labels
    return SVC model, StandardScaler, accuracy

```

6. Finally we load the images and their labels, extracts SIFT features, and creates the visual vocabulary using KMeans clustering. Then, it computes the Bag of Words features for each image and trains an SVM classifier using those features. Finally, it evaluates the classifier's performance and prints the accuracy. I have shown this part in Python so it is more clear what is being executed.

```
# directories for image classes
path1 = './images/hotdog/'
path2 = './images/notdog/'
hotdogs,notdogs = [],[]
# add each image in dir to list
for img in os.listdir(path1): hotdogs.append(path1+img)
for img in os.listdir(path2): notdogs.append(path2+img)

# set labels for each class
image_paths = {0:hotdogs,1:notdogs}
categories = list(image_paths.keys())

images, labels = load_images_labels(image_paths,categories)
```

```
visual_vocab = build_vocab(descriptors, K=2)

bovw_features = extract_bovw_features(images, keypoints, visual_vocab)
svm,scaler,accuracy = train(bovw_features, labels)
print(f'Accuracy: {accuracy*100:.2f}%')
```

Once you have implemented the above code and run it, you should recieve a very simple output.

Accuracy: 49.35%

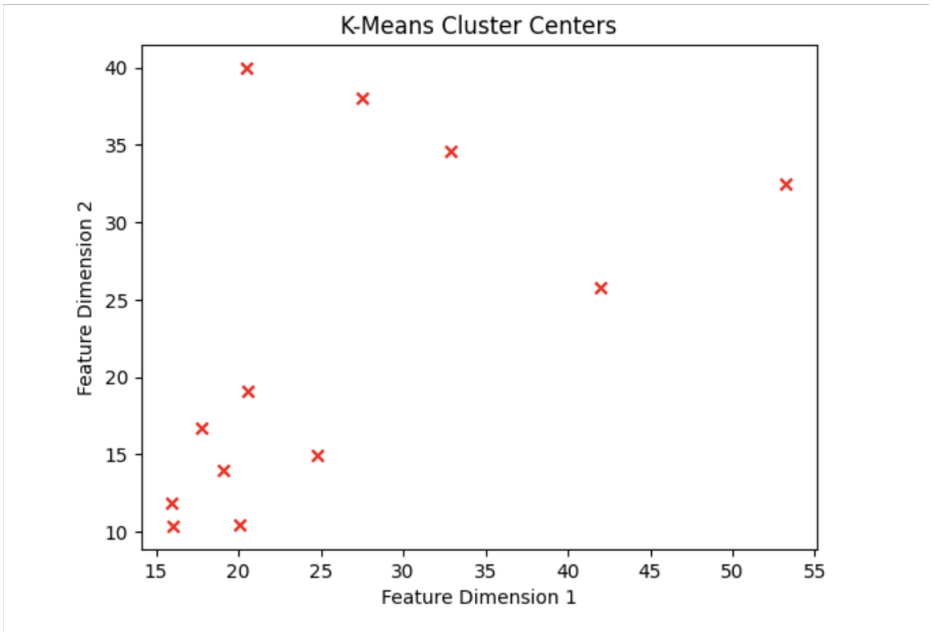
Now, we can compare this to the classifier that we found online (`bovw_knn.ipynb`)[7]. This code uses an alternate classifier via KNN, and makes use of various libraries such as `CV2, matplotlib, etc` and was purposely built to classify images. Running it will show a very impressive figure:

```
Average accuracy: %99.5

Class-based accuracies:

.DS_Store : %0.0
hotdog : %99.0
notdog : %100.0
```

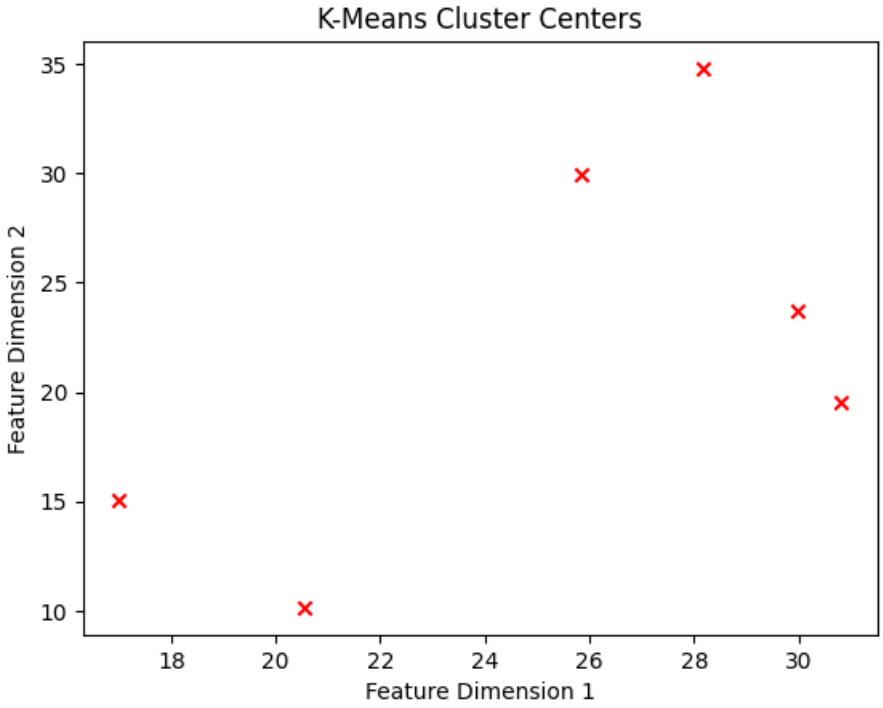
As you can see, this classifier is near perfect! Though our efforts aren't too off, we can correctly classify a hotdog about half the time.

The creator of that classifier even wrote a plot chart that tracks clusters given a k value:

[7] `https://medium.com/@aybukeyalcinerr/bag-of-visual-words-bovw-db9500331b2f`



k=12



k=6

The top chart shows when we build the BoW with a k of 12. It clusters towards

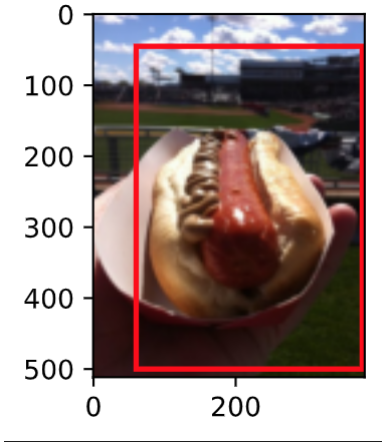


Figure 2: An example of our bounds detecting a hot dog

the bottom left, so perhaps give our dataset, many of the images have a hotdog (or another object) facing diagonally away from that corner. However, running with a lower k value does not have this same result curiously.

Clarity, Figures, and Completeness of Tutorial

From what we can discern from this experiment, we can build a classifier from the ground up and achieve a 50% identification rate. There's certainly room for improvement, but we figured that our simple collection of libraries were the main bottleneck in performance. The performance could potentially be improved by employing more advanced image processing techniques and feature extraction methods available in libraries such as OpenCV (which we already used for SIFT) and deep learning frameworks like TensorFlow. Leveraging deep learning models such as Convolutional Neural Networks (CNNs) for feature extraction or even end-to-end image classification could yield better results.