Simple Image Classification using Principal Component Analysis (PCA)

Alex Norko, Electrical and Computer Engineering Department
GMU Volgenau School of Engineering, Fairfax, VA, USA
December 9, 2015

***Abstract*—This project researches, analyzes, develops, runs, and evaluates simple image classification algorithms both using and not using principal component analysis (PCA). The approach compares k - nearest neighbor (kNN) and linear support vector machine (SVM) classification algorithms to classify 10000 32x32 color images. Classification accuracy using training and test sets and confusion matrixes are compared.**

Keywords— image classification; principal component analysis; k - nearest neighbor; support vector machine

I. Introduction

This project researches, analyzes, develops, runs, and evaluates simple image classification algorithms using principal component analysis (PCA).

Images are high-dimensional objects which are difficult for machines to classify. Principal Component Analysis (PCA) creates reduced-dimension image features. This project compares the performance of some simple image classification algorithms both using and not using Principal Component Analysis (PCA).

The approach compares k - nearest neighbor (kNN) and linear support vector machine (SVM) classification algorithms to classify 10000 32x32 color images and looks at their performance both using and not using Principal Component Analysis (PCA). Classification accuracy using training and test sets and confusion matrixes are compared.

II. Related Work

The initial effort reviewed the Stanford Computer Science (CS) class, CS231n: Convolutional Neural Networks for Visual Recognition lecture on Image Classification and Nearest Neighbor Classifier [1].

Reference [1] introduced image classification and the Nearest Neighbor classifier on the CIFAR-10 dataset, described how to determine the optimal k hyper-parameter using a validation set or cross-validation, demonstrated how poor the test set accuracy using the kNN algorithm and L1 and L2 pixelwise distances.

It then looks at two papers using PCA on images, Principal Component Analysis in Image Processing [2] and Feature Based Image Classification by using Principal Component Analysis [3] and the early paper on the topic by Turk and Pentland, Eigenfaces for Recognition [4].

Reference [2] shows how PCA can be applied to reducing color image dimensions by three and identifying image rotations by finding the rotations of the image's principal components. Reference [3] applies PCA to effectively classify various types of clouds.

Reference [4] applies PCA to detecting and classifying faces. It applies the approach fairly successfully to properly positioned, scaled, and oriented faces. It serves as a good start to the broader image classification problem.

It finally looks at two papers by Rigamonti on sparse representations of images, Are Sparse Representations Really Relevant for Image Classification [5] and On the relevance of sparsity for image classification [6].

References [5] and [6] demonstrate that enforced sparsity is not an effective approach at improving classification unless the data contains redundancy. Experiments on CIFAR-10 confirmed the results and reference [6] also demonstrated the benefit of using whitening.

The CIFAR-10 dataset [7] and Learning Multiple Layers of Features from Tiny Images [8] provide a good background on the CIFAR-10 dataset written by Alex Krizhevsky, a machine learning researcher working with Geoffrey Hinton at the University of Toronto.

Reference [7] provides the basic description and layout, shows some sample image classes, and provides several references.

Reference [8] evaluates using Restricted Boltzmann Machine (RBM) and Deep Belief Network (DBN) models to classify images in the CIFAR-10 dataset. It describes the properties of natural images in the CIFAR-1- dataset, describes and demonstrates whitening, and RBMs and DBNs. It then describes, presents, and compares the results of these models in object classification experiments on the CIFAR-10 dataset.

III. Analysis

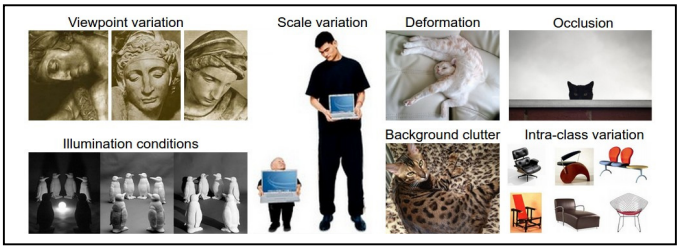
Automated image classification is a daunting problem. Simple pattern recognition alone is not enough to solve the problem well.

Some of the challenges with image classification sown in Fig. 1 have to do with:

1. "Viewpoint variation - A single instance of an object can be oriented in many ways with respect to the camera.

2. Scale variation - Visual classes often exhibit variation in their size (size in the real world, not only in terms of their extent in the image).
3. Deformation - Many objects of interest are not rigid bodies and can be deformed in extreme ways.
4. Occlusion - The objects of interest can be occluded. Sometimes only a small portion of an object (as little as few pixels) could be visible.
5. Illumination conditions - The effects of illumination are drastic on the pixel level.
6. Background clutter - The objects of interest may blend into their environment, making them hard to identify.
7. Intra-class variation - The classes of interest can often be relatively broad, such as chair. There are many different types of these objects, each with their own appearance [1]."

Fig. 1. Image classification challenges [1]



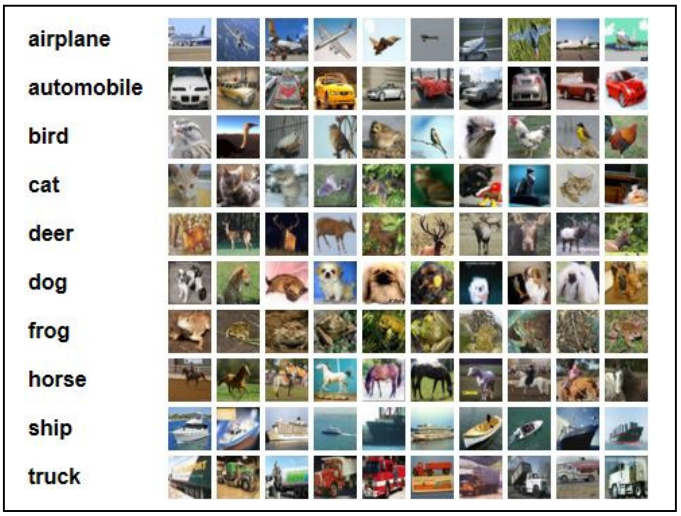
IV. Computational Work

The data, plan, and computations are described next.

A. *Data*

The CIFAR-10 dataset is used for the project. It comprises 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images [7], [8]. Fig. 2 shows several sample images of the CIFAR-10 image classes.

Fig. 2. CIFAR-10 image classes [7]



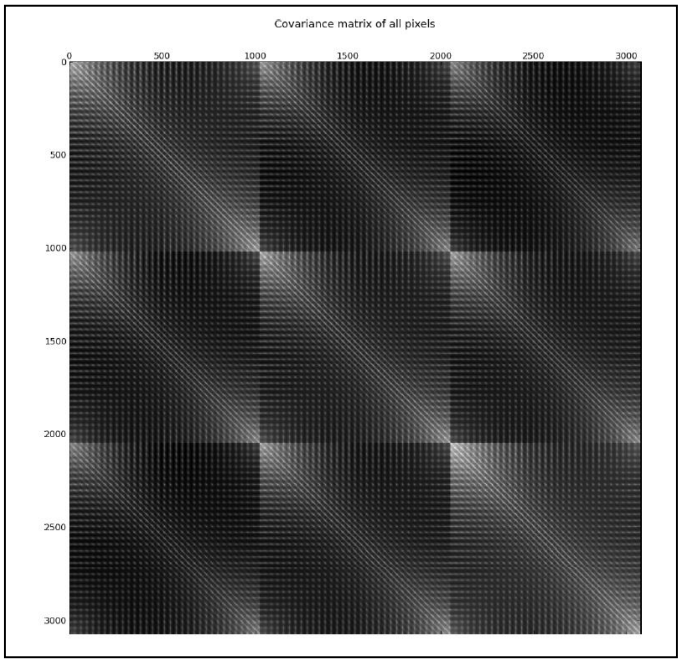
The Matlab file, cifar-10-matlab.tar.gz, was downloaded from [7]. It is a 179MB gzip'ed file. PeaZip was downloaded, installed, and run on Windows 8 to uncompress the gzip'ed data file. The following files were extracted:

1. readme.html – points to the CIFAR-10 page at [7]
2. data_batch_1.mat - data_batch_5.mat – 5 binary Matlab files containing the following workspace variables:
    a. batch_label – as the name says, batch label, e.g. 'training batch 1 of 5'
    b. data - image data stored as 10000x3072 unsigned, 8-bit integers
    c. labels – 10000x1 unsigned, 8-bit integers representing the training data classes
3. test_batch.mat – same as the data_batch_1.mat file but just 10,000 images to use for testing
4. batches.meta.mat – class labels stored as 10x1 cells, e.g. 'airplane'

"Each row of the data or test image array stores a 32x32 color image. The first 1024 entries contain the red channel values, the next 1024 the green, and the final 1024 the blue. The image is stored in row-major order, so that the first 32 entries of the array are the red channel values of the first row of the image." [7]

The covariance matrix in Fig. 3 reveals several interesting properties of the CIFAR-10 images.

Fig. 3. CIFAR-10 covariance matrix [8]



Pixels in the 32 x 32 (= 1024) bit images are indexed in row-major order (top-left moving right across rows and then moving down to the next row and repeating until done). The first 1024 rows or columns represent the red values, the next

1024 the green values, and the last 1024 the blue values. The 9 large squares represent the split in colors. Values range from 1 – 255 running from black to white. The brighter diagonal bands show that nearby pixels are the most correlated. [8]

B. Plan

k - Nearest Neighbor Classifier (kNN) and linear Support Vector Machines (SVM) algorithms are used for classification. Principal Component Analysis (PCA) is used for dimensionality reduction. The training data is cross-validated. Accuracy measures and confusion matrixes are used for evaluation.

1) Tools

All the work is done using Matlab. The Matlab Classification Learner (MCL) interactive application is used to create the classification models. The models are trained, cross-validated, and evaluated in the classification learner application. The model or code are then exported and applied to the test data.

2) Program Flow

The following flow is followed:

1. Dataset
    a. Import dataset
    b. Extract data
    c. Import data into Matlab
    d. Organize data into training and test sets
    e. Pre-process data as needed
2. Train classifiers
    a. Configure Matlab Classification Learner (MCL) for the desired classification algorithm, use or not of PCA, validation, and any other appropriate settings
    b. Import the test data into MCL
    c. Train the classifier
    d. Evaluate the trained classifier
3. Test classifiers
    a. Export the trained classifier
    b. Develop the test classification wrapper
    c. Run the test data through the trained classifier
    d. Evaluate the test results and compare to training results

3) Code

The following code elements were used:

1. Data set processing code
2. Trained classifier codes
3. Test classification wrapper
4. Evaluation code

4) Evaluation

The following evaluation measures were calculated and analyzed:

1. Classification accuracy –calculated for training and test data and with no validation and cross-validation of various size folds
2. Confusion matrixes
3. ROC (receiver operating characteristic) curves

C. Computations

Computations for kNN and SVM models with and without PCA are shown next.

1) k - Nearest Neighbor Classifier (kNN)

a) kNN on Images (without PCA)

PREPARE ENVIRONMENT

Set up Matlab to access the data. Add the data file folder and its subfolders to the Matlab path, C:\Users\anorko\Documents\0 - Okron\Schools\GMU\ECE699-ML\project\matlab

PREPARE DATA

Matlab Classification Learner (MCL) first needs its data in the Matlab workspace. Add variables to the Matlab workspace by double-clicking on a mat file in the current folder or by running a script. Mat file and script variables are added to the workspace.

Use the data from the first 10,000 image data files, data_batch_1.mat and put it into a single 10,000 image test data set.

PRE-PROCESS DATA

Center the data by subtracting the mean, $\boldsymbol{x}$, from each of the data points.

$$\boldsymbol{x} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n \quad (1)$$

$$\boldsymbol{z}_n = \boldsymbol{x}_n - \boldsymbol{x} \quad (2)$$

Matlab Classification Learner (MCL) also expects its data in a table format with data samples in rows and predictors and responses in the columns.

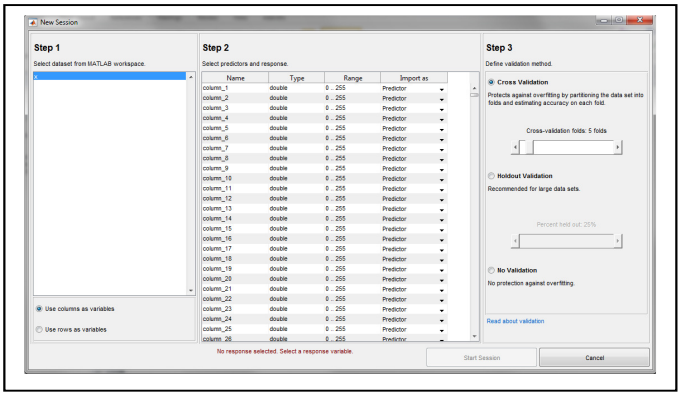
Add the mat file data and label variables to a new variable to use in MCL. Add the response column to the end of the 3,072 image predictor coordinates.

```
x = zeros(10000,3073);
x(1:10000,1:3072) = data(1:10000,1:3072);
x(1:10000,3073) = labels(1:10000,1);
```

Start MCL and click New Session, From Workspace to select data from the workspace and to specify a validation

scheme. Since the variable *x* was set up correctly as a table it shows up in Step 1 of MCL and its columns are listed in Step 2 defaulted as predictors in Fig. 4.

Fig. 4. Matlab Classification Learner data import



In Step 2, scroll down to the last row, named column_3073 and click the Import As drop-down and select it as a Response.

VALIDATE

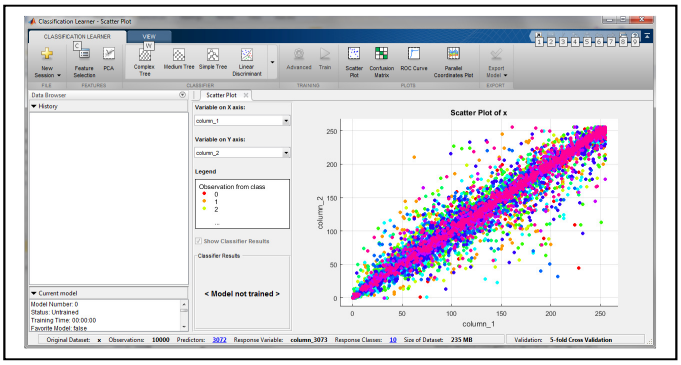
Continuing from the pre-processing section above, confirm that Step 3 is set for the Cross Validation with 5 folds default.

For k fold cross validation, MCL:

1. Partitions the data into k disjoint sets or folds
2. For each fold:
    a. Trains a model using the out-of-fold observations
    b. Assesses model performance using in-fold data
3. Calculates the average test error over all folds

Click the Start Session button and the MCL user interface opens as shown in Fig. 5.

Fig. 5. Matlab Classification Learner user interface



5-fold cross validation was run with Fine (k=1), Medium (k=10), and Coarse (k=100) kNN models to find the optimal k which had the lowest average training set accuracy.

From the Classifier drop-down, select the Fine kNN. Click the Advanced menu selection and uncheck Standardize Data. The Advanced setting are than set for:

1. Number of neighbors: 1
2. Distance metric: Euclidean (L2)
3. Distance weight: Equal
4. Standardize data unchecked

Standardized data is used to scale coordinate distances and would be useful if scales across features varied widely. It is less important in this case with common pixel scales.

Next run the Medium (k=10) and Coarse (k=100) kNN models using 5-fold cross validation. Table I tabulates the k-values and accuracy for each. The medium, k=10 model is optimum in this case with the highest accuracy = 28.9%.

TABLE I. OPTIMUM K FOR KNN MODELS

Optimum k for kNN models		
kNN Model	***k***	***Accuracy (%)***
Fine	1	28.2
Medium	**10**	**28.9**
Coarse	100	26.3

TRAIN

Once the best k is found, train the model with that k and the full training set with no validation. Validation is not necessary at this stage for the training set.

Apply the kNN classifier algorithm to the centered training dataset. Use the L2 distance.

Train the kNN model with the optimum k = 10, no data standardization, and no validation. Standardized data would scale the coordinate distances and would be useful if the scales across features varied widely. In this case with common pixel scales, it adds no value.

After training, export the model from MCL into a structure called trainedClassifier in the Matlab workspace to use for testing.

TEST

Apply the kNN model with the optimal k to the test dataset.

Cross validation showed k = 10 to be the optimum kNN model.

Rerun the exported training model on the original training dataset and confirm that the confusion matrix matches the one derived during training. (see the Evaluation section below to see the plots)

Test the model using the first 2,000 samples of the test_batch.mat test dataset.

EVALUATE

Calculate the training and test dataset accuracy and tabulate and plot the confusion matrix and the ROC (receiver operating characteristic) curves.

Accuracy measures the number of true results as a percentage of all data samples examined. The cross validation accuracy is the averaged in-fold accuracy calculated over the k out-of-fold training models.

Table II shows the accuracy of the 10NN model without PCA. As expected, training accuracy without validation is the best. The validation accuracy estimates a model's performance on new data compared to the training data and is close to the test accuracy.

TABLE II. 10NN MODEL ACCURACY (WITHOUT PCA)

10NN Model Accuracy (without PCA)		
Dataset type	***Validation***	***Accuracy (%)***
Training	None	39.7
Training	5 fold cross validation	28.9
Test	None	29.5

Overall 29.5 is not very good performance for a classifier. More data samples can improve the performance.

Fig. 6. CIFAR-10 kNN classifiaction [1]

Fig. 6 shows an example of the kNN classifier using k=7, L1 distance metric, 50K training samples and resulting in an accuracy of 38.6%. The left images are examples from the CIFAR-10 dataset. On the right, the first column shows a few test images and next to each the top 10 nearest neighbors in the training set according to pixel-wise difference. [1]

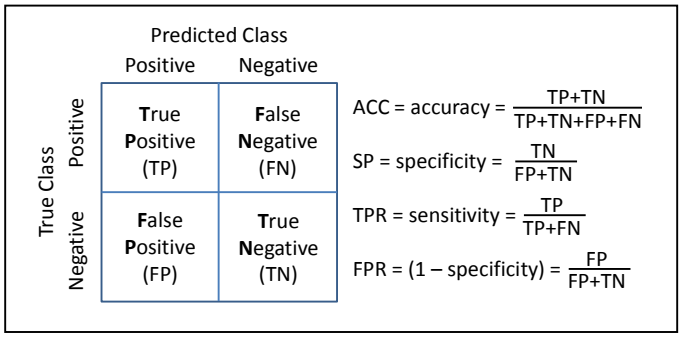
Though kNN is not the best model for image classification it does provide a good educational example of applying machine learning to simple images.

Confusion Matrixes

A confusion matrix is a table that describes the performance of a classification model (or "classifier") on a set of test data for which the true values are known.

Fig. 7 shows some of the key confusion matrix variables. The accuracy, true positive rate (TPR), and false positive rate (FPR) calculations are shown.

Fig. 7. Confusion matrix variables



In MCL confusion matrixes, the rows show the true class, and the columns show the predicted class. The diagonal cells show where the true class and predicted class match. If these cells are green and display high percentages, the classifier has performed well and classified observations of this true class correctly.

Fig. 8 shows the confusion matrix of the 10NN model run on the training dataset with no data standardization or validation as presented from the MCL interface.

Fig. 8. 10NN training dataset confusion matrix

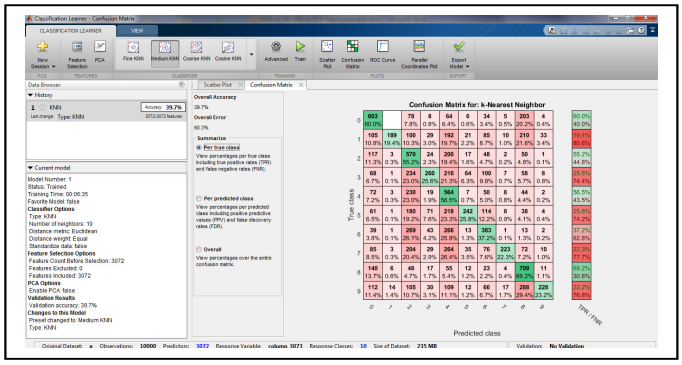
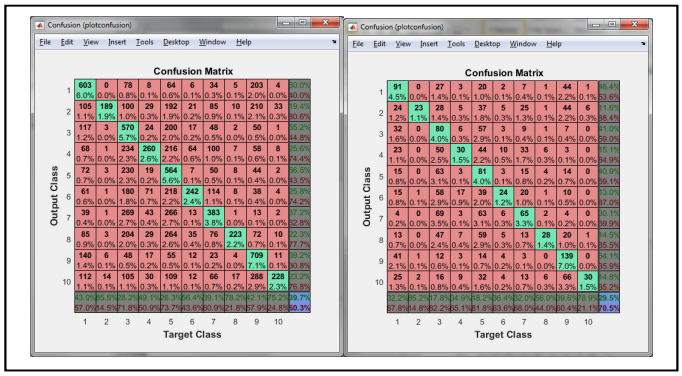


Fig. 9 shows the confusion matrixes of the 10NN model run on the training and test datasets with no data standardization or validation as presented using the Matlab plotconfusion function.

Fig. 9. 10NN training (left) and test (right) confusion matrix comparison



"On the confusion matrix plot, the rows show the predicted class, and the columns show the true class. The diagonal cells show where the true class and predicted class match. The off

diagonal cells show instances where the classifier has made mistakes. The column on the right hand side of the plot shows the accuracy for each predicted class (TPR and FNR), while the row at the bottom of the plot shows the accuracy for each true class (PPV and FDR). The cell in the bottom right of the plot shows the overall accuracy." [9]

The trained confusion matrix on the left shows higher overall accuracy than the test confusion matrix on the right with percentages in each cell calculated relative to the entire confusion matrix also generally higher in the trained confusion matrix but reasonably tracking them in the test confusion matrix.

Receiver Operating Characteristic (ROC)

The receiver operating characteristic (ROC) curve plots the true positive rate (TPR) versus the false positive rate (FPR). True positive rate (TPR) is the proportion of positive cases that were correctly identified. False positive rate (FPR) is the proportion of negatives cases that were incorrectly classified as positive [10].

Fig. 10. Receiver operating characteristic (ROC) curve for class 0

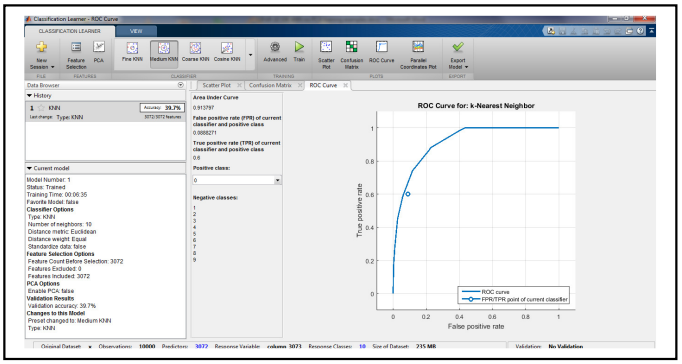
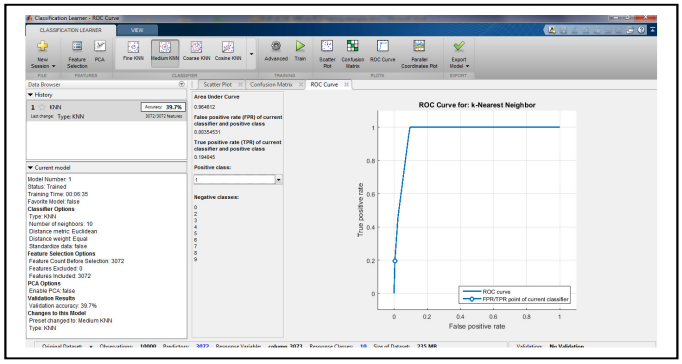


Fig. 10 shows the ROC curve for class 0. The marker on the plot shows the performance of the classifier model. The marker shows the values of the false positive rate (FPR) and the true positive rate (TPR) for the classifier model. For example, a false positive rate (FPR) of 0.1 indicates that the classifier model assigns 10% of the observations incorrectly to the positive class. A true positive rate of 0.6 indicates that the classifier model assigns 60% of the observations correctly to the positive class [10].

A perfect result with no misclassified points is a right angle to the top left of the plot. A poor result that is no better than random is a line at 45 degrees. The Area Under Curve (AUC) number is a measure of the overall quality of the classifier. Larger Area Under Curve values indicate better classifier performance [10].

Fig. 11 shows the ROC curve for class 1. With the AUC of 0.96 for class 1 vs 0.91 for class 0, the trained classifier model better classifies class 1 classes over class 0.

Fig. 11. Receiver operating characteristic (ROC) curve for class 1



A new curve can be plotted for each class. AUC's of the ROC curves for each class can be used to determine which class gets classified best with the trained classifier model.

b) kNN on Principal Components (using PCA)

PREPARE DATA

Use the data from the first 10,000 image data files, data_batch_1.mat and put it into a single 10,000 image test data set.

PRE-PROCESS DATA

Center the data by subtracting the mean, $\boldsymbol{x}$, from each of the data points.

$$\bar{x} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n \quad (1)$$

$$\boldsymbol{y}_n = \boldsymbol{x}_n - \bar{\boldsymbol{x}} \quad (2)$$

Scale the data so that it has unit standardization.

$$\boldsymbol{\sigma_i^2} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{y_{ni}^2} \quad (3)$$

$$\boldsymbol{z}_n = \boldsymbol{y}_n ./ \boldsymbol{\sigma_{ni}} \quad (4)$$

Matlab Classification Learner (MCL) also expects its data in a table format with data samples in rows and predictors and responses in the columns.

Add the mat file data and label variables to a new variable to use in MCL. Add the response column to the end of the 3,072 image predictor coordinates.

```
x = zeros(10000,3073);
x(1:10000,1:3072) = data(1:10000,1:3072);
x(1:10000,3073) = labels(1:10000,1);
```

Start MCL and click New Session, From Workspace to select data from the workspace and to specify a validation scheme. Since the variable *x* was set up correctly as a table it

shows up in Step 1 of MCL and its columns are listed in Step 2 defaulted as predictors.

In Step 2, scroll down to the last row, named column_3073 and click the Import As drop-down and select it as a Response.

VALIDATE

Continuing from the pre-processing section above, confirm that Step 3 is set for the Cross Validation with 5 folds default.

For k fold cross validation, MCL:

1. Partitions the data into k disjoint sets or folds
2. For each fold:
    a. Trains a model using the out-of-fold observations
    b. Assesses model performance using in-fold data
3. Calculates the average test error over all folds

Click the Start Session button and the MCL user interface opens.

5-fold cross validation was run with Fine (k=1), Medium (k=10), and Coarse (k=100) kNN models to find the optimal k which had the lowest average training set accuracy.

From the Classifier drop-down, select the Fine kNN. Click the Advanced menu selection and uncheck Standardize Data. The Advanced setting are than set for:

1. Number of neighbors: 1
2. Distance metric: Euclidean (L2)
3. Distance weight: Equal
4. Standardize data unchecked

Standardized data is used to scale coordinate distances. It was manually applied before performing PCA. This check box would apply it after PCA and that is not where it is required.

Next run the Medium (k=10) and Coarse (k=100) kNN models using 5-fold cross validation. Table III tabulates the k-values and accuracy for each. The medium, k=10 model is optimum in this case with the highest accuracy = 30.3%.

TABLE III. OPTIMUM K FOR KNN MODELS - PCA

Optimum k for kNN models - PCA		
kNN Model	***k***	***Accuracy (%)***
Fine	1	29.8
Medium	**10**	**30.3**
Coarse	100	27.7

TRAIN

Once the best k is found, train the model with that k and the full training set with no validation. Validation is not necessary at this stage for the training set.

Apply the kNN classifier algorithm to the centered training dataset. Use the L2 distance.

Train the kNN model with the optimum k = 10 and no validation. Standardized data prior to applying PCA.

After training, export the model from MCL into a structure called trainedkNNwithPCAClassifier in the Matlab workspace to use for testing.

TEST

Apply the kNN model with the optimal k to the test dataset.

Cross validation showed k = 10 to be the optimum kNN model.

Rerun the exported training model on the original training dataset and confirm that the confusion matrix matches the one derived during training. (see the Evaluation section below to see the plots)

Test the model using the first 2,000 samples of the test_batch.mat test dataset.

EVALUATE

Calculate the training and test dataset accuracy and tabulate and plot the confusion matrix and the ROC (receiver operating characteristic) curves.

Accuracy measures the number of true results as a percentage of all data samples examined. The cross validation accuracy is the averaged in-fold accuracy calculated over the k out-of-fold training models.

Table IV shows the accuracy of the 10NN model using PCA. As expected, training accuracy without validation is the best. The validation accuracy estimates a model's performance on new data compared to the training data and is close to the test accuracy.

TABLE IV. 10NN MODEL ACCURACY (USING PCA)

10NN Model Accuracy (using PCA)		
Dataset type	***Validation***	***Accuracy (%)***
Training	None	42.0
Training	5 fold cross validation	30.3
Test	None	30.6

Overall 30.6 is not very good performance for a classifier.

Though kNN is not the best model for image classification using PCA does improve its performance slightly.

Confusion Matrixes

Fig. 12 shows the confusion matrix of the 10NN model run on the training dataset with validation using PCA.

Fig. 12. 10NN training dataset confusion matrix using PCA

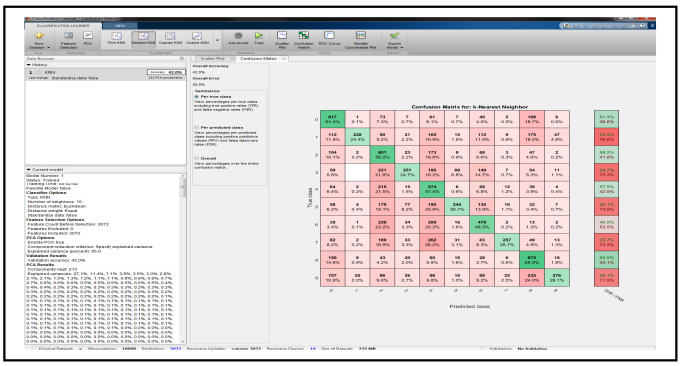
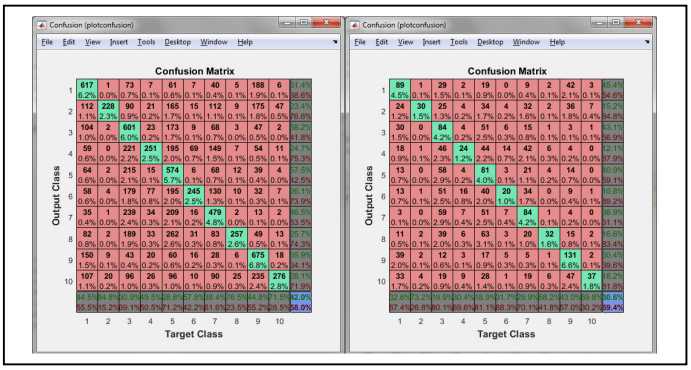


Fig. 13 shows the confusion matrixes of the 10NN model run on the training and test datasets with no validation using PCA as presented using the Matlab plotconfusion function.

Fig. 13. 10NN training (left) & test (right) confusion matrix comparison-PCA



The trained confusion matrix on the left shows higher overall accuracy than the test confusion matrix on the right with percentages in each cell calculated relative to the entire confusion matrix also generally higher in the trained confusion matrix but reasonably tracking them in the test confusion matrix.

Receiver Operating Characteristic (ROC)

Fig. 14 shows the ROC curve for class 0. The marker on the plot shows the performance of the classifier model. The marker shows the values of the false positive rate (FPR) and the true positive rate (TPR) for the classifier model.

Fig. 14. Receiver operating characteristic (ROC) curve for class 0 - PCA

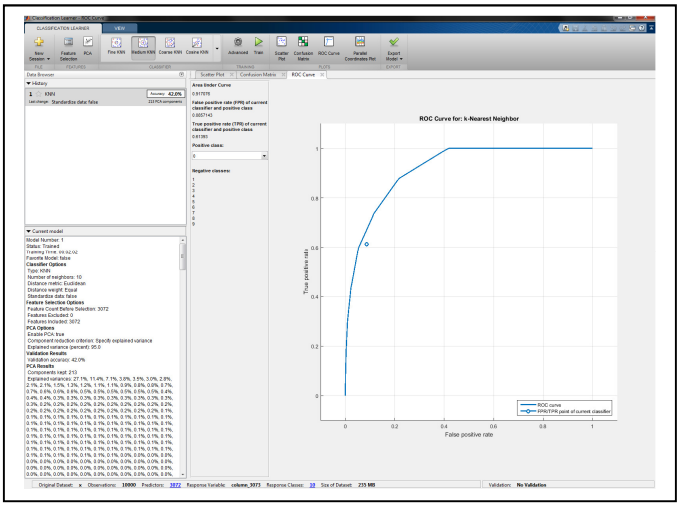
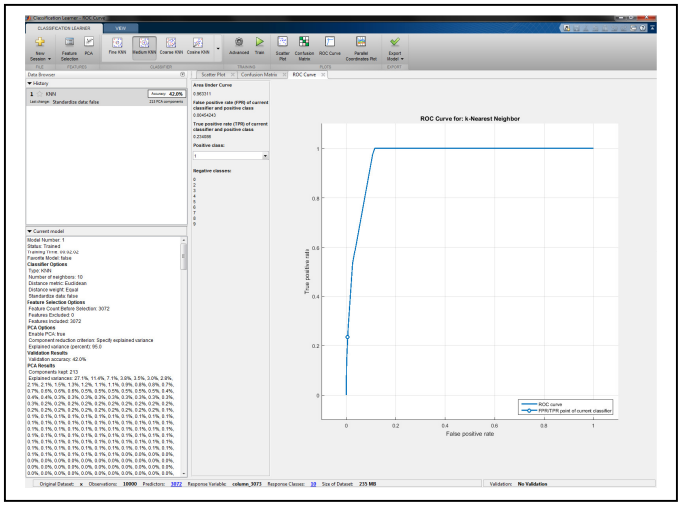


Fig. 15 shows the ROC curve for class 1. With the AUC of 0.96 for class 1 vs 0.92 for class 0, the trained classifier model better classifies class 1 classes over class 0. 0.96 for class 1 vs 0.92 for class 0, the trained classifier model better classifies class 1 classes over class 0.

Fig. 15. Receiver operating characteristic (ROC) curve for class 1 - PCA



A new curve can be plotted for each class. AUC's of the ROC curves for each class can be used to determine which class gets classified best with the trained classifier model.

2) Support Vector Machines (SVM)

a) SVM on Images (without PCA)

PREPARE DATA

Use the data from the first 10,000 image data files, data_batch_1.mat and put it into a single 10,000 image test data set.

PRE-PROCESS DATA

Center the data by subtracting the mean, $\boldsymbol{x}$, from each of the data points.

$$\bar{x} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n \quad (1)$$

$$\boldsymbol{y}_n = \boldsymbol{x}_n - \bar{\boldsymbol{x}} \quad (2)$$

Scale the data so that it has unit standardization.

$$\boldsymbol{\sigma_i^2} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{y_{ni}^2} \quad (3)$$

$$\boldsymbol{z}_n = \boldsymbol{y}_n ./ \boldsymbol{\sigma_{ni}} \quad (4)$$

Matlab Classification Learner (MCL) also expects its data in a table format with data samples in rows and predictors and responses in the columns.

Add the mat file data and label variables to a new variable to use in MCL. Add the response column to the end of the 3,072 image predictor coordinates.

```
x = zeros(10000,3073);
x(1:10000,1:3072) = data(1:10000,1:3072);
x(1:10000,3073) = labels(1:10000,1);
```

Start MCL and click New Session, From Workspace to select data from the workspace and to specify a validation scheme. Since the variable *x* was set up correctly as a table it shows up in Step 1 of MCL and its columns are listed in Step 2 defaulted as predictors.

In Step 2, scroll down to the last row, named column_3073 and click the Import As drop-down and select it as a Response.

VALIDATE

Continuing from the pre-processing section above, confirm that Step 3 is set for the Cross Validation with 5 folds default.

For k fold cross validation, MCL:

1. Partitions the data into k disjoint sets or folds
2. For each fold:
    a. Trains a model using the out-of-fold observations
    b. Assesses model performance using in-fold data
3. Calculates the average test error over all folds

Click the Start Session button and the MCL user interface opens.

5-fold cross validation was run with SVM box constraint level=1 and SVM box constraint level=10 models to find the model with the lowest average training set accuracy.

From the Classifier drop-down, select the Linear SVM. Click the Advanced menu selection and uncheck Standardize Data. The Advanced setting are than set for:

1. Kernel function: Linear
2. Box constraint level: 1
3. Kernel scale mode: Auto
4. Multiclass method: One-vs-One
5. Standardize data checked

Next run the SVM box constraint level=10 model using 5-fold cross validation. Table V tabulates the box constraint levels and accuracy for each. The SVM box constraint level=1 model is optimum in this case with the highest accuracy = 39.2%.

TABLE V. OPTIMUM BOX CONSTRAINT LEVEL FOR LINEAR SVM MODELS (NO PCA)

Optimum box constraint level for linear SVM models (no PCA)		
SVM Model	***Box constraint level***	***Accuracy (%)***
Linear	**1**	**39.2**
Linear	10	37.5

TRAIN

Once the best model is found, train it with that optimum box constraint level with the full training set and no validation. Validation is not necessary at this stage for the training set.

Apply the linear SVM classifier algorithm to the normalized training dataset.

Train the linear SVM model with the optimum box constraint level = 1 and no validation. Standardize the data.

After training, export the model from MCL into a structure called trainedSVM1vs1noPCAClassifier in the Matlab workspace to use for testing.

TEST

Apply the linear SVM model with the optimal box constraint level to the test dataset.

Cross validation showed box constraint level = 1 to be the optimum linear SVM model.

Rerun the exported training model on the original training dataset and confirm that the confusion matrix matches the one derived during training. (see the Evaluation section below to see the plots)

Test the model using the first 2,000 samples of the test_batch.mat test dataset.

EVALUATE

Calculate the training and test dataset accuracy and tabulate and plot the confusion matrix and the ROC (receiver operating characteristic) curves.

Accuracy measures the number of true results as a percentage of all data samples examined. The cross validation accuracy is the averaged in-fold accuracy calculated over the k out-of-fold training models.

Table VI shows the accuracy of the linear SVM model. As expected, training accuracy without validation is the best. The validation accuracy estimates a model's performance on new data compared to the training data and is close to the test accuracy.

TABLE VI. LINEAR SVM MODEL ACCURACY (NO PCA)

Linear SVM Model Accuracy (no PCA)		
Dataset type	***Validation***	***Accuracy (%)***
Training	None	49.0
Training	5 fold cross validation	39.2
Test	None	39.6

Overall 39.6 is not very good performance for a classifier.

Though linear SVM is not as good as convolutional neural networks (CNN) for image classification it performs better than the kNN model performance.

Confusion Matrixes

Fig. 16 shows the confusion matrix of the linear SVM model run on the training dataset with validation and no PCA.

Fig. 16. Linear SVM training dataset confusion matrix - no PCA

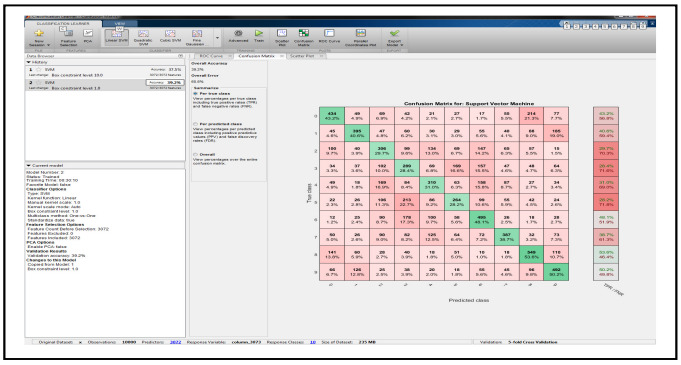
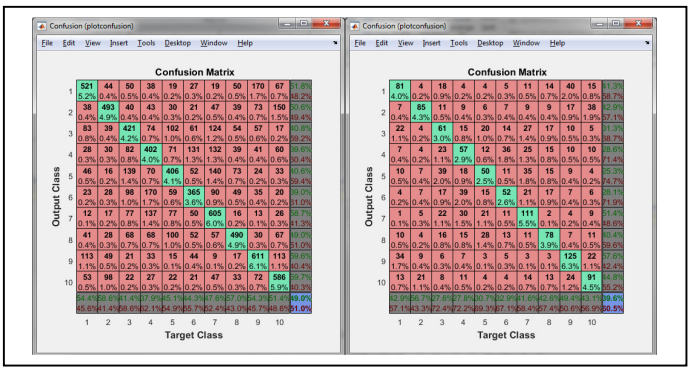


Fig. 17 shows the confusion matrixes of the linear SVM model run on the training and test datasets with no validation and no PCA as presented using the Matlab plotconfusion function.

Fig. 17. SVM training (left) & test (right) confusion matrix comparison



The trained confusion matrix on the left shows higher overall accuracy than the test confusion matrix on the right with percentages in each cell calculated relative to the entire confusion matrix also generally higher in the trained confusion matrix but reasonably tracking them in the test confusion matrix.

Receiver Operating Characteristic (ROC)

Fig. 18 shows the ROC curve for class 0. The marker on the plot shows the performance of the classifier model. The marker shows the values of the false positive rate (FPR) and the true positive rate (TPR) for the classifier model.

Fig. 18. Receiver operating characteristic (ROC) curve for class 0 – no PCA

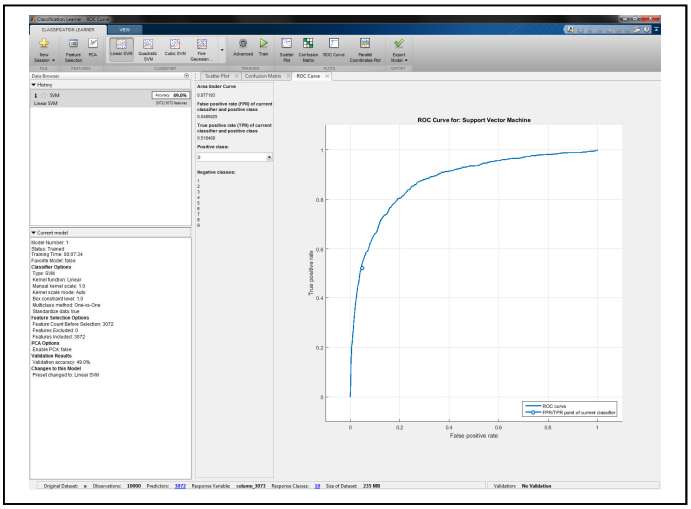
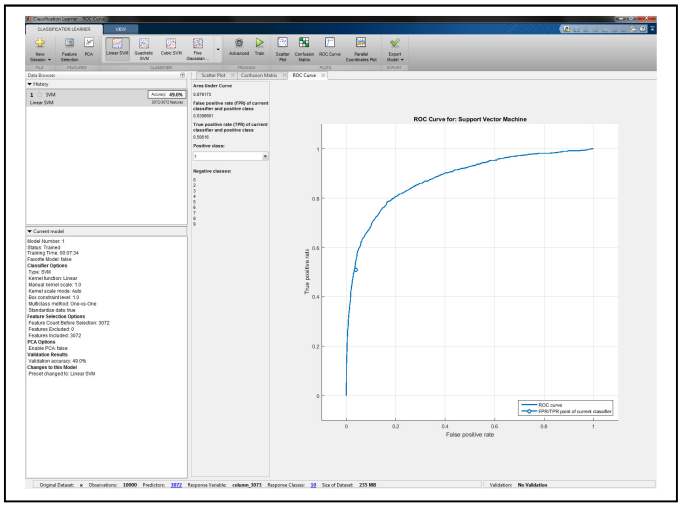


Fig. 19 shows the ROC curve for class 1. With the AUC of 0.879 for class 1 vs 0.877 for class 0, the trained classifier model slightly better classifies class 1 classes over class 0. 0.879 for class 1 vs 0.877 for class 0, the trained classifier model better classifies class 1 classes over class 0.

Fig. 19. Receiver operating characteristic (ROC) curve for class 1 – no PCA



A new curve can be plotted for each class. AUC's of the ROC curves for each class can be used to determine which class gets classified best with the trained classifier model.

b) SVM on Principal Components (using PCA)

PREPARE DATA

Use the data from the first 10,000 image data files, data_batch_1.mat and put it into a single 10,000 image test data set.

PRE-PROCESS DATA

Center the data by subtracting the mean, $\boldsymbol{x}$, from each of the data points.

$$\bar{x} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n \quad (1)$$

$$\boldsymbol{y}_n = \boldsymbol{x}_n - \bar{\boldsymbol{x}} \quad (2)$$

Scale the data so that it has unit standardization.

$$\boldsymbol{\sigma_i^2} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{y_{ni}^2} \quad (3)$$

$$\boldsymbol{z}_n = \boldsymbol{y}_n ./ \boldsymbol{\sigma_{ni}} \quad (4)$$

Matlab Classification Learner (MCL) also expects its data in a table format with data samples in rows and predictors and responses in the columns.

Add the mat file data and label variables to a new variable to use in MCL. Add the response column to the end of the 3,072 image predictor coordinates.

```
x = zeros(10000,3073);
x(1:10000,1:3072) = data(1:10000,1:3072);
x(1:10000,3073) = labels(1:10000,1);
```

Start MCL and click New Session, From Workspace to select data from the workspace and to specify a validation scheme. Since the variable *x* was set up correctly as a table it shows up in Step 1 of MCL and its columns are listed in Step 2 defaulted as predictors.

In Step 2, scroll down to the last row, named column_3073 and click the Import As drop-down and select it as a Response.

VALIDATE

Continuing from the pre-processing section above, confirm that Step 3 is set for the Cross Validation with 5 folds default.

For k fold cross validation, MCL:

1. Partitions the data into k disjoint sets or folds
2. For each fold:
    a. Trains a model using the out-of-fold observations
    b. Assesses model performance using in-fold data
3. Calculates the average test error over all folds

Click the Start Session button and the MCL user interface opens.

5-fold cross validation was run with SVM box constraint level=1 and SVM box constraint level=10 models to find the model with the lowest average training set accuracy.

From the Classifier drop-down, select the Linear SVM. Click the Advanced menu selection and uncheck Standardize Data. The Advanced setting are than set for:

1. Kernel function: Linear
2. Box constraint level: 1
3. Kernel scale mode: Auto
4. Multiclass method: One-vs-One
5. Standardize data checked

Standardized data is used to scale coordinate distances. It was manually applied before performing PCA. This check box would apply it after PCA.

Next run the SVM box constraint level=10 model using 5-fold cross validation. Table VII tabulates the box constraint levels and accuracy for each. The SVM box constraint level=10 model is optimum in this case with the highest accuracy = 37.1%.

TABLE VII. OPTIMUM BOX CONSTRAINT LEVEL FOR LINEAR SVM MODELS - PCA

Optimum box constraint level for linear SVM models - PCA		
SVM Model	***Box constraint level***	***Accuracy (%)***
Linear	1	37.0
Linear	**10**	**37.1**

TRAIN

Once the best model is found, train it with that optimum box constraint level with the full training set and no validation. Validation is not necessary at this stage for the training set.

Apply the linear SVM classifier algorithm to the normalized training dataset.

Train the linear SVM model with the optimum box constraint level = 10 and no validation. Standardize the data.

After training, export the model from MCL into a structure called trainedSVMwithPCAClassifier in the Matlab workspace to use for testing.

TEST

Apply the linear SVM model with the optimal box constraint level to the test dataset.

Cross validation showed box constraint level = 10 to be the optimum linear SVM model.

Rerun the exported training model on the original training dataset and confirm that the confusion matrix matches the one derived during training. (see the Evaluation section below to see the plots)

Test the model using the first 2,000 samples of the test_batch.mat test dataset.

EVALUATE

Calculate the training and test dataset accuracy and tabulate and plot the confusion matrix and the ROC (receiver operating characteristic) curves.

Accuracy measures the number of true results as a percentage of all data samples examined. The cross validation accuracy is the averaged in-fold accuracy calculated over the k out-of-fold training models.

Table VIII shows the accuracy of the linear SVM model using PCA. As expected, training accuracy without validation is the best. The validation accuracy estimates a model's performance on new data compared to the training data and is close to the test accuracy.

TABLE VIII. LINEAR SVM MODEL ACCURACY (USING PCA)

Linear SVM Model Accuracy (using PCA)		
Dataset type	***Validation***	***Accuracy (%)***
Training	None	51.3
Training	5 fold cross validation	37.1
Test	None	38.2

Overall 38.2 is not very good performance for a classifier.

Though linear SVM is not as good as convolutional neural networks (CNN) for image classification it performs better than the kNN model for this simple image classification.

Confusion Matrixes

Fig. 20 shows the confusion matrix of the linear SVM model run on the training dataset with validation and using PCA.

Fig. 20. Linear SVM training dataset confusion matrix using PCA

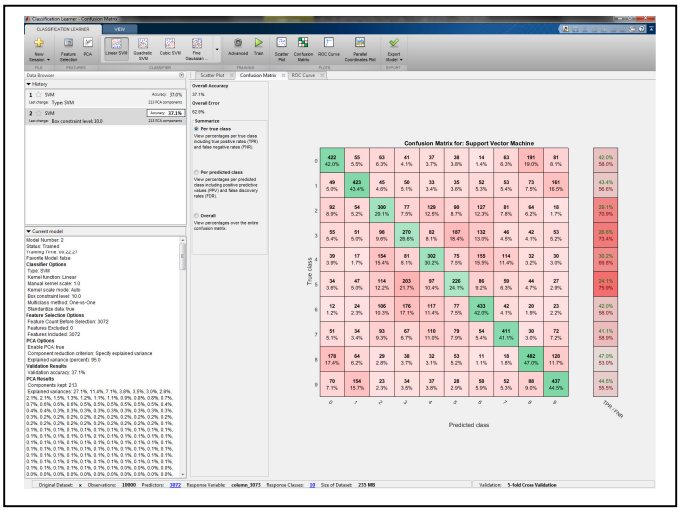
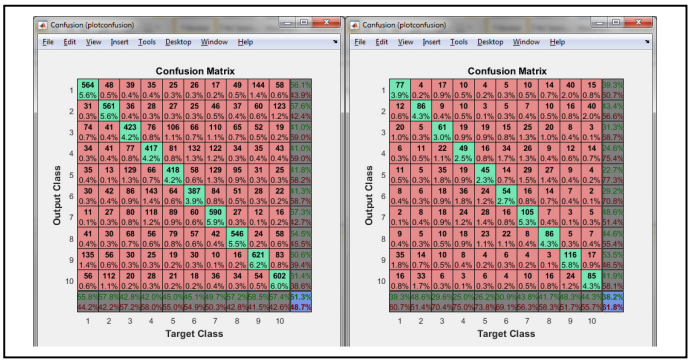


Fig. 21 shows the confusion matrixes of the linear SVM model run on the training and test datasets with no validation and using PCA as presented using the Matlab plotconfusion function.

Fig. 21. SVM training (left) & test (right) confusion matrix comparison - PCA



The trained confusion matrix on the left shows higher overall accuracy than the test confusion matrix on the right with percentages in each cell calculated relative to the entire confusion matrix also generally higher in the trained confusion matrix but reasonably tracking them in the test confusion matrix.

Receiver Operating Characteristic (ROC)

Fig. 22 shows the ROC curve for class 0. The marker on the plot shows the performance of the classifier model. The marker shows the values of the false positive rate (FPR) and the true positive rate (TPR) for the classifier model.

Fig. 22. Receiver operating characteristic (ROC) curve for class 0 - PCA

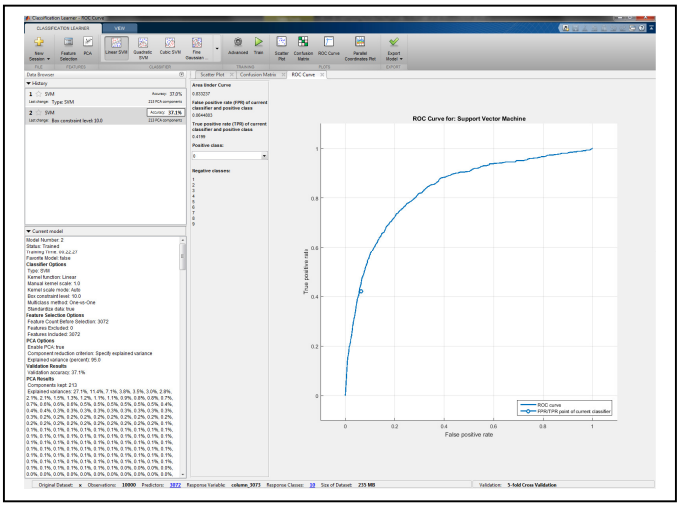
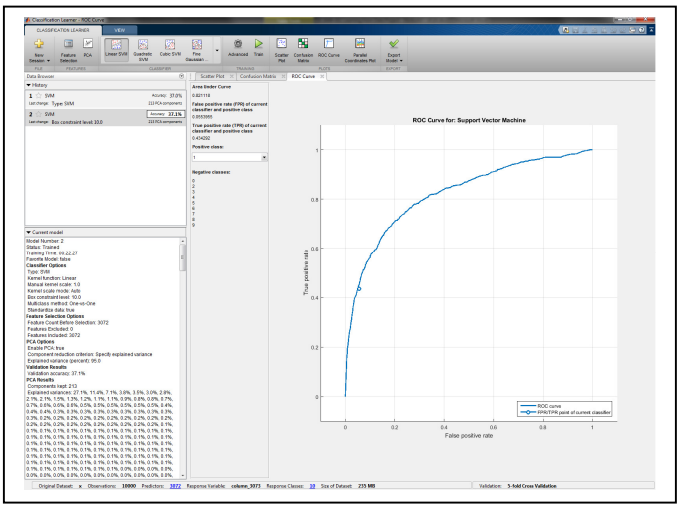


Fig. 23 shows the ROC curve for class 1. With the AUC of 0.821 for class 1 vs 0.833 for class 0, the trained classifier model better classifies class 0 classes over class 0. 0.821 for class 1 vs 0.833 for class 0, the trained classifier model better classifies class 0 classes over class 1.

Fig. 23. Receiver operating characteristic (ROC) curve for class 1 - PCA



A new curve can be plotted for each class. AUC's of the ROC curves for each class can be used to determine which class gets classified best with the trained classifier model.

V. MATLAB INSTRUCTIONS

Set up and work with the Matlab environment as follows:

1. Set up a project folder and add it to the Matlab path and make it the Current Folder in Matlab
2. Add the following extracted data files described in section IV.A to the project folder:
    a. data_batch_1.mat - data_batch_5.mat
    b. test_batch.mat
3. Add the following Matlab scripts to that folder:
    a. ppdata.m
    b. test.m
4. Double-click on the data_batch_1.mat data file in the Matlab Current Folder so that the following variables are loaded into the Matlab Workspace:
    a. batch_label
    b. data
    c. labels
5. Run the ppdata script that normalizes the data and appends the labels so it is the right format for the Matlab Classification Learner app. The data is stored in the "x" variables that gets added to the Matlab Workspace. ppdata outputs a small sample of data including the last column of class values.
6. Open the Matlab Classification Learner (MCL) app
7. Start a New Session, From the Workspace
8. Select the "x" variable in Step 1
9. Scroll down to column_3073 and switch its Import as value to Response in Step 2
10. Select either cross or no validation in step 3; use cross validation for tuning models and no validation for training
11. Select the desired classifier (kNN or SVM), set the appropriate Advanced settings, and select PCA if required
12. Click Train
13. After the model runs, review its accuracy in the History window, its settings in the Current model window, and Confusion Matrix and ROC plots; save screenshots as desired
14. Adjust tuning parameters and run again; select optimal model based on validation accuracy
15. Close the Classification Learner, and reopen and train the optimal model with no validation; save screenshots as desired
16. Select Export Compact Model and use the default name, TrainedClassifier; TrainedClassifier will show up as a variable in the Matlab Workspace
17. Run the Matlab test.m script; it will operate on the data and labels variables added to the Matlab Workspace in step 4; initially they are from the data_batch_1.mat
18. Review the confusionmat output and the plotconfusion plot to confirm it matches the confusion matrix results from the optimal, un-validated model run
19. Double-click on the test_batch.mat data file in the Matlab Current Folder so that the following variables are overwritten in the Matlab Workspace:
    a. batch_label
    b. data
    c. labels
20. Edit the test.m script and change the N value to 2000
21. Run test.m; it now runs against 2000 samples of the test data and produces the confusionmat output and the the plotconfusion plot; save screenshots as desired
22. Rerun for other models and settings

VI. CONCLUSION

k - Nearest Neighbor Classifier (kNN) and linear Support Vector Machines (SVM) algorithms were trained using Principal Component Analysis (PCA) and also without it.

Classification accuracy was calculated and calculation time noted and compared across all models and shown in Table IX.

TABLE IX. SIMPLE IMAGE CLASSIFICATION (WITH AND WITHOUT PCA)

Simple Image Classification Model (with and without PCA)			
Model	***PCA***	***Accuracy (%)***	***Time (hr:min:sec)***
kNN	No	29.5	00:05:45
kNN	Yes	30.6	00:10:51
Linear SVM	**No**	**39.6**	**00:30:10**
Linear SVM	Yes	38.2	00:22:27

The linear SVM model without using PCA performed best with the highest accuracy though it took the longest time to train.

REFERENCES

[1] A. Karpathy. (2015, Nov 8). *CS231n: Convolutional Neural Networks for Visual Recognition, Image Classification and Nearest Neighbor Classifier* [Online]. Available: http://cs231n.github.io/classification/

[2] M. Mudrova and A. Prochazka. (2004). *Principal Component Analysis in Image Processing* [Online]. Available: http://staff.ustc.edu.cn/~zwp/teach/MVA/pcaimg.pdf

[3] I. Bajwa *et al*. (2008, Jan). *Feature Based Image Classification by using Principal Component Analysis* [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.301.6363&rep=rep1&type=pdf

[4] M. Turk, A. Pentland. (1991). *Eigenfaces for Recognition* [Online]. Available: http://www.face-rec.org/algorithms/PCA/jcn.pdf

[5] R. Rigamonti *et al*. (2011). *Are Sparse Representations Really Relevant for Image Classification?* [Online]. Available: http://cvlabwww.epfl.ch/~lepetit/papers/rigamonti_cvpr11.pdf

[6] R. Rigamonti *et al*. (2014). *On the relevance of sparsity for image classification* [Online]. Available: http://infoscience.epfl.ch/record/197897/files/Rigamonti_CVIU2014.pdf

[7] A. Krizhevsky. (2015, Nov 8). *The CIFAR-10 dataset* [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[8] A. Krizhevsky. (2009, Apr 8). *Learning Multiple Layers of Features from Tiny Images* [Online]. Available: http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[9] Matlab. *Plotconfusion function* [Online]. Available: http://www.mathworks.com/help/nnet/ref/plotconfusion.html?refresh=true

[10] Matlab. *Assess Classifier Performance* [Online]. Available: http://www.mathworks.com/help/stats/assess-classifier-performance.html

[11] Matlab. *Feature Selection and Feature Transformation* [Online]. Available: http://www.mathworks.com/help/stats/feature-selection-and-feature-transformation.html

[12] Matlab. *Choose a Classifier* [Online]. Available: http://www.mathworks.com/help/stats/choose-a-classifier.html