Machine Learning for Computer Vision Coursework 2

Hao Ding (CID: 00734091), Tanat Sanpaveeravong (CID: 00733970),
Department of Electrical and Electronic Engineering, Imperial College London, SW7 2AZ
Email: hd1812@ic.ac.uk, ts3912@ic.ac.uk

PART 1: IMAGE QUANTISATION BY K-MEANS

The emphasis of this part is on the K-means algorithm. Three quantised images are constructed by applying K-means clustering to the pixel data of a RGB cat image. This is done by initialising K cluster centres (centroids) and assigning a cluster to each data point by making sure that the Euclidean distance from a data point to its corresponding cluster is minimised. After this, the centroids will be recalculated and the data points will be reassigned. The whole process will repeat until the centroids no longer move. To be more specific, this process aims to minimise the objective function given by

$$J = \sum_{N}^{n=1} \sum_{K}^{i=1} r_{ni} ||d_n - \mu_i||^2, \tag{1}$$

where d_n denotes a data point and μ_i denotes a center of cluster, has to be minimised.

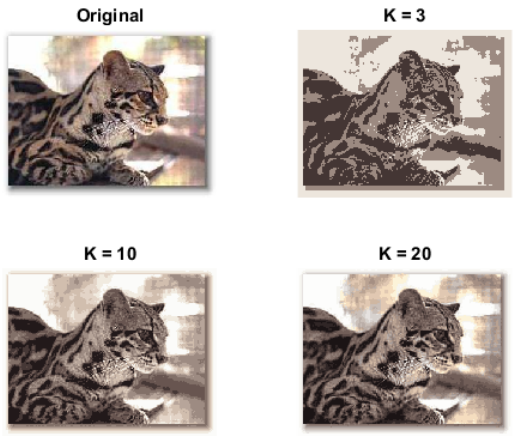

Fig. 1. The original cat image and the three quantised images for different values of K.

From Figure 1, when the value of K is set to 3, the pixel data are sorted into 3 clusters. As a consequence, the quantised cat image is represented by only 3 colours. This new image is basically the original image with much less colour details. Increasing the value of K, in this case to 10 and 20, allows the quantised image to be represented by more colours and thus, making it a more accurate representation of the original image.

By keeping K value fixed at 3, it can be observed from the right plots of Figure 2 that, under the default initialisation algorithm in Matlab, approximately 10 iterations are needed for the objective function to converge and hence, reaches its local minima. Once the convergence is reached, there will be no significant improvements to the result and thus, more iterations will only incur higher computational complexity.

It should be noted that the default initialisation algorithm in Matlab is the K-means++ algorithm, which uses an heuristic to find centroids seeds before the iterations start. The intuition behind K-means++ is to spread the initial cluster centroids. The first cluster center is chosen randomly while other centroids are chosen sequentially with the probability that is proportional to the squared distance from the data point to the closest cluster center. That is to say, the further the data point is from the existing cluster center, the higher the probability that it will be initialised as the next cluster center. Several generalisations are shown in the left plots of Figure 2. It can be observed that the initial cluster centers are more evenly spread compared to when random initialisation is used.

Additionally, the cluster centroids tend to move in every iteration. The step-sizes of their movements decrease as the number of iterations increases. This relationship will hold until a convergence is reached and is illustrated by the decreasing exponential shape of the objective function shown in all the right plots of Figure 2 and 3.

For a fixed value of K, several randomly initialised generalisations are shown in Figure 3. It can be observed that the random initialisation process gives higher cost in objective function and converges with larger number of iterations compared to when the K-means++ algorithm is used. Thus, from all the findings discussed earlier, the seeding method used by the K-means++ algorithm is considered to be more desirable.

PART 2: MULTI-CLASS SVM FOR TOY SPIRAL DATA

Support Vector Machines (SVM) for binary classification can be extended to tackle a multi-class classification problem by considering the one-versus-one and one-versus-rest algorithm.

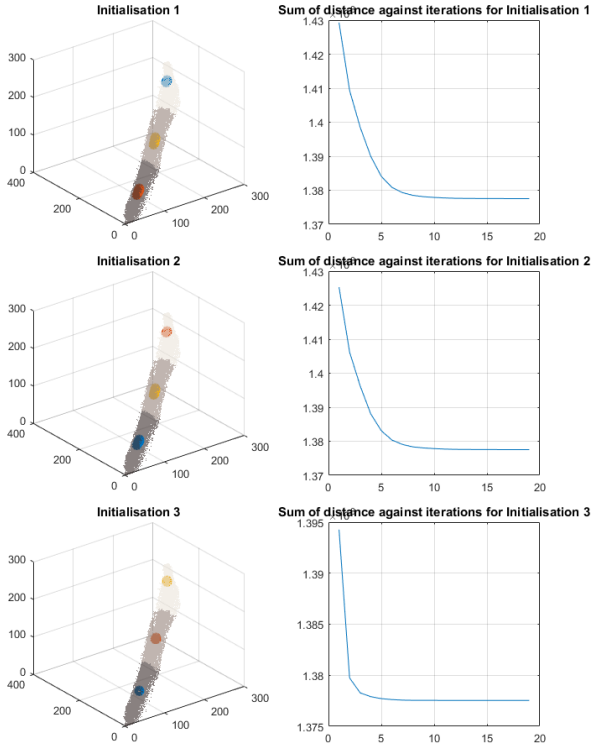

Fig. 2. Realisations with the default kmeans++ algorithms.

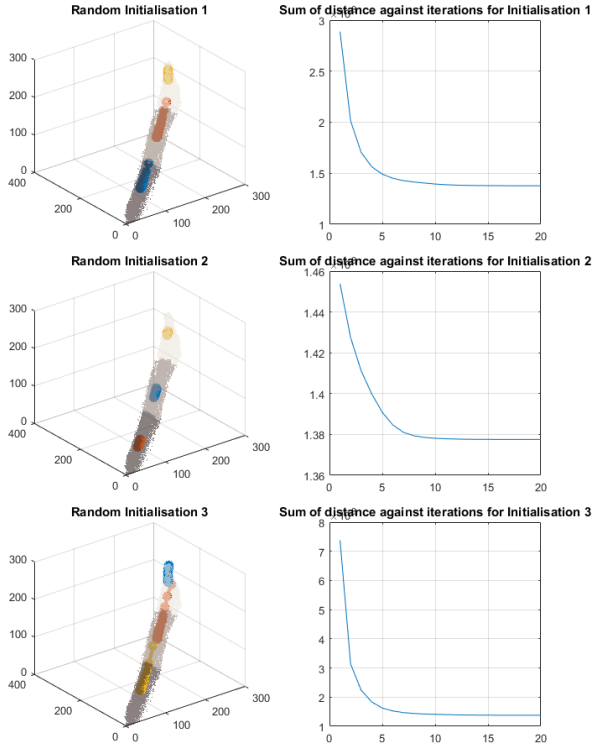

Fig. 3. Realisations with Random Initialisation.

The one-against-one (OAO) algorithm, also known as one-versus-one algorithm, constructs and trains a classifier for each pair of classes. Assuming that there are M classes, the number of classifiers will be $\frac{M\times(M-1)}{2}$. Each classifier votes on assigning a class to each testing sample and the class with the highest vote will be assigned. If the same number of votes are obtained for more than one class, the decision will become ambiguous. This problem can be solved by comparing the sum of the confidence levels (output probability of SVM). This algorithm gives a good performance in the case where a kernel function, which will be discussed later on, does not scale well. This is because each classifier is only trained with a small set of training data. However, the one-vesus-one algorithm can be comparatively slow due to its high complexity of $O(N^2)$.

The one-against-all (OAA) algorithm, also known as one-versus-rest algorithm, trains one classifier for each class. It labels the data in the class as positive one (+1) and the rest as negative one (-1). Classifiers trained from all classes are used to predict the class for the testing data. The label of the classifier with the highest SVM output value (probability) will be assigned to the testing data. This algorithm is generally faster than OAO. Its complexity its is $O(N)$ where N is the number of classes.

In order to classify a non-linear data-set, Kernel functions can be used to map this data-set into a higher dimension. Thus, its shape will be transformed in such a way that it can now be linearly classified (using a separating hyperplane). Several types of Kernel function are available in the LIBSVM library, these include: linear, polynomial and radial basis function (Gaussian) kernel. Their performance will be investigated later on in this part.

With the introduction of kernels, it is important to mention the error weighting constant C before each type of kernel can be discussed. Its function is to penalise misclassification of samples in proportion to their distance from the classification boundary. If C takes on a large value, the optimization will choose a smaller-margin hyperplane. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more data points. It should be noted that the default C-SVC formulation is used throughout the whole SVM implementation.

Linear Kernel:

Let us begin by looking at the linear kernel, which is defined as

$$k(x, x') = x^T x \tag{2}$$

From Figure 4, it can be observed that the linear kernel shows poor classification performance when used on the given spiral data. Unlike other kernels, its performance cannot be further improved by varying the error weighting constant C since it is prone to over-fitting.

The linear SVM has the complexity of $\mathcal{O}(K)$ where K is the input dimension. Due to its simplicity, its computational speed

is much faster than that of other kernels, which are non-linear. Therefore, it is often used in the cases where the number of features is large and the features are linearly separable. Clearly, spiral data classification is not one of those cases.

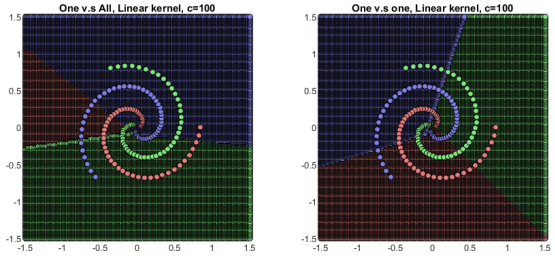

Fig. 4. One vs all and one vs one svm using linear kernel.

Polynomial Kernel:

The polynomial kernel is defined as

$$k(x, x') = (\gamma x^T x' + coef0)^M \tag{3}$$

In LIBSVM, the parameters associated with the polynomial kernel include: γ, $coef0$ and $degree$. The $degree$ parameter determines the order M of the polynomial kernel. Higher degree of polynomial means that the data is projected into a higher dimension. This allows for a more flexible decision boundary as shown in Figure 5. However, if the order is too high, it is very likely that overfitting will occur. The parameter γ and $coef0$ also influence the shape of the separating hyperplane by scaling the term $x^T x'$ and introducing a bias respectively.

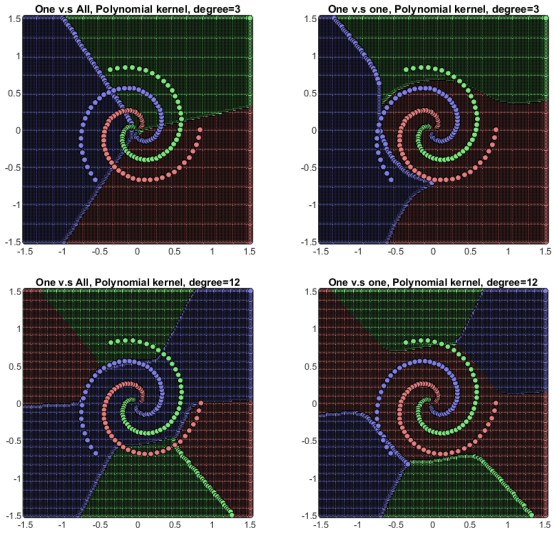

Fig. 5. One vs all and one vs one svm using polynomial kernel with various degrees.

It can be seen from the Figure 6 that the polynomial kernel performs a reasonably good classification on the spiral data after $degree$, $coef0$ and γ are carefully selected. By

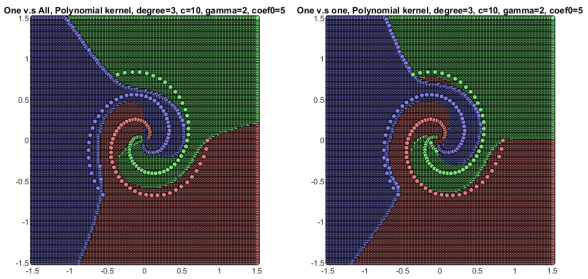

Fig. 6. One vs all and one vs one svm using polynomial kernel with careful parameter selection.

increasing the value of $coef0$ from its default value of 0 to 5, the decision boundary is altered in such a way that it can now handle non-linear data better. This gives the significant improvement to the classification performance.

It should be noted that the computational complexity of the polynomial kernel is $\mathcal{O}(K * N)$ where K is the vector dimension and N is the number of support vectors.

Radial Basis Function (RBF) Kernel:

The RBF kernel is one of the most popular types of kernel. Its definition is given by

$$k(x, x') = e^{-\gamma(||x^T - x'||)^2} \tag{4}$$

where γ equals to $\frac{1}{\sigma^2}$ and the term $(||x^T - x'||)^2$ is the squared Euclidean distance between the two feature vectors. This type of kernel can project the data into an infinite feature space since the Taylor exponential of the exponential term gives a power series.

By considering the definition of γ, it can be deduced that σ determines the area of influence that a support vector has over the data space. Since σ is inversely proportional to γ, a smaller value of γ will give the support vector more influence over a larger area and thus, a smoother (more generalised) decision surface. If γ is too small, the detailed features of the data will not be captured by the decision boundary. Larger γ will give the support vector more focus on the region close to it, thus the decision boundary is likely to capture the complex shape of data. However, a very large value of γ will cause the support vector to only focus on itself. This limit the ability of C to reduce overfitting.

From Figure 7, it can be observed that small γ tend to underfit the training data while larger gamma value will overfit the data. The results obtained from the experiment further confirm the explanation of γ given earlier.

The computational complexity of the RBF kernel is $\mathcal{O}(N^2)$. Despite its high complexity, the RBF kernel generally show better performance compared to other kernels.

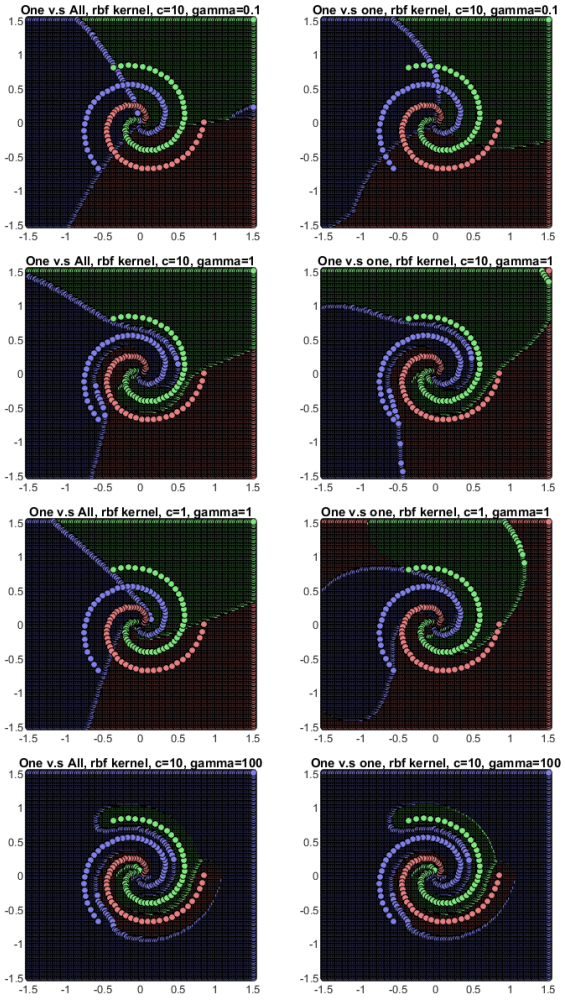

Fig. 7. One vs all and one vs one svm using RBF kernel with various degrees.

PART 3: BAG-OF-WORDS & MULTI-CLASS SVM FOR THE CALTECH101 DATASET

In order to perform image categorisation, the Bag-of-words model and multi-class SVM will be used. The training data set given has 10 classes with 15 randomly selected images per class. The same goes for the testing data set with 15 images per class.

100,000 descriptors are randomly selected for building the visual vocabulary. These descriptors are multi-scaled dense SIFT features. SIFT descriptors are Gaussian derivatives computed at 8 orientations over a 4×4 grid of spatial locations. Each descriptor has a dimension of 128. They can be considered as the low-level features that makes up an image. One example of a SIFT descriptor is shown in Figure 8.

The visual vocabulary is constructed through K-means clustering. Each cluster centroid, i.e. codeword, is a higher level description of the image feature and together they form a complete codebook. In the vector quantisation process, each image is described by the codewords. This process is done by finding the nearest clustering center for the SIFT

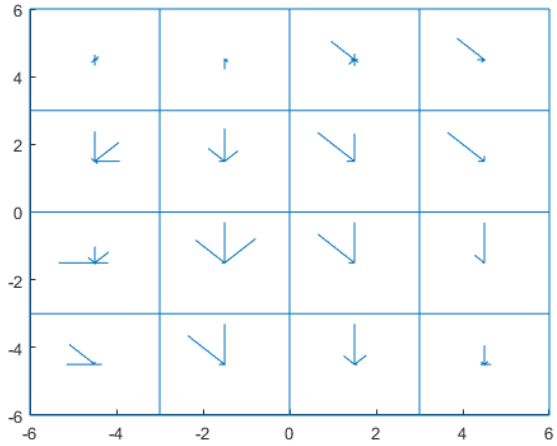

Fig. 8. An example of a SIFT descriptor.

descriptors in a image. The vocabulary size is defined as the number of cluster centres in a codebook. It should be large enough to distinguish the relevant features in different images but should not be too large as the model will try distinguish irrelevant variations such as noise, which is not desirable.

Figure 9 shows the images and bag-of-words histograms for two training and testing samples. The bins of each histogram are codewords and each bin counts the number of visual words that are assigned to each codeword. It can be observed from the first two histograms that both the training and testing image of an umbrella give a prominent peak at the same bin, which means that a unique feature corresponding to an umbrella is a codeword in that bin. The bottom two histograms show several prominent peaks corresponding to the unique features of the watch. It is important to note that, in this case, a good histogram should have only a few prominent peaks while the rest of the bins should have a low number of counts.

The multi-class SVM is trained and tested using the bag-of-words histogram vectors. In order to find the SVM specifications that deliver the best result, the SVM is trained using both the one-against-one (OAO) and one-against-all (OAA) algorithm. At the same time, different kernel types and parameter values will be investigated. The performance, in term of accuracy, for each variation will be compared to one another.

Let us first consider kernel types and parameter values. The RBF kernel gives the best accuracy performance out of all three kernels. The linear kernel is the second best while the polynomial kernel gives the worst performance. These findings are shown in Figure 10 where the accuracy level are plotted against the parameter C and γ. It should be noted that for the linear kernel, the only parameter associated with it is C, thus its accuracy is only plotted against C. By finding the most optimal combination of γ and C for the RBF kernel, the highest average level of accuracy achieved is 74.7% and 71.3% for OAA and OAO respectively. This

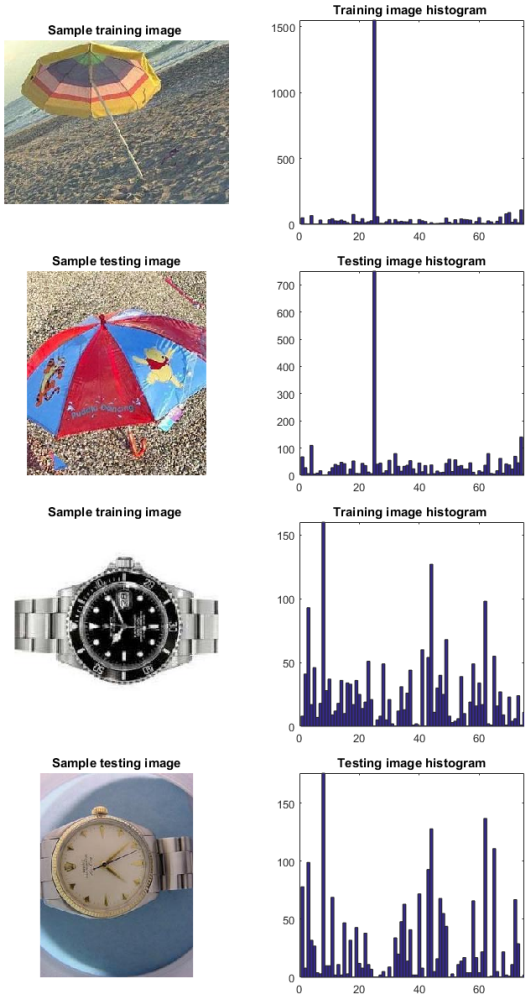

Fig. 9. Image and histogram samples for training and testing data.

value of γ is found to be 0.2 while the value of C used is 900. Although the RBF kernel seems to be the most desirable choice when looking at the accuracy level, its high complexity should also be considered when deciding which kernel to use. A compromise between accuracy and speed should be made for the case the case where the classification has to be done in real-time. Thus linear kernel could be a better option since the level of accuracy that it can deliver is only slightly lower than that of the RBF kernel.

Based on the description of γ given in the part 2, a small value of γ will yield a more generalised classifier. This in turn gives a low the degree the overfitting. At the same time, the large value of C used (900) further reduces overfitting. Therefore, by looking at the choice of parameters, it can be deduced that this data set is sparse and thus, requires a relatively linear classifier. This deduction can be further justified by looking at the data set itself, which contains only 15 training and testing images per class and considering the fact that the linear kernel gives the accuracy level close to that of the RBF kernel.

Now, the performance OAO and OAA algorithm, which are described earlier in part 2, will be compared. The OAA is expected to be faster than OAO since OAO has the computational complexity of $\mathcal{O}(N^2)$ while the OAA has the complexity of $\mathcal{O}(N)$. In term of the accuracy, OAA shows a better performance than OAO. This conclusion is drawn by looking at Figure 10 where OAA gives a more accurate prediction for all types of kernel compared to when OAO is used. This is expected as, in this case, overfitting is not desirable. By considering the way that the OAA algorithm works, OAA tends to give a more generalised classification compared to the OAO. This implies less overfitting and as such, OAA is better.

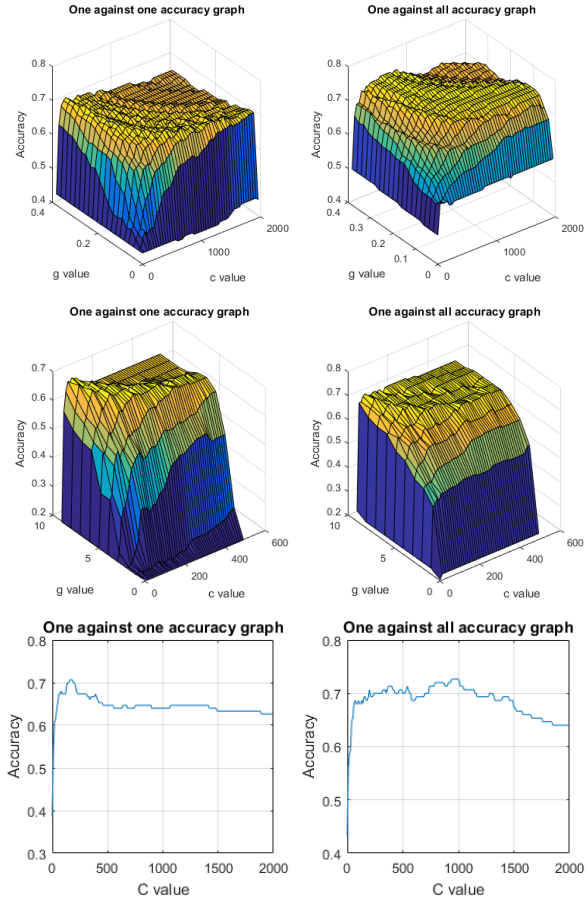

Fig. 10. Accuracy plot for RBF kernel (Top), polynomial kernel with $M = 2$ (Middle) and Linear Kernel (Bottom).

An alternative way to check for accuracy is through the mean rank. It is computed by averaging over the rank of the predicted class label after classification. For example, if a testing image belongs to class 1, and in the result matrix class 1 receives the 2nd highest votes (the 2nd highest probability), the rank for that testing image is 2. The same process is repeated ob all testing images and the averaged value can be used to estimate the prediction accuracy of the algorithm. In the case that gives the highest prediction accuracy, which is when the RBF kernel is used with γ = 0.2 and γ = 900, the mean rank for the OAO and OAA is approximately 1.81 and 1.69 respectively. This agrees with the prediction accuracy shown in the plot. For a linear

kernel, the OAO gives the mean rank of 1.78 while the OAA gives the mean rank of 1.61. For the polynomial kernel, the mean rank for OAO and OAA is 2.41 and 1.85 respectively.

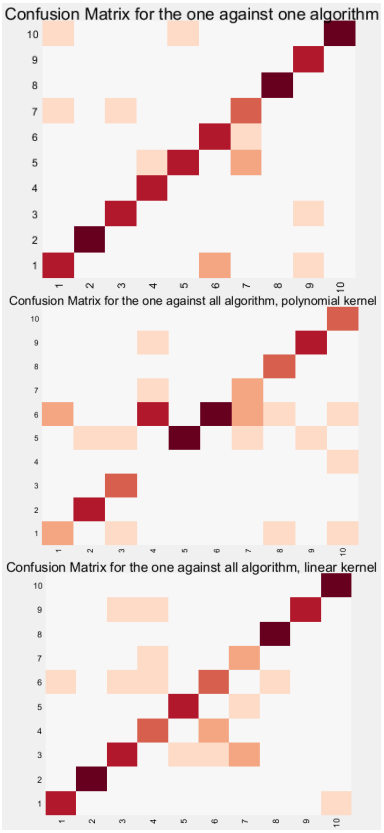

Fig. 11. Confusion matrix for RBF kernel (Top), 2nd-order polynomial kernel (middle) and linear kernel (Bottom).

Figure 11 shows the confusion matrices for different types of kernel. It can be observed that most data is diagonally clustered, which indicates a suitable level of prediction accuracy.

Regarding the time efficiency, many factors mentioned in the previous section has an impact on the computation time. For example, the OAO algorithm has a complexity of $\mathcal{O}(N * N)$, which is higher than that of OAA, and thus will consume more computation time when used. For the kernel types and parameter values, linear kernel gives the fastest computation time and thus, it should be used when the data is linearly separable. Non-linear kernels such as the RBF kernel and polynomial kernel are more computationally expensive. It should be noted that the time efficiency of the polynomial kernel is highly dependent on its order. The vocabulary size is another factor that should be considered. It determines the input dimension for the training and testing of the SVM model. Therefore, L=larger vocabulary size will obviously to lead to grater computation time.

Figure 12 shows the performance graph for the RBF kernel with different vocabulary size. It can be observed that if the vocabulary size is too small or too large, prediction accuracy decreases. The optimal vocabulary size is found to be at around 75. The codebook with small vocabulary size has limited ability to distinguish the difference between image classes. However, if the vocabulary size is too large, the codebook tend to capture the noise between image classes, that is the insignificant differences between images. Therefore, testing accuracy drops if the vocabulary size exceeds the optimal value.

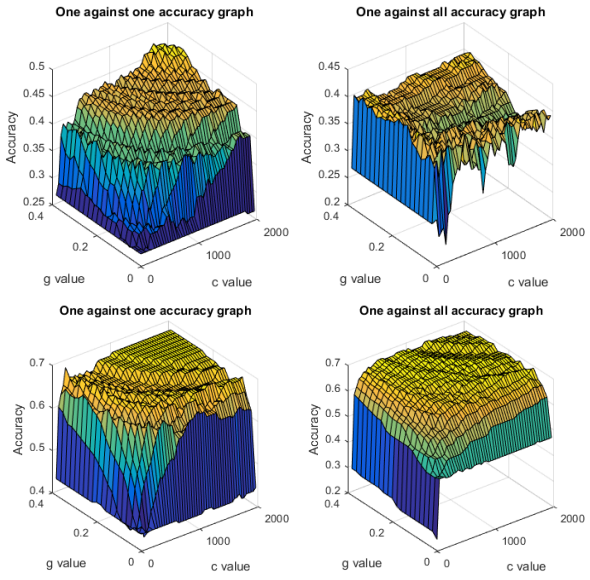

Fig. 12. Performance graph for RBF kernel, vocabulary size = 10 (Top), 80 (Bottom).

APPENDIX

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Code snippet for getData.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% K-means clustering
% write your own codes here
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Find the cluster centroids for selected SIFT descriptors
kclass=75;
[idx,C,sumd,D]=kmeans(desc_sel',kclass,'Display','iter','MaxIter',10000);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('Encoding Images...')
% Vector Quantisation
% write your own codes here
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Variable Initialisation
SVMTrainingData=[];

for imagec=1:10
    for imagei=1:15

        %Find the SIFT decriptors for the specified image class
        %and index
        imageDes=desc_tr{imagec,imagei};

        %Find the nearest neighbour to of the query image
        [IDD,D]=knnsearch(C,single(imageDes'));

        %Using BoW to represent the image
        [Counts,Centers]=hist(IDD,[1:kclass]);

        %Normalise the histogram and update training data array
        data_entry=[Counts/sum(Counts)*1,imagec];
        SVMTrainingData=[SVMTrainingData;data_entry];

        %Visualise training and testing images for selected class
        %and index
        if imagei<2 && (imagec==3 || imagec==4 )
            figure('Position', [100, 100, 800, 300]);
            subplot(1,2,1);
            subFolderName = fullfile(folderName,classList{imagec});
            imgList = dir(fullfile(subFolderName,'*.jpg'));
            I = imread(fullfile(subFolderName,imgList(imgIdx_tr(imagei)).
                name));
            imshow(I);
            title('Sample training image ');
            subplot(1,2,2);
            hist(IDD,[1:kclass]);
            axis([0,kclass,0,inf]);
            title('Training image histogram ');
        end

    end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Vector Quantisation
% write your own codes here
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Variable initialisation
%Code in
SVMTestingData=[];
for imagec=1:10
    for imagei=1:15

        %Select descriptors for testing images
        imageDes=desc_te{imagec,imagei};

        %Use knn search to find the nearest centroid
        [IDD,D]=knnsearch(C,single(imageDes'));

        %Use BoW to represent testing image
        [Counts,Centers]=hist(IDD,[1:kclass]);

        %Normalise testing image and update result array
        data_entry=[Counts/sum(Counts)*1,imagec];
        SVMTestingData=[SVMTestingData;data_entry];


        %Visualise images for selected training and testing
        %images
        if imagei<2 && (imagec==3 || imagec==4 )
            figure('Position', [100, 100, 800, 300]);
            subplot(1,2,1);
            subFolderName = fullfile(folderName,classList{imagec});
            imgList = dir(fullfile(subFolderName,'*.jpg'));
            I = imread(fullfile(subFolderName,imgList(imgIdx_te(imagei)).
                name));
            imshow(I);
            title('Sample testing image ');
            subplot(1,2,2);
            hist(IDD,[1:kclass]);
            axis([0,kclass,0,inf]);
            title('Testing image histogram ');
        end
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Code snippet for mainGuidelines.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%This code shows an exhausting search over parameters for svm training
%Both one v one and one v all are included

%Variable Initialisation
Accuracy1VAMat=[];
Accuracy1V1Mat=[];
Accuracy1VATrainMat=[];
Accuracy1V1TrainMat=[];
crange=0.0001:40:2000;
grange=0.001:0.02:0.4;
```

```
%Looping through various parameters
for cvalue = crange

    %Param initialisation
    Accuracy1VA=[];
    Accuracy1V1=[];
    Accuracy1VATrain=[];
    Accuracy1V1Train=[];

    for gvalue=grange

    %String to set svm params
    tValueStr='-t 2';
    cValueStr=[' -c ',mat2str(cvalue)];
    gValueStr=[' -g ',mat2str(gvalue)];
    param=strcat(tValueStr,cValueStr,gValueStr);

    %Param initialisation
    score=[];
    class=10;
    LabelIdx=size(data_train,2);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %1vA algotithm on testing data
    for i=1:class

        %Label one class as +1 and the rest as -1
        LabelMat=2*double(data_train(:,LabelIdx)==i)-1;

        %Train svm
        SVMModel = svmtrain(LabelMat,data_train(:,[1:LabelIdx-1]),param);

        %Test model
        [predict_label, accuracy, prob_estimates]=svmpredict(data_test(:,
            LabelIdx),data_test(:,[1:LabelIdx-1]),SVMModel);

        %Store prediction probability
        score=[score,prob_estimates];
    end

    %Find the max probability and save it
    [~,result]=max(score,[],2);

    %Find the testing accuracy
    Accuracy1VA=[Accuracy1VA;sum(double(result==data_test(:,LabelIdx)))/
        length(result)];

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %1v1 algorithm

    %Param initialisation
    voteMat=[];

    %Looping through class
    for i=1:class-1
        for j=i+1:class

            %Select two classes of data
```

```
        data_train_1v1=data_train(data_train(:,LabelIdx)==i | data_train
            (:,LabelIdx)==j,:);
        %Label one class as 1 and the other class as -1
        LabelMat1v1=2*double(data_train_1v1(:,LabelIdx)==i)-1;

        %Model training
        SVMModel1v1 = svmtrain(LabelMat1v1,data_train_1v1(:,[1:LabelIdx
            -1]),param);

        %Model testing
        [predict_label_1v1, accuracy1v1, prob_estimates_1v1]=svmpredict(
            data_test(:,LabelIdx),data_test(:,[1:LabelIdx-1]),SVMModel1v1)
            ;

        %Find all vote and save to a matrix
        vote=(predict_label_1v1+1)/2*i+(-predict_label_1v1+1)/2*j;
        voteMat=[voteMat,vote];
    end
end

%Find the class with the highest vote
result1v1=mode(voteMat,2);

%Calculate accuracy
Accuracy1V1=[Accuracy1V1;sum(double(result1v1==data_test(:,LabelIdx)))/
    length(result)];

%find confusion matrix
cfm1vA=confusionmat(result,data_test(:,LabelIdx));
cfm1v1=confusionmat(result1v1,data_test(:,LabelIdx));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Same process has been repeated on the training data

%Trainning accuracy
score=[];
class=10;
for i=1:class
    LabelMat=2*double(data_train(:,LabelIdx)==i)-1;
    SVMModel = svmtrain(LabelMat,data_train(:,[1:LabelIdx-1]),param);
    [predict_label, accuracy, prob_estimates]=svmpredict(data_train(:,
        LabelIdx),data_train(:,[1:LabelIdx-1]),SVMModel);
    score=[score,prob_estimates];
end
[~,result]=max(score,[],2);
Accuracy1VATrain=[Accuracy1VATrain;sum(double(result==data_test(:,
    LabelIdx)))/length(result)];

voteMat=[];
for i=1:class-1
    for j=i+1:class
        data_train_1v1=data_train(data_train(:,LabelIdx)==i | data_train
            (:,LabelIdx)==j,:);
        LabelMat1v1=2*double(data_train_1v1(:,LabelIdx)==i)-1;
        SVMModel1v1 = svmtrain(LabelMat1v1,data_train_1v1(:,[1:LabelIdx
            -1]),param);
        [predict_label_1v1, accuracy1v1, prob_estimates_1v1]=svmpredict(
            data_train(:,LabelIdx),data_train(:,[1:LabelIdx-1]),
```

```
                SVMModel1v1);
            vote=(predict_label_1v1+1)/2*i+(-predict_label_1v1+1)/2*j;
            voteMat=[voteMat,vote];
        end
    end

    result1v1=mode(voteMat,2);
    Accuracy1V1Train=[Accuracy1VATrain;sum(double(result1v1==data_test(:,
        LabelIdx)))/length(result)];
    end
    Accuracy1VAMat=[Accuracy1VAMat,Accuracy1VA];
    Accuracy1V1Mat=[Accuracy1V1Mat,Accuracy1V1];
    Accuracy1VATrainMat=[Accuracy1VATrainMat,Accuracy1VATrain];
    Accuracy1V1TrainMat=[Accuracy1V1TrainMat,Accuracy1V1Train];

end
```