Capstone Project

Machine Learning Engineer Nanodegree

Mariko Wakabayashi

May 16, 2016

Definition

Project Overview

State Farm is interested in alleviating the problem of car accidents caused by distracted drivers. According to the CDC motor vehicle safety division, one of five car accidents are caused by distracted drivers. Every year, distracted driving is causing 425,000 injuries and 3,000 deaths. State Farm's goal is to increase driver's safety and insure their customers by testing to see if a dashboard camera can accurately detect drivers engaging in distracted behaviors.

State Farm created a computer vision competition on Kaggle, a platform that provides data science projects and company sponsored competitions. The company is challenging competitors to classify the driver's behavior.

State Farm provided a dataset of driver images where each image is taken inside a car with a driver engaging in some activity such as texting, eating, and etc. The given dataset is in csv format. The following files/folder were given to tackle this problem:

- sample_submission.csv - sample submission file in the correct format
- imgs.zip - zipped folder of test and train images
- driver_imgs_list.csv - a list of training images' filenames, their subject (driver) id, and class id

The types of activities State Farm wants the competitors to predict the likelihood is listed below:

- c0: safe driving
- c1: texting - right
- c2: talking on the phone - right
- c3: texting - left
- c4: talking on the phone - left
- c5: operating the radio
- c6: drinking
- c7: reaching behind
- c8: hair and makeup
- c9: talking to passenger

Problem Statement

The goal of this project is to implement and train a classifier that can predict the likelihood of what the driver is doing in each picture with an accuracy of over 90%. Using computer vision and

machine learning methods, I will give each image scores with range [0-1] for each behavior listed above.

To get a better idea of how we can tackle this problem, I looked into several research papers that have dealt with human activity recognition from still images.

This article called "A Review of Human Activity Recognition Methods" provides an overview of the techniques that have been applied to perform human activity recognition in still images or videos. It introduced me to several papers relevant to my goal. One paper presents a survey on still image based human action recognition. Through this paper, I've learned that many researchers used high level cues for still image-based action recognition such as the human body, body parts, action-related objects, human object interaction and the whole scene or context. These cues can be helpful in defining the different types of human actions. In our case, we would like to focus more on the human object interaction.

There are four popular low level features used by researchers in tackling action recognition.

- Dense Scale Invariant Feature Transform (DSIFT)
  - Dense sampling of a grayscale image to extract low level features for action analysis. This feature has been applied in problems such as object recognition, gesture recognition, navigation etc.
  - By extracting DSIFT features from many images, you can apply clustering to reduce the amount of information in obtaining a limited number of "keywords". Then create a histogram as a feature for each image.
- Histogram of oriented gradients (HOG)
  - Counts occurrences of discretized gradient orientations with a local image patch.
  - Used for still-image object detection and human detection.
- Shape Context (SC)
  - Help detect and segment the human contour.
  - Crucial for high level cue representation of human body silhouettes for action recognition.
- GIST
  - Computes a set of holistic, spatial properties of the scene. An Abstract representation of the scene that spontaneously activates memory representations of scene categories.
  - Mainly used to integrate scene or background information.

By analyzing the researcher's techniques, the paper observes that SIFT and HOG features are used in most existing approaches to action recognition in still images.

I. Proposed Solution
   A. Analyze DSIFT and HOG features
      1. I believe these two approaches are most appropriate for this particular problem. We will not need SC for human contour detection. SC will not give us much information because all the images consist of a driver in a sedentary posture. Also GIST will not be much of help because it focuses on obtaining the background information. In our case, the background is always the same. What's most important for us is to use a feature that can extract detail about the human's interaction with an object and here SIFT and HOG can be helpful.
   B. Choose a descriptor to extract from images.
      1. In this part, I may have to use PCA to reduce the dimensionality of the features I extract from a large set of images.

        2. I would like to find the most efficient and effective feature here that retains most of the image's information.
    C. Implement classifiers
        1. I hope to implement three classifiers most appropriate for this problem.
    D. Train and test classifiers
        1. Using the extracted features of all the training images, I like to train each classifier, and test to see which one does a better job.
    E. Improve the classifiers and decide on one classifier
        1. Using sklearn's GridSearch function, I would like to improve the classifiers and pick the one that does the best.

I. Outline of tasks
    A. Analyze the data
    B. Extract feature of couple of images with DSIFT
    C. Extract feature of couple of images with HOG
    D. If the features of all the training images are too large, find ways to reduce dimensionality of data
    E. Pick the descriptor that is most efficient and effective
    F. Implement SVM, RandomForestClassifier, and Logistic Regression
    G. Train and test each classifier
    H. Plot confusion matrix and improve the classifiers with GridSearch

Metrics

To evaluate the image classification accuracy, I will perform the following tasks.

- Compute accuracy score
- Create confusion matrix
- Produce precision, recall and f1 score

Accuracy Score - With sklearn, you can compute the accuracy of the classifier with sklearn.metrics.accuracy_score. This will give you an idea of how the classifier is doing. It is simply telling you how many correct predictions the classifier made. However, it will not tell me the details of misclassification, thus we will have to take some extra steps to evaluating the classifier.

Confusion matrix is great in that you can visualize the detailed performance of a supervised learning model with a table layout. It is specifically useful for multi category classifications in which each column represents the instances in a predicted class while each row represents the instances in an actual class. With this visualization one can see clearly if the system is mislabeling instances or "confusing" two or more categories. A perfect model will show a confusion matrix with all the answers in the diagonal of the table. Thus with a confusion matrix, one can easily detect the errors of the model when there are values outside of the diagonal.

For a more detailed classification performance analysis in numerical representation, precision, recall and f1 score will do the job well.
Precision is a measurement of the classifier's ability in preventing false positive labeling.
Recall is a measurement of the classifier's ability in finding all positive instances.
F1 score is the weighted average of the precision and recall. The score reaches its best value at 1 and worst at 0.

Analysis

Data Exploration

The dataset consists of a “imgs” folder that has a test and train folder of 640 x 480 jpg files. The images were taken by a dashboard camera. Each image consists of a driver performing a task from one of the distracted tasks. There are no duplicate images in the dataset. Also State Farm removed metadata from each image (e.g. creation dates). State Farm set up these experiments in a controlled environment. While performing each task, the drivers were not driving as a truck dragged the car on the streets.

Below I listed the number of files for each category in the training data and the test data.

c0 Safe driving	c1 Texting - right	c2 Talking on the phone - right	c3 Texting - left	c4 Talking on the phone - left	c5 Operating the radio	c6 Drinking	C7 Reaching behind	c8 Hair and makeup	c9 Talking to passenger
2490	2267	2317	2346	2326	2312	2325	2002	1911	2129

In total, there are 22,424 training examples. “Safe driving” has the most examples, and “Hair and makeup” the least. It makes sense to have a lot of “safe driving“ examples because State farm is in general interested in finding out if the driver is driving safely or not. They would rather have false negatives than false positives when labeling safe driving. More examples would improve the classifier’s performance in decreasing false positives for safe driving. “Hair and makeup” may be less because it is a task mostly performed by women.

Testing data	79726 files

There is also a csv file that lists training images’ filenames, their subject (driver) id, and class id.

Drivers	26 drivers

It may be hard to accurately train a classifier with 26 drivers due to the differences in driver’s physical traits, especially if the dataset is relatively small for each driver. However, in the end we can create a robust classifier that will make good predictions on people with different sizes, gender etc.

```
Analyzing subject p081
       c0   c1   c2   c3   c4   c5   c6   c7   c8   c9
p002   76   74   86   79   84   76   83   72   44   51
p012   84   95   91   89   97   96   75   72   62   62
p014  100  103  100  100  103  102  101   77   38   52
p015   79   85   88   94  101  101   99   81   86   61
p016  111  102  101  128  104  104  108  101   99  120
p021  135  131  127  128  132  130  126   98   99  131
p022  129  129  128  129  130  130  131   98   98  131
p024  130  129  128  130  129  131  129  101   99  120
p026  130  129  130  131  126  130  128   97   97   98
p035   94   81   88   89   89   89   94   87   56   81
p039   65   63   70   65   62   64   63   64   70   65
p041   60   64   60   60   60   61   61   61   59   59
p042   59   59   60   59   58   59   59   59   59   60
p045   75   75   76   75   75   76   71   67   66   68
p047   80   91   81   86   82   87   81   82   82   83
p049   84   85  119  110  109  116  119   74   79  116
p050  123   45   52   98   83   91   82   81   65   70
p051  182   81   81   83   81   83   95   80   62   92
p052   72   71   84   75   72   72   77   71   71   75
p056   81   80   80   78   82   81   80   74   83   75
p061   84   81   81   83   79   81   80   79   81   80
p064   83   81   83   84   86   85   82   79   81   76
p066  129  100  106  101  102  101  105   86  114   90
p072   63   62   36   31   34    6   35    2   21   56
p075   81   81   85   79   89   79   82   82   79   77
p081  100   90   96   82   77   81   79   77   61   80
```

Figure (1) Table of number of images for each category for each driver

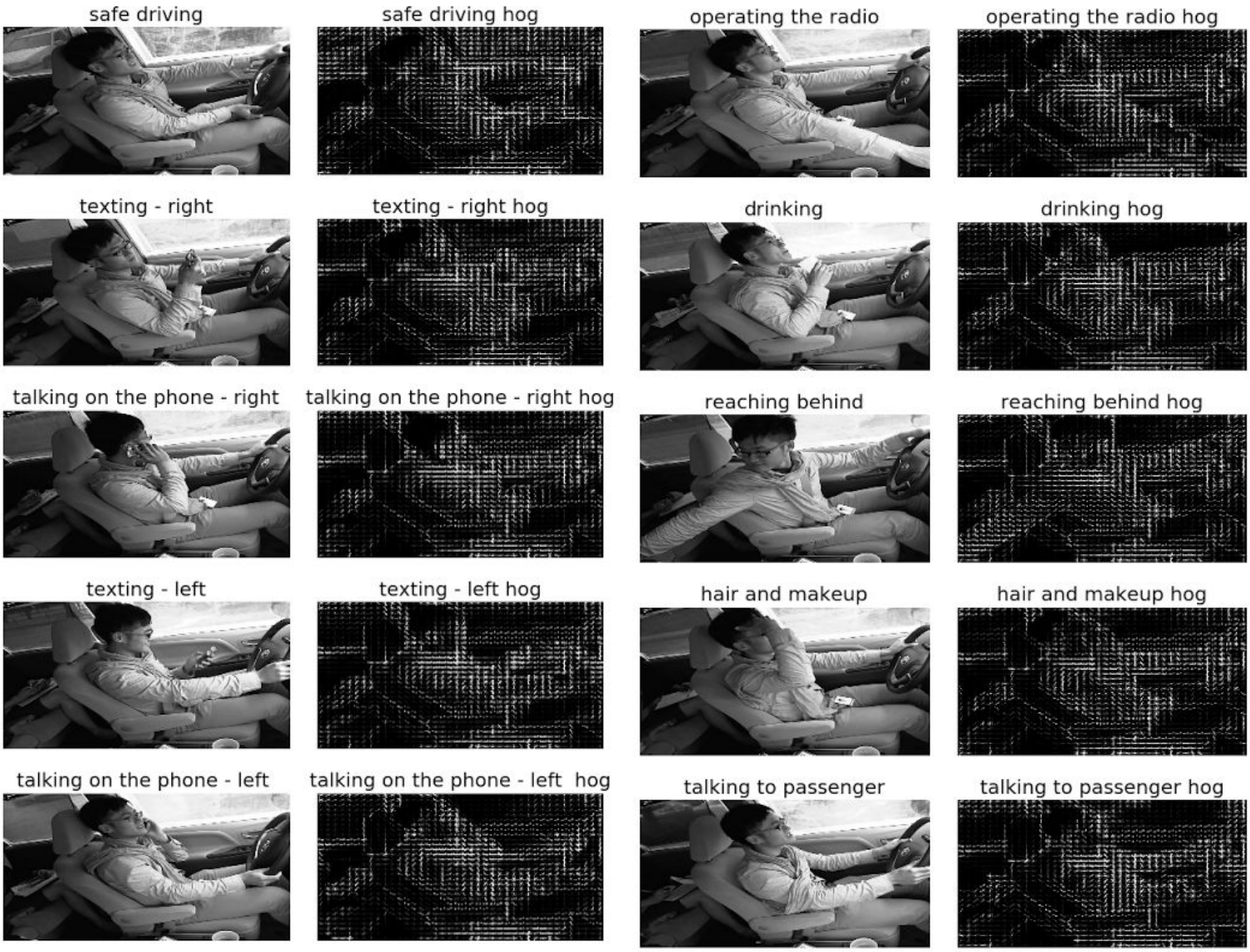
```
              c0          c1          c2          c3          c4          c5  \
mean   95.730769   87.192308   89.115385   90.230769   89.461538   88.923077
std    29.747683   22.847353   24.020536   24.826289   23.839850   27.161625
min    59.000000   45.000000   36.000000   31.000000   34.000000    6.000000
max   182.000000  131.000000  130.000000  131.000000  132.000000  131.000000

              c6          c7          c8          c9
mean   89.423077   77.000000   73.500000   81.884615
std    24.100080   19.279004   21.369605   24.045502
min    35.000000    2.000000   21.000000   51.000000
max   131.000000  101.000000  114.000000  131.000000
```

Figure (2) Table of summary statistics of Figure (1) table

As you can see, most of the drivers performed each task equally. However, we can see some outliers here. For example, subject p072 has quite a variance in the images that were taken for each category.

The subject has only 6 images for c5 (Operating the radio) and 2 images for c7 (Reaching behind) whereas for c0 (Safe driving), there are 63 images. When training the classifier, I may experiment with omitting data from subject p072 to see if it affects the performance.

Exploratory Visualization



HOG Descriptor

I extracted the HOG feature from one image from each category for one subject. The idea here is for me to find out if the HOG feature will do a good job in distinguishing the activity in each image. As you can see, the HOG feature is basically dividing up the image into square grid cells, computing the histogram of oriented gradients in each cell, normalizing the result using a block-wise pattern and finally returning a descriptor for each cell. HOG features are widely used for object or human detection.

Observing what I have here above, I can see that the HOG feature is great in capturing the silhouette of the driver. However it may not be the most efficient in extracting the differences in each category. In all of these images, the driver is most likely in a sedentary position. This feature will be redundant and memory inefficient when it is extracting information from every pixel of the image.

The HOG descriptor may not be as helpful in extracting the human to object interaction information. Also based on the analysis I did above, the feature vector length of each image is 9,600. This may take up space very quickly when processing couple ten thousands of images.

SURF

Safe Driving

Total matches: 2052
Selected matches: 340
Mean distance: 33.97

Texting Right

Total matches: 1936
Selected matches: 365
Mean distance: 34.37

Talking on the phone right

Total matches: 1810
Selected matches: 380
Mean distance: 31.78

Texting Left

Total matches: 1872
Selected matches: 341
Mean distance: 32.15

Talking on the phone left

Total matches: 1812
Selected matches: 280
Mean distance: 33.15

Operating the radio

Total matches: 1890
Selected matches: 406
Mean distance: 28.38

Drinking

Total matches: 1731
Selected matches: 288
Mean distance: 33.82

Safe Driving

Total matches: 2022
Selected matches: 337
Mean distance: 35.81

Hair and makeup

Total matches: 1901
Selected matches: 240
Mean distance: 36.69

Talking to passenger

Total matches: 1815
Selected matches: 375
Mean distance: 30.66

Next, I extracted the SURF feature from two images from each category for one subject. Through further research, I decided to analyze the SURF feature instead of the SIFT or DSIFT. SIFT is a common feature detector and descriptor used by many researchers studying object detection. However, SURF is known to be several times faster and more robust against different image transformations than SIFT.

Above, I only analyzed the matches that is within the threshold of half the distance mean to see whether the matchings are accurate. By analyzing several examples, each image has about 2000 keypoints. This is a significant reduction of description compared to the HOG which had about four times more information for each image. This will take less memory in overall computation. We may have to perform some clustering to further reduce the dimensionality of data.

Algorithms and Techniques

Feature Set

Instead of flattening an image into an array and using that as a feature for the classifier, we would like to perform certain algorithms on the image that will provide the best information in the smallest amount of space. In order to do that, we will have to extract interesting key points in the images and reduce the dimensionality of data. I listed the following features I would like to use to create the best feature set.

1. **SURF feature/Dense SURF Feature**

   Algorithm inspired by SIFT, SURF which stands for Speeded up robust features is a local feature detector and descriptor used for object detection.I would like to use SURF because it is several times faster and more robust against different image transformations than SIFT.

   The mahotas library provides the surf and the dense surf function. The dense surf function is only different in that it computes descriptors densely for every pixel in the image. It is helpful for image registration, pose estimation and object detection. The only parameters I will be tweaking for the dense function is the spacing between the points. I will retrieve the descriptors for each image by resizing the image, converting the image into grayscale, and flattening into a numpy array. The function will take the flattened image, run SURF detection and descriptor computations, and return a 64-element descriptors.

2. **Haralicks Feature**

   Haralicks Algorithm will compute the texture of an image. Texture of an image can be one of the most important characteristics used in identifying objects.

   The mahotas library provides the haralicks function. The input for this function will be a grayscale flattened image, and the output will be a 4x13 feature vector.

3. **Linear Binary Patterns**

   Linear Binary Patterns is a powerful feature for texture classification. Combination of LBP and a HOG descriptor is known to improve object detection performance.

   The mahotas library provides a Linear Binary Patterns function. Some parameters to tweak are the radius size and the number of points to consider. Given an grayscaled image, radius size, and points number, the function will return a histogram of features.

Reducing the Dimensionality of Data

1. MiniBatchKMeans

SURF features can take up quite a lot of space when processing couple of ten thousands of images. Reducing the dimensionality of the feature data will reduce the computation time and space and avoid the curse of dimensionality. MiniBatchKMeans is a variant of the K Means algorithm. I would like to use MiniBatchKMeans over regular K Means because it is faster using mini batches to reduce the computation time. The only disadvantage here is that MiniBatchKMeans perform a little worse than K Means.

Scikit-learn library provides a Mini batch K Means function. Some parameters I will be tweaking to get the best result is the number of clusters, size of mini batch, and number of random initializations. The MiniBatchKMeans will be fitted with an array of surf descriptors. Then the algorithm can predict which group a specific surf descriptor will fit in and create a histogram.

Model Algorithms

1. **Logistic Regression**

Logistic Regression is robust to noise and avoids overfitting. The clear advantage to using Logistic Regression is that the output can be interpreted as a probability. This is good because I will need a ranking more than a classification for solving this problem.

I will be using scikit-learn's logistic regression function. For the solver, I chose the 'lbfgs' setting. I chose the setting 'multinomial' for the multi_class parameter in order to solve the probability distribution. Before training this classifier, I would like to use pipeline to preprocess data first and fit to the classifier. I will be using the StandardScaler algorithm to standardize the dataset by performing a zero mean and unit variance. By fitting the classifier with the features and label, the classifier can predict label for new data, calculating the estimates for all classes.

2. **One vs Rest with Random Forests**

There are many advantages to using random forest over logistic regression. One, it can handle non linear features, and second, random forest can work with very high dimensional spaces and large number of training examples. It is in other words...:

- Non parametric
- Capable of using continuous and categorical data sets,
- Not sensitive to overfitting
- Easy to parametrize
- Good at dealing with outliers in training data

The disadvantage may be that the large number of trees can slow down the algorithm's real time prediction.

I will use scikit-learn's random forest classifier function. Some parameters I will be tweaking are the number of decision trees in the forest and the number of features to consider when looking for the best split. Since the first approach is multinomial, I would like to train a one vs all classifier by training one RF classifier for each class.

Benchmark

We are benchmarking the performance of the models against the performance of an image classification taught in Building Machine Learning Systems with Python by Willi Richert and Luis Pedor Coelho. I am using the same features as this example, and the best performance the book was able to achieve was 67%. For the results, I am expecting a performance of 45-60%. In the book, the images are classified into four classes: cars, animals, transportation, and natural scenes. However, in my case, the problem will be a lot harder in that I will be classifying the images into 10 different classes. Also compared to the four classes mentioned above, the type of classes I will be classifying the images have

very little differences in images. The differences between a image of a car and a bee is easier to distinguish than the difference between a driver talking to a passenger and a driver that's not.

The training set includes already manually labeled image instances. We can compare each model's performance with the training data by estimating the accuracy using a Leave One Out cross validation or K Fold Cross Validation.

K Fold Cross Validation - K Fold(n, n_folds) : divides all the samples in k groups (folds) of samples. This validation creates k-1 training set and leave one out for test.

Leave One Out Cross Validation - LeaveOneOut(n) : cross validation technique provided by scikit-learn. The general idea is that if there are n samples, the validation will create n different training and n test sets. The only difference here with the K Fold is that Leave One Out CV will create n models from n samples instead of k models where k < n. Thus Leave One Out can be computationally expensive compared to K Fold if the dataset is very large. Also Leave One Out is known to have a high variance due to the overlap it has in each training set. However it is advantageous in that a 5 or 10 fold CV can overestimate the generalization error when the learning curve for a training set is very steep. Many researchers have suggested using 5 - 10 fold cross validation instead of LOO. I hope to use these two validation for testing my model. However if the dataset is too large, I will use 5 and 10 fold cross validation.

Methodology

Data Preprocessing

I had to preprocess data before using them to fit with the models. In total I experimented with three features to get the best classification. I created five classifiers with different combinations of features for each model to see what works best.

- 1st classifier: Haralicks
- 2nd classifier: Linear Binary Patterns
- 3rd classifier: Surf Descriptors
- 4th classifier: Haralicks and Linear Binary Patterns
- 5th classifier: Haralicks, Linear Binary Patterns, and Surf descriptors

Surf Feature/Dense SURF Feature

1. Get surf descriptor for each image

```
import mahotas as mh
from mahotas.features import surf
alldescriptors = []
for im in images:
        # alldescriptors.append(surf.dense(im, spacing=16))
        surf_desc = surf.surf(im, descriptor_only=True)
        alldescriptors.append(surf_desc)
```

For each image, we will compute the surf descriptors. The descriptor_only is set to True, the function will only return descriptors. By default, it will also return the descriptor's pixel location, size, and etc. With mahotas, we can also compute the dense surf descriptors. The above commented function will return descriptors computed on points that are at a distance of 16 pixels from each other.

For the surf function, the return value is a n by 64 array where n is the number of points sampled. We retrieve about 57,600 descriptors per image. For the dense surf function, the return value is a 600 by 64 array which is 38,400 descriptors per image.

2. Create bag of words model

```
concatenated = np.concatenate(alldescriptors)
concatenated = concatenated[::64] # use only every 64th vector
k = 256
km = MiniBatchKMeans(n_clusters=k, batch_size=20000, n_init=iterations)
km.fit(descriptors)
surf_descriptors = []
for d in alldescriptors:
        c = km.predict(d)
        surf_descriptors.append(np.bincount(c, minlength=k))
surf_descriptors = np.array(surf_descriptors, dtype=float)
```

The descriptors we collected cannot be directly fed into the models due to its dimensionality. We will reduce the dimensions by applying k means clustering to obtain centroids. For speed, we will not use all descriptors. We will then go through each descriptor and create feature vectors. Surf descriptors will contain histograms of each image. Each image is represented as a single array of features of the same size, in this case k = 256. With this, we can now feed it into our models.

Haralicks Feature

1. Compute haralicks texture for each image

```
import mahotas as mh
haralicks = []
for im in images:
        im = mh.colors.rgb2grey(mh.imread(im))
        im = im.astype(np.uint8)
        texture = mh.features.haralick(im).ravel()
        haralicks.append(texture)
```

Haralicks feature is a texture feature. It can distinguish images depending on the patterns. Using the mahotas library, I can call the haralick function for each image. The haralick function will return a 4 by 13 array. Each row represents a direction in which the features were computed. (up, down, left and right). Then we convert the array into a contiguous flattened array using the numpy ravel function. By doing this, we can feed into the classifier.

Linear Binary Patterns

1. Compute patterns for each image

```
import mahotas as mh
lbps = []
for im in images:
        im = mh.colors.rgb2grey(im)
        binary_patt =  lbp(im, radius=8, points=6)
        lbps.append(binary_patt)
```

Linear Binary Patterns is another texture feature that is robust to illumination changes. We can compute the pattern by using mahotas's lbp function. Giving a grayscale image, the function will return a histogram of feature counts where position x corresponds to the number of pixels that had code x.

The size of the radius is in pixels. Points is the number of points to consider. The histogram will be in the format of a 1-D numpy array.

Implementation

1. **Reading large dataset of images**
   a. Reading a large dataset can be an expensive operation especially if the data is stored in memory. For the final implementation I decided to use glob to retrieve an iterator of all the file names. There are two advantages to using Glob in this situation. One is that by retrieving the list of filenames in an iterator format, the data is not stored in a buffer or in memory, but can be read one at a time. Second advantage is that I can easily retrieve the filenames that I want by using regular expression. Since the files are already organized in training and test folder, glob made it easy to retrieve files that follow a certain pattern.

```
classes = ['c0','c1','c2','c3','c4','c5','c6','c7','c8','c9']
images = []
# Use glob to get all the train_images
if train:
        for i in classes:
                images += glob('{}/{}/{}/*.jpg'.format('imgs', 'train', i))

# Use glob to get all the test_images
else:
        images += glob('{}/{}/*.jpg'.format('imgs', 'test'))

images.sort()
```

2. **Preprocessing Data**
   a. Getting features

      For each image, I calculated the haralicks, linear binary patterns, and dense surf descriptors. In the methodology section, I explained in detail how I would obtain these features using the mahotas library. I mentioned how I can retrieve both surf and dense surf features. However, while retrieving the surf features for the testing data, I ran into a memory issue where the program was getting killed by the os. Retrieving 56,700 descriptors for each image was a very memory hungry process. I tried resizing the image, but the function was returning empty arrays for some images.

      The dense surf function is slower compared to the surf function, but it guarantees non empty return values and return the same number of descriptors for each image. Dense surf function is known to collect more features at each location and scale in an image, increasing recognition accuracy.

   b. Clustering

      To reduce the dimensionality of the surf descriptor data and create a bag of words models, I used scikit-learn's MiniBatchKMeans to perform faster clustering than regular K Means. For both test and train set I calculated k as..

      $k = \sqrt{\frac{n}{2}}$ *where n is the number of data points*

      ```
      km = MiniBatchKMeans(n_clusters=k, batch_size=1000, n_init=iterations)
      ```

      I set the batch size to 1000, iterations to 30, and use k_means++ method for the initialization. The algorithm is basically taking small batches of the dataset for each iteration. Then depending on the previous locations of the cluster centroids, it assigns a cluster to each data point in the batch. By adding new data points to the cluster, the algorithm updates the locations of the cluster centroids.

I used k_means++ because it is commonly used to estimate k intelligently. K-means++ has an additional step compared to the regular k-means by estimating the number and initial position of the centroids. Due to its fast convergence, the k_means++ is superior in both performance and accuracy than regular k_means.

3. **Training two models**
   a. Logistic Regression

```
classifier = Pipeline([
                        ('preproc', StandardScaler()),
                        ('log', LogisticRegression(solver="lbfgs", multi_class="multinomial"))
                        ])
```

I used scikit-learn's logistic regression function. For the solver, I chose the 'lbfgs' setting. I chose the setting 'multinomial' for the multi_class parameter in order to solve the probability distribution. I used the pipeline function to preprocess data first and fit to the classifier. StandardScaler algorithm is used to standardize the dataset by performing a zero mean and unit variance.

   b. One vs Rest Random Forest

```
classifier = [ RandomForestClassifier(n_jobs=-1,n_estimators=100) for i in classes ]
```

I used scikit-learn's random forest classifier function. The parameter n_jobs allows parallel construction of the trees and parallel computation of the predictions. I set the n_jobs to -1 to use all the available cores on my machine. I set the n_estimators, the number of trees in the forest to 100.

4. **Predicting labels for test data**

For both models, I am using the predict_proba function that returns the predicted class probabilities of the test data. Once I make the predictions for the test dataset, I write the data to a csv file in the format State Farm wants.

Refinement

1. **Refinement to reading large dataset of images**

Initially to keep track of the labels and the filenames I first created a python dictionary with the key as the filename and the value as the label. This was very inefficient in that I was storing a large amount of data in memory and wasting space for redundant information for storing classes as values.

For the final solution, I decided to use glob. With glob I am able to easily obtain the train and testing dataset based on the file path pattern and avoid storing data in memory. I can keep track of the labels and the images by iterating through one at a time. This has significantly reduced the computation time.

2. **Refinement to preprocessing data**

Initially when I was testing each model, I was recalculating the features everytime. Image processing especially when extracting three types of features from each image can be computationally expensive.

To speed up the process, I decided to use the pickle module for serializing and deserializing a Python data structure. Once the features are calculated for the whole dataset I will "pickle" or convert the features into a byte stream, and "unpickle" or convert back into an object for the next time I run the program. This is done in the get_obj and save_obj function below.

```
def get_obj(train, objectContent):
    if train:
        filename = 'objects/train/train_{}.obj'.format(objectContent)
```

```
    else:
        filename = 'objects/test/test_{}.obj'.format(objectContent)

    if os.path.exists(filename):
        print("Getting object %s" % filename)
        with open(filename, 'rb') as fp:
            return pickle.load(fp)
    else:
        return None

def save_obj(train, objectContent, obj):
    if train:
        filename = 'objects/train/train_{}.obj'.format(objectContent)
    else:
        filename = 'objects/test/test_{}.obj'.format(objectContent)

    with open(filename, 'wb') as fp:
        pickle.dump(obj, fp)
        print("Saved %s" % filename)
```

Using pickle has shortened the computation time significantly.

3. **Refinement to finding the best model with the best set of features**

Our goal is to find the classifier with the best accuracy score that can predict new values very well. Initially, the models were created and fitted with constant parameters. However for the final solution, I incorporated cross validation and grid search to find the best model with the best set of features.

a. Cross Validation

With scikit-learn's K Fold cross validation, we can divide the training data into k subsets, train on k-1 subsets and test on 1 subset. This can be repeated k times. The advantage to kfold is that every data points gets to be in the test set exactly once and train set k-1 times. As k is increased, the variance of the result is reduced. This method will produce a much lower variance compared to the holdout method. I did 5 and 10 fold cross validation for each model.

```
cv = cross_validation.KFold(n=len(features), n_folds=5, shuffle=False,random_state=None)
# 5 fold cross validation on dataset with length n
```

b. Grid Search

With scikit-learn's GridSearch function, you can select a wide range values for one or more specific parameters. Then the function will implement the specified classifier and fit on a dataset evaluating all the possible combinations of parameter values. The function will then retain the classifier with the best combination. With grid search, we can get the best score and parameter info for the specific model.

c. Logistic Regression

```
param_grid = {
        'n_estimators': [100, 700],
        'max_features': ['auto', 'sqrt', 'log2']
}
```

```
classifier = Pipeline([
                    ('preproc', StandardScaler()),
                    ('log', LogisticRegression(solver="lbfgs", multi_class="multinomial"))
                    ])
CV_pip = GridSearchCV(estimator=classifier, param_grid=param_grid_1, cv=cv)
CV_pip.fit(features, labels)
score = CV_pip.best_score_
best_parameters = CV_pip.best_params_
```

With gridsearch, I can select the range for n_estimators and types for max features. You can obtain attributes such as best_score_a and best_params_ from gridsearch to find out what worked out best.

d. One vs Rest with Random Forests

```
param_grid_1 = {'log__C': [0.001, 0.01, 0.1, 1, 10, 100, 1000] }
classifier = [RandomForestClassifier(n_jobs=-1,n_estimators=100) for i in classes]
CV_rfcs = [GridSearchCV(estimator=clf, param_grid=param_grid, cv=cv) for clf in classifier]

for i, clf in zip(classes, CV_rfcs):
        clf.fit(features, labels)
        score = clf.best_score_
        best_parameters = clf.best_params_
        scores.append(score)
        score = np.mean(scores)
```

With gridsearch, I can select the values to use for C (Inverse of regularization strength).

e. Comparing score between different combinations of features

Using cross validation and gridsearch, I can make a comparison and find the best model for this problem.

Results

Model Evaluation and Validation

Logistic Regression Performance with 5 fold cross validation

Set of Features	Best Parameters	Accuracy Score
Haralicks Feature	C = 1000	5.0%
Linear Binary Patterns	C = 100	4.1%
Dense Surf Descriptor	C = 100	19.6%
Haralicks and LBP	C = 100	10.4%
Haralicks, LBP and Surf	C = 1000	25.0%

Logistic Regression Performance with 10 fold cross validation

Set of Features	Best Parameters	Accuracy Score
Haralicks Feature	C = 10	9.1%
Linear Binary Patterns	C = 10	4.8%
Dense Surf Descriptor	C = 1000	37.3%
Haralicks and LBP	C = 100	14.4%
Haralicks, LBP and Surf	C = 100	47.9%

One vs Rest with Random Forests with 5 fold cross validation

Set of Features	Best Parameters	Accuracy Score
Haralicks Feature	Max_features = auto N_estimators = 700	22.6%
Linear Binary Patterns	Max_features = sqrt N_estimators = 700	15.9%
Dense Surf Descriptor	Max_features = sqrt N_estimators = 700	8.9%
Haralicks and LBP	Max_features = sqrt N_estimators = 700	24.8%
Haralicks, LBP and Surf	Max_features = sqrt N_estimators = 700	18.9%

One vs Rest with Random Forests with 10 fold cross validation

Set of Features	Best Parameters	Accuracy Score
Haralicks Feature	Max_features = sqrt N_estimators = 700	46.2%
Linear Binary Patterns	Max_features = log2 N_estimators = 700	30.5%
Dense Surf Descriptor	Max_features = auto N_estimators = 700	17.5%
Haralicks and LBP	Max_features = sqrt N_estimators = 700	50.2%
Haralicks, LBP and Surf	Max_features = sqrt N_estimators = 700	37.0%

Based on the score and speed I chose the Logistic Regression model with 10 fold cross validation and Haralicks, LBP and Surf as the feature set. The one vs rest with random forest does have a better score of 50.2% but training took much longer than the logistic regression model because I am training a list of 10 classifiers whereas LR is just one. The final model's score is within the score range I was expecting. Using Gridsearch, the best parameter of C= 100 is chosen. Like the image classification example from Building Machine Learning Systems with Python, the performance gradually increased as the features are combined. One feature by itself does not produce a good score. The model is tested well with 10 fold cross validation, and predicted over 75,000 unlabeled images.

As you can see the training score is not so great to rely on a dashboard camera to detect unsafe driving. However small changes to the variety of feature combinations increased the performance. With logistic regression, combining surf descriptors with haralicks and linear binary patterns gave the performance a 10.6% increase.

Justification

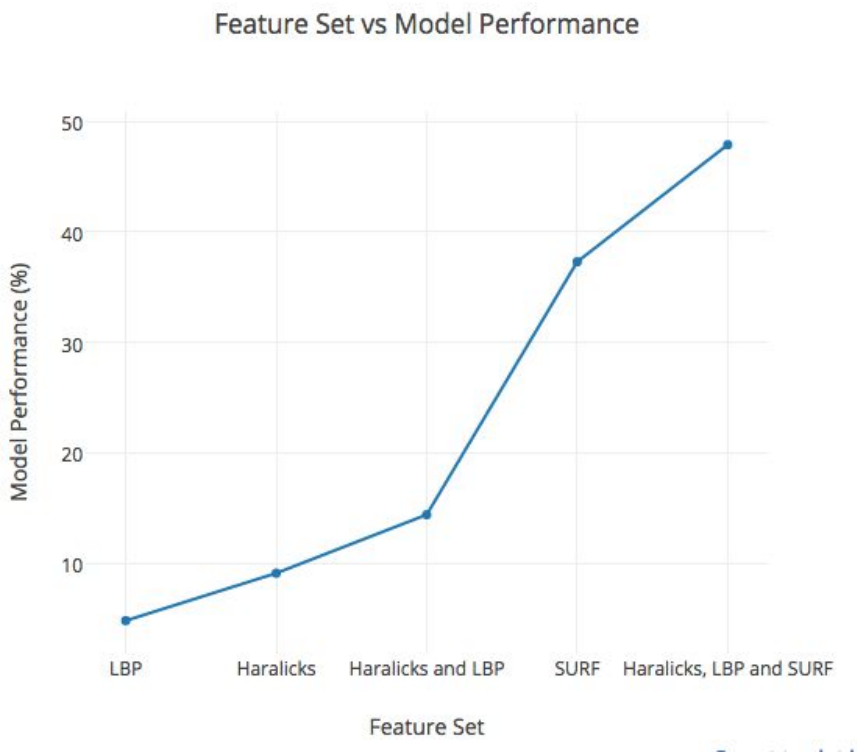
As I expected, the final results are not as strong as the benchmark result. Compared to the example done in Building Machine Learning Systems with Python, the problem is a lot more complex. The number of classes is twice the example's classes and the images especially the background is all very similar. All the images include a driver in a car performing certain activities. Unlike classifying the image into two completely different classes such as an animal or a vehicle, we are dealing with human activity classification where just a person opening their mouth or raising their arm in a particular angle can mean a certain activity.

The final solution is not significant enough to have a dashboard camera and a program to detect what the driver is doing. If State farm is trying to figure out whether a driver is driving safely or not, that would definitely been an easy binary classification compared to this multi class classification. But for State farm to understand what certain type of activity is popular or common or what is the root cause of all the accidents each year, multi classification is the solution.

Analyzing the results for each model and the parameter setting, I realized that for the random forest model, the best parameter for models I trained included estimators of 700. I tested the model with estimators of 100 through 700, and 700 being the maximum, if I chose a higher maximum, I might have gotten a better performance. But the higher the number of estimators, the slower the model will be trained. 700 is already too many according to this article, "How many Trees in a Random Forest?" where the recommended is 64-128.

Conclusion

Free-Form Visualization



Above is a line graph of the final model's performance depending on the feature set. The x axis displays the type of features used to create the classifier, and on the y-axis, shows the classifier's training score. As you can see, the model does best with the surf descriptor. Both Haralicks and linear binary patterns are texture features and as the results show, it does little to help with the classification alone. The image dataset includes a same setting with very similar activities but with different drivers. Texture will not be the main solution to classifying these images into ten classes but it can help improve the performance when combined with the surf descriptor. The surf descriptor will do much better than the texture feature since it detects and describes local features in the images.

Reflection

Overall as someone who only took an introductory course on computer vision in the past, this was a challenging problem. I have worked on image processing projects but this was my first time working on a very large image dataset and creating a classifier for image classification. I've only scratched the surface of object detection by working on this capstone project but I believe it was a great start for me to learn how these problems are solved.

The most interesting and informative part was learning all these different features extracted from images to use in detection. I found a great article that did a survey on all the techniques used by many researchers in the computer vision field. It has helped me carefully pick the best feature set to use for this problem. Also the book Building Machine Learning Systems with Python has helped me choose the python libraries and modules to use to extract features.

The most difficult part was working with the State-Farm dataset. It was my first time processing ten thousands of images. I've run into a lot of out of memory issues and maximized my CPU power

working on this project, but I found some great resources along the way to solve these issues. Modules such as glob and pickle has helped me solve the memory problems I faced while tackling this problem. I learned to optimize and create better more efficient programs working on a large data problem.

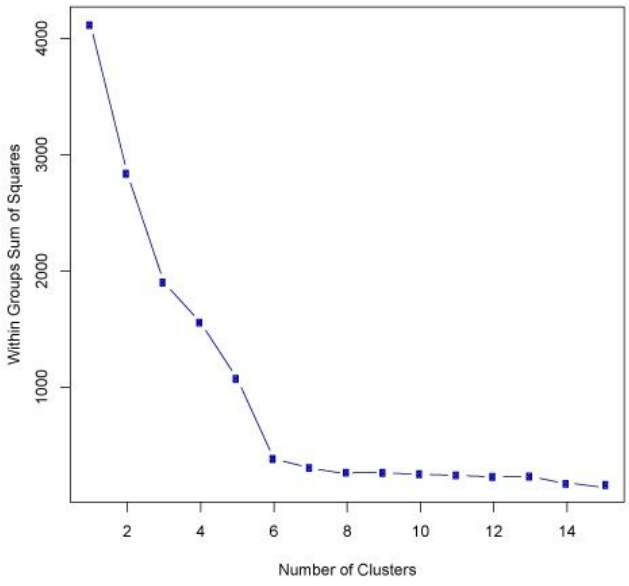
As a beginner of machine learning in computer vision, I think I did my best learning image features and trying them out on two models to compare performances. However, there are still many more that can be done to increase the current performance. The current solution is not quite finish for its low performance, but in a general setting, the features I used for this problem can do well in image classification. However, any image classification problem requires a trial and error process. Parameters and settings must be tweaked in order to create the best result for the specific problem.

Improvement

In this section, you will need to provide discussion as to how one aspect of the implementation you designed could be improved. As an example, consider ways your implementation can be made more general, and what would need to be modified. You do not need to make this improvement, but the potential solutions resulting from these changes are considered and compared/contrasted to your current solution.

Due to time restrictions and CPU power, I could not experiment with another model, but if I could I would have liked to try training a SVM model with RBF kernel. Towards the end of my project, I came across an academic paper that presents a SVM classifier that can accurately label hang gestures 82% of the time using Surf descriptors. The differences in each class in the state farm dataset is in the hand gesture, body position, interaction with an object and posture. The differences in hand gesture can be subtle and if the classifier can perform over 80%, I believe using this method with this problem will produce a better performance score than the current one.

Some other improvements I would have liked to make was the decision on choosing the best k for k means clustering. This time I roughly estimated K to be the square root of half the datapoints, but I would have liked to use a method called elbow method to estimate the best k for clustering.With elbow method, you calculate the sum of squared error for several values of k. When you plot these k values and its SSE, you can see a pattern of the error decreasing while the k increases. The k you should use is the point where the SSE decreases suddenly in the graph below. This graph produces a curve that looks like an elbow in the graph. In this case the best k is 6.



There is a method similar to the simple bag of words I created called Vector of Linearly Aggregated Descriptors that I would have like to use but did not know how to implement. VLAD assigns local descriptor to elements in a visual dictionary using K Means. Instead of storing the visual word occurrences only, these representations store statistics of the difference between dictionary elements and pooled local features.

An even better solution will definitely exist than the final solution I produced. I hope to make more improvements to the current system I have and create a classifier with an accuracy of over 80%. Before I do that I should upgrade my laptop.

Sources

1. "State Farm Distracted Driver Detection." *Kaggle: Your Home for Data Science*. Web. 1 Apr. 2016.
2. "A Review of Human Activity Recognition Methods." *Frontiers*. Web. 16 Apr. 2016.
3. "Documentation of Scikit-learn 0.17." *Documentation Scikit-learn: Machine Learning in Python — Scikit-learn 0.17.1 Documentation*. Web. 17 Apr. 2016.
4. Guo, G., and Lai, A. (2014). A survey on still image based human action recognition. *Pattern Recognit.* 47, 3343–3361. doi:10.1016/j.patcog.2014.04.018
5. "Histogram of Oriented Gradients." *Histogram of Oriented Gradients — Skimage V0.12dev Docs*. Web. 20 Apr. 2016.
6. "OpenCV: Introduction to SIFT (Scale-Invariant Feature Transform)." *OpenCV: Introduction to SIFT (Scale-Invariant Feature Transform)*. Web. 24 April 2016.
7. "Introduction to SURF (Speeded-Up Robust Features)." *Introduction to SURF (Speeded-Up Robust Features) — OpenCV 3.0.0-dev Documentation*. Web. 25 April 2016.
8. "Feature Matching." *Feature Matching — OpenCV 3.0.0-dev Documentation*. Web. 26 April 2016.
9. "Tutorial: Classification Using Mahotas." *Tutorial: Classification Using Mahotas — Mahotas 1.4.1 Git Documentation*. Web. 27 April 2016.
10. Richert, Willi, and Luis Pedro. Coelho. *Building Machine Learning Systems with Python: Master the Art of Machine Learning with Python and Build Effective Machine Learning Systems with This Intensive Hands-on Guide*. Print.
11. "Luispedro/BuildingMachineLearningSystemsWithPython." *GitHub*. Web. 27 April 2016.
12. "Determining the Number of Clusters in a Data Set." *Wikipedia*. Wikimedia Foundation. Web. 26 Apr. 2016.
13. "VLAD- An Extension of Bag of Words." *A Positronic Brain*. 2014. Web. 30 April 2016.
14. "VLFeat.org." *VLFeat*. Web. 1 May 2016.
15. Sykora, Peter, Patrik Kamencay, and Robert Hudec. "Comparison of SIFT and SURF Methods for Use on Hand Gesture Recognition Based on Depth Map." *AASRI Procedia* 9 (2014): 19-24. Web.
16. Oshiro, Thais Mayumi, Pedro Santoro Perez, and José Augusto Baranauskas. "How Many Trees in a Random Forest?" *Machine Learning and Data Mining in Pattern Recognition Lecture Notes in Computer Science* (2012): 154-68. Web.