Hotel Image Classification

A machine learning project

This report corresponds to Kaggle competition for machine learning problem of Hotel Image classification. The detail about design and implementation is elaborated in this report. This is individual project by Sachin Badgujar.

Hotel Image Classification

A machine learning project

Objective:

This real world problem deals with image classification. Images are of the hotel and need to be classified in one on the eight classes 'Bathroom', 'Guestroom', 'Pool', 'Gym', 'Restaurant', 'Lobby', 'Aerial View', 'Business Center'. The data set contains the training and testing images. The number of training images is 38,372 and number of testing images is 19,648. The code is implemented in python and used image processing, machine learning libraries like cv2 and Scikit.

Design:

Design depends on how good the features are and how you learn the features. Instead of considering only one feature, I have considered multiple features and then take votes of these classifications depending upon the how accurately that feature classifies the image. As given in proposal, the system extracts the features with the help of SIFT. These features are in the form of key and descriptor. All the descriptors are collected and formed K-clusters from it. This clusters will act like a dictionary of features and given the training images, we find out features of the training image. These features are pass through clusters and a histogram i.e. a bin count is made for every image. We can imagine this as- x-axis will be te number of clusters and y-axis will be the count of feature belonging to the particular cluster. This histogram of features is then trained with SVM. The Image on the next page will clear the idea of the system.

Image Classification

This project deals with interesting real world problem. The report tries to address all the efforts taken to solve the machine learning problem. The report might look lengthy but covers everything and it is cordial and easy to understand.

Implementation section deal with all validation and testing variations.

Real world images have real world problem. For example if the image is **corrupted** then what will be your prediction? This project treat these images as **equal probable** for every class. These probabilities are **programmatically** appended at the end of final CSV file.

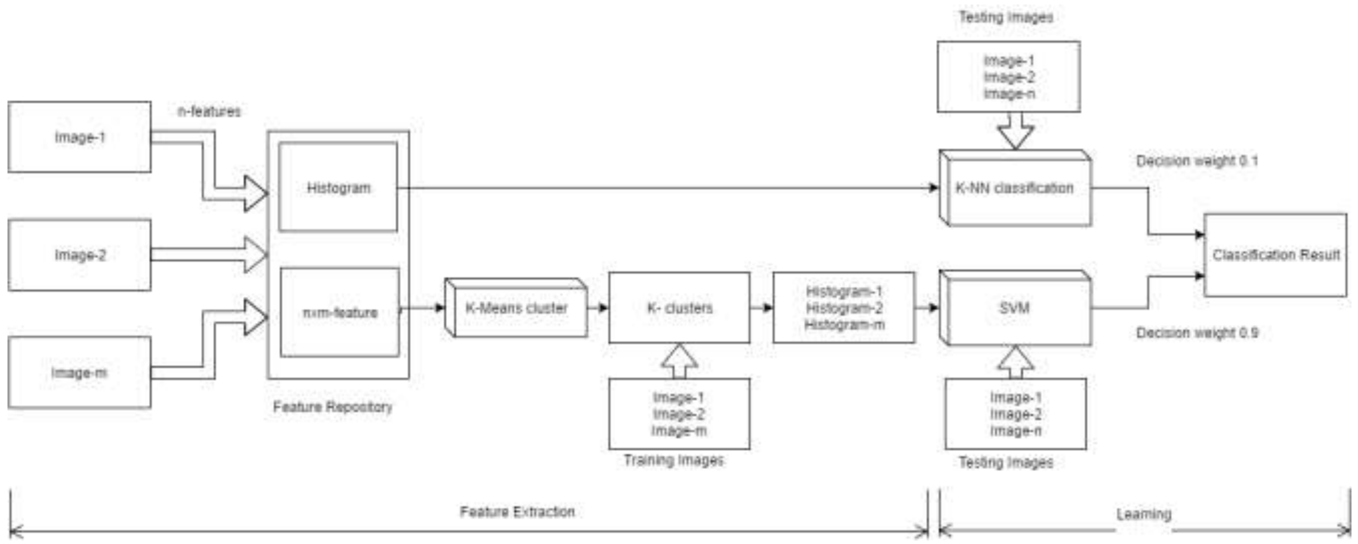


Fig 1: Hotel image classification system.

The system has **two** variants:

1. with gray histogram – KNN – weighted decision.
2. without gray histogram.

Both the system give an almost same performance.

Why not MLP?

Neural networks like MLP basically are not designed for classification and they need some modifications to use as classifier and they can learn sophisticated curves but they have their problems like over fitting and over parameters. In general we have following law in machine learning (Occam Razor principle) simplest model is the best model[2].

Implementation

Implementation is done in python with the help of Scikit-Learn package, cv2 bindings, and numpy package. As the size of training data is large and the system need to hold the million of bytes, the system is split into two part. The two parts are Feature Extraction and Learning.

Challenges:

Processing power and memory are two main limitations for this project. This limit the **fine tuning** of parameters and performance analysis of different approaches.

Overcome:

The system extracts all the SIFT features from all training images and writes into a CSV file. Here, we find first 500 feature of image ranked with respect to their importance. Each feature is represented by an array of 128 integers. This process needs to be done only once and part of feature extraction. This

CSV will be used to make clusters (Dictionary of features). This way systems handles all the out of memory errors and **runs smoothly even on low memory machines** (RAM 8GB and higher). Use of 'MiniBatchKMeans' function is part of this solution. This is a technique used to cluster the data by taking less time and compromising on accuracy. We found the **tradeoff** between the **performance and accuracy**.

MiniBatchKMeans

The algorithm takes small batches (randomly chosen) of the dataset for each iteration. It then assigns a cluster to each data point in the batch, depending on the previous locations of the cluster centroids. It then updates the locations of cluster centroids based on the new points from the batch. The update is a gradient descent update, which is significantly faster than a normal Batch K-Means update [1].

Flow of a program:

Below is the flow of program.

Part A: ExtractSIFTtoCSV.py

Input: Training images folder path

Output: ExtractSIFT.csv in same working directory (Caution- consumes space)

1. Extracts SIFT features from the images and stores into a CSV file.

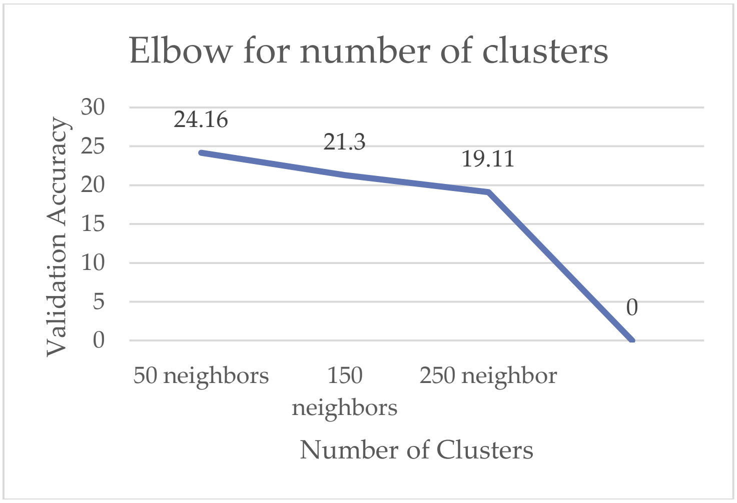
Part B: Kmeans_SVM_Histogram.py

Needs cv2 version 2.4.9 installed on Anaconda python 2.7

Input: ExtractSIFT.csv, Training images, Testing Images

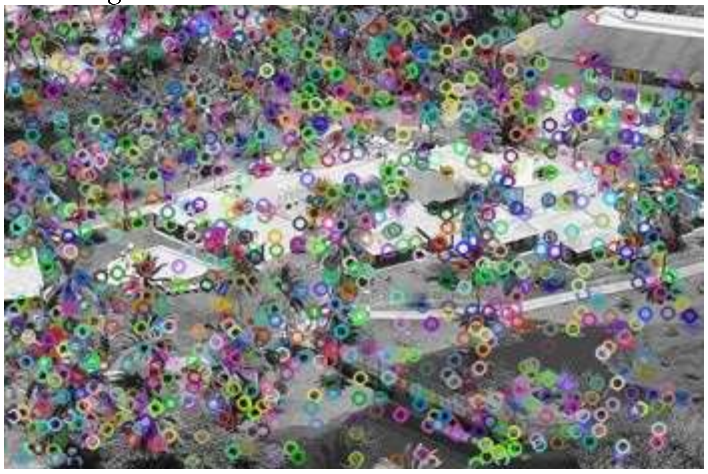
Output: ImagePredictions.csv – Predicted Class probabilities.

1. Read the CSV file created in part A and make clusters of it.
   Function: ms=MiniBatchKMeans (n_clusters=N_cluster,max_no_improvement=3,batch_size=20000)
   The number of clusters are 500, If no improvement after 3 rounds clustering stops, batch size 2000 denotes that 20000 samples will be taken every time for making 500 clusters.
   Use: ms.fit(SIFT_data)
   SIFT_data is the SIFT descriptors which we want to cluster.
   Reason: For number of clusters I have found elbow at 400-500. Below are some details of experiments I carried out to do so.



The above experiment carried out to find the suitable number of clusters. Validation set in above experiments is sample set of 2000 images. The primary purpose is to find out the elbow for number of clusters.

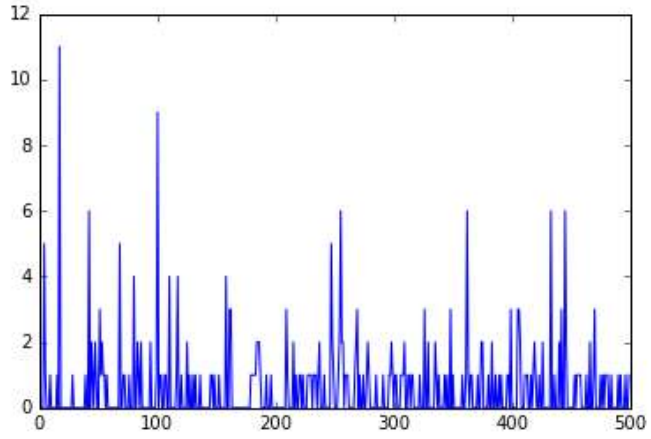
2. Read every training image and extract 300 SIFT feature from each image also extract gray histogram of 64 bins.
   Function: sft = cv2.SIFT(300) , kp,ds=sft.detectAndCompute(img,None),
   gray_hist = cv2.calcHist(img,[0],None,[64],[0,256])
   SIFT is a cv2 function and parameter given is the number of features you want to extract. Detect and compute will give you key-points and descriptors. Parameters to calcHist are image, 64 bins and range is 0-256.
   Use: Usage is shown above.


All features from SIFT

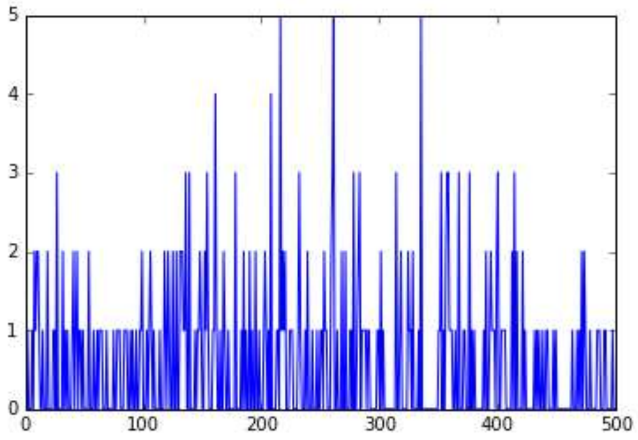
Top 100 feature of an Image.
Reason: These feature will be **learned** with a machine learning algorithm

3. Make a frequency count of these descriptor with respect to the clusters. Hence, every image will produce a bincount showing to which cluster its feature belong to.
   Function: np.bincount (cluster_predicted,minlength=N_cluster)

   Results:



Sample bincount for an image. Number of clusters are 500

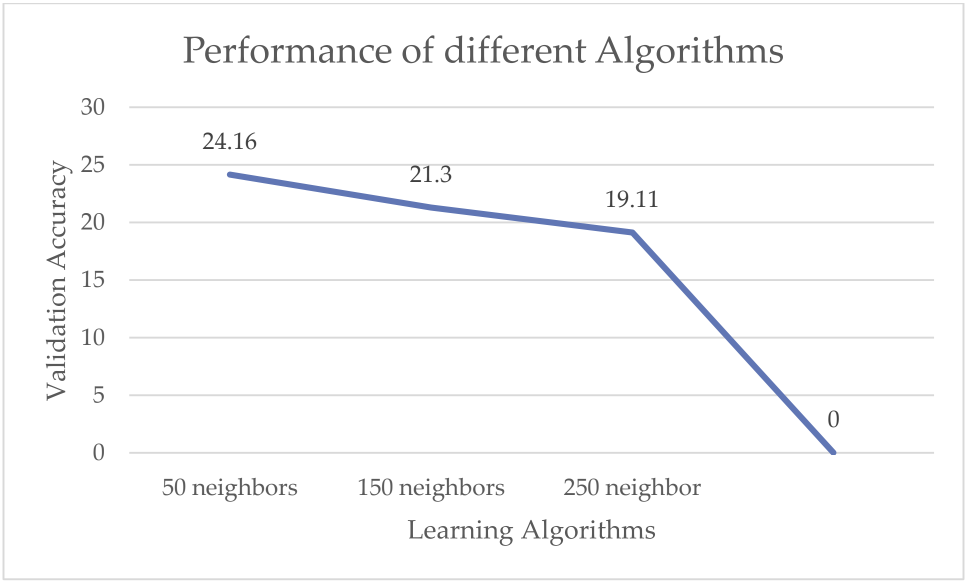


Sample bincount for an image. Number of clusters are 500

4. List of feature vectors is made which consist of a above bincounts and histogram (64 bin histogram)
   Function: Python lists are used to append the feature vectors.
5. SIFT features are learn with the help of SVM using rbf kernel. This step **generates** the model
   Function: clf = SVC(kernel='rbf', cache_size=4096,probability=True), clf_SIFT.fit(train_f_vectors,Y)
   Fit will make a model for all training feature vectors.
   Reason: SVM is a strong learner and different learning are applied and depending on **the validation accuracy.**
6. Do the same step 2,3, and 4 for testing data
7. **Predict** the data
   Function: Y_predicted_SIFT = clf_SIFT.predict_proba(test_f_vectors),
   Y_predicted_HIST = clf_HIST.predict_proba(test_f_vectors_hist). This will give probabilistic prediction for test data.

Results:

This project is a machine learning project and there are several learning methods. Below are some learning method we tried depending on the data we have.



These results have **parameters** associated with it. For example, these results are based on the cluster of 500 and number of descriptor used to make 500 clusters are 500000, these results are for SIFT feature learning. Furthermore, **BaggingClassifier** uses K-nearest neighbor with k=250 and 100 such classifiers and bootstrapping. You can see that accuracy is increased in bagging classifier. The **validation** set is a sample set for testing purpose.

We select SVM for learning SIFT features and KNN to learn histogram features. Below are some tuning experiment for SVM.

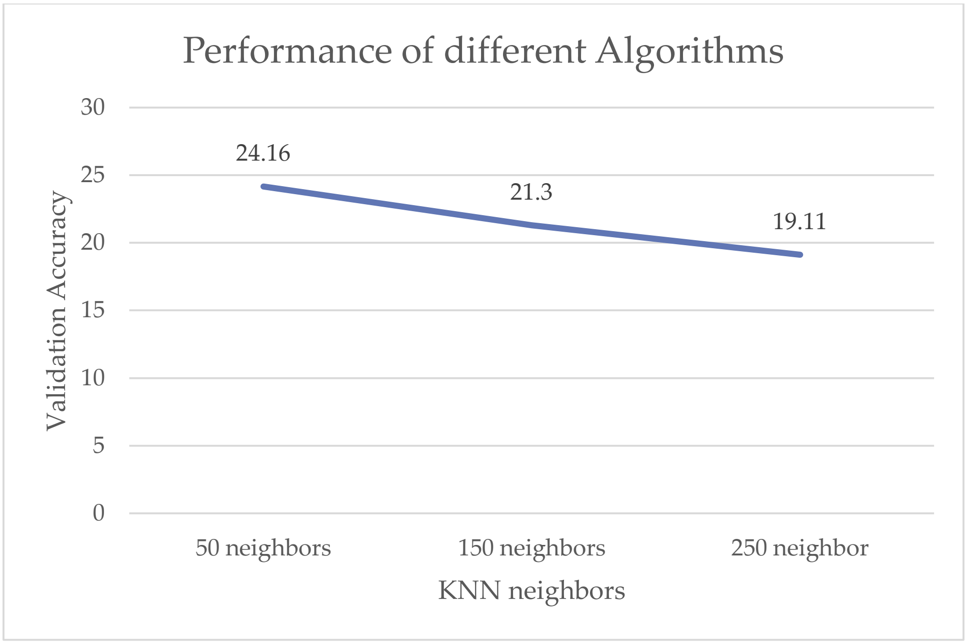
1. SVM – Very large computation time
   Descriptor = 500000, Clusters = 500
   Test SIFT = 200
   SVM = kernal 'poly' cache 2GB, degree 3
   Validation- 49%

2. SVM – Less time more accuray
   Descriptor = 500000, clusters = 500
   Test SIFT = 200
   SVM = kernal 'rbf' cache 2GB
   Validation- 52%

3. SVM – Increased number of clusters
   Descriptor = 500000, clusters = **1000**, batch size = 1000
   Test SIFT = 200
   SVM = kernal 'rbf' cache 2GB
   Validation- 52.65%

4. SVM – Increased number of Descriptor
   Descriptor = **800000**, clusters = 500, batch size = 1000
   Test SIFT = **300**
   SVM = kernal 'rbf' cache 2GB
   Validation- 55.46%

These all parameters are tuned based on the algorithm and its significance. When we increase the number of SIFT feature to be extracted the accuracy increases.
Problem: Sometimes SIFT gives hard classification, which indeed affect the logarithmic error function. If SVM confidently predicts a wrong class then, error function gives significant error.
Solution: We take a help of histogram feature to smoothen the SIFT prediction. This is done by taking vote of KNN who learned the histogram. Validation accuracy for KNN is as below:



Why this is so?

The choice of K equal to the square root of the number of instances is an empirical rule-of-thumb popularized by the "Pattern Classification" book by Duda et al.,

How does it work?

Consider an example where SVM predicted a **wrong** class with probability of 0.9. But, histogram – Knn predicted 0.2 for the same class. Now, we take 10% vote (because of low accuracy) of histogram-Knn and 90% vote of SIFT-SVM.

Hence, new probability = 0.9*0.9 + 0.2*0.1 = 0.81+0.01 = **0.82 which is less than 0.9**

Running the code:

The code is kept simple with lot of manual settings. For validation you need to make a separate directory of the images from training images. Give this as a testing images directory and run the code. You need to uncomment the lines

```
#Y_actual = [dict_Y[int(x)] for x in test_img_ids]
#accuracy = clf.score(test_f_vectors,Y_actual,sample_weight=[1 for i in range(len(test_f_vectors))])
#print accuracy*100
```

You may skip running the Part A of the project by downloading the CSV output at https://drive.google.com/a/uncc.edu/file/d/0BxQ3gjBIyZGxT2xmbjNpaUFISE0/view?usp=sharing (UNCC login needed)

References:

[1] : Working of MiniBatchKmeans – Stack overflow and Scikit learn.

[2]: Machine learning blog in internet