STA142A Final Project

3/18/2022

Kevin Xu's Contribution:

- Wrote code for KNN from scratch
- Wrote code for K-fold Cross validation.
- Determined optimal K in KNN.

Jasper Tsai's Contribution:

- SVM implementation on non-PCA data.
- Tune hyperparameter for polynomial kernel SVM.
- Report formatting and notebook condensing

Wyatt Workman's Contribution:

- Data preprocessing, including image resizing and creation of data matrix.
- Compute principal component (perform PCA from scratch).
- Wrote code for convolutional neural network (CNN) model on non-PCA data.

1 Introduction

For this course, we focused our studies on supervised machine learning methods. In this report, we attempt to apply multi-class classification models to classify five different groups of vehicles. We will be implementing K-Nearest Neighbors from scratch after dimension reduction with principal component analysis (PCA), then we will compare our results with Neural Network and Kernel SVM without PCA. In doing this, our goal is to make comparisons between the use of manual feature extraction techniques, such as PCA, and automatic feature extraction techniques, such as support vector machines (SVMs) and neural networks.

We expect the automatic feature extraction techniques to yield better results in terms of prediction accuracy. This is because when using manual feature extraction techniques such as PCA, it is typical to only keep enough components to explain 80-90% of the variance in the original dataset. By contrast, we can account for all of the variance with automatic feature extraction, since we can run the model on a subset of the original dataset, rather than a dataset of principal components.

2 Methodology

Here we present a detailed description of the various methods used in data preprocessing, manual feature extraction, model fitting, and automatic feature extraction.

2.1 Data Preprocessing

Before we began building predictive models, some initial data preprocessing was required. Our original dataset contained approximately 78,000 color images of three different dimensions. First, we decided to grayscale the images. This allowed us to eliminate the color dimension of each image, reducing the amount of data that would need to be stored, and later, processed. Next, we decided to focus on five specific classes of vehicles, rather than the 23 total classes in the dataset. We specifically chose five classes that had a similar number of observations, in order to avoid any potential class imbalance issues. Finally, the images were resized in order to ensure that all of the images in our dataset were of the same size. This was accomplished by choosing the smallest image size in the dataset, and resizing all of the images to this size. This was a necessary step to ensure compatibility with our models. The end result was a subsetted dataset that contained approximately 25000 grayscale images of vehicles belonging to five different classes.

2.2 Principal Component Analysis

After completing the preprocessing steps, we decided to apply principal component analysis (PCA) to our dataset. Our objective was to extract the most important features and use these to transform our dataset. This was accomplished by calculating the eigenvalues and eigenvectors of the covariance matrix of the standardized data. We then computed the total cumulative variance explained by each component using the corresponding eigenvalues.

2.3 K-Nearest Neighbors

For our manual feature extraction model, we decided to employ the nonparametric supervised k-nearest-neighbors model. KNN was attractive to us for its heuristic-type simplicity for implementation and its robustness to real data. The idea behind this model is to (for each data point $\overrightarrow{x_j}, j = 1, ...n, \overrightarrow{x_j} \in R^p$ in our testing set) identify k data points from the training set that are closest. We then identify the "majority" label associated with these k "neighbors". Our test data point would then be classified as the majority label among the k neighbors. We decided to implement KNN with two types of Minkowski distance metrics: Manhattan distance (L1 norm), and Euclidean distance (L2 norm). Formally, this can process can be understood as:

$$y' = argmax_v \sum_{(x_i, y_i) \in D'} (v = y_i),$$

, where y': class assigned to test observation, D': the set of k neighbors in the training set, v: count of a given class.

2.4 Support Vector Machine with Polynomial Kernel

We chose to implement a support vector machine (SVM) with polynomial kernel as one of our automatic feature extraction for comparison. In simple terms, SVM performs data transformations based on the selected kernel function. Using the transformations, it tries to maximize the separation boundaries (hyperplane) between the data points depending on the classes defined. SVM does not support multi-class classification natively. For multi-class classification, the same principle is utilized; the approach is to break down into multiple binary classification problems. There are two approaches, One-to-One and One-to-Rest. One-to-One breaks down the multi-class problem into multiple binary classification problems. A binary classifier per each pair of classes. For One-to-Rest, the breakdown is set to a binary classifier per each class. Assume we want to classify m labels, for One-to-Rest , the classifier use m SVMs and in One-to-One approach, classifier uses $\frac{m(m-1)}{2}$ SVMs.

2.5 Convolutional Neural Network

We also chose to employ a convolutional neural network (CNN) for one of the methods of automatic feature extraction. This is advantageous for image classification, as CNN model architectures are explicitly designed for image classification. The figure below shows the architecture of the model we will employ for the scope of this project.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 98, 73, 64)        640

 max_pooling2d (MaxPooling2D  (None, 49, 36, 64)       0
 )

 batch_normalization (BatchN  (None, 49, 36, 64)       256
 ormalization)

 conv2d_1 (Conv2D)           (None, 47, 34, 32)        18464

 max_pooling2d_1 (MaxPooling  (None, 23, 17, 32)       0
 2D)

 batch_normalization_1 (Batc  (None, 23, 17, 32)       128
 hNormalization)

 conv2d_2 (Conv2D)           (None, 21, 15, 16)        4624

 flatten (Flatten)           (None, 5040)              0

 dropout (Dropout)           (None, 5040)              0

 dense (Dense)               (None, 5)                 25205

=================================================================
Total params: 49,317
Trainable params: 49,125
Non-trainable params: 192
_________________________________________________________________
```

This model is made up of an input layer that accepts the grayscale image data matrix, several hidden layers that perform the convolutions and compute the model weights, and a fully connected dense layer that computes the class scores and assigns a class label on the basis of the index of the largest class score.

To compile and train this model, we use the Adam optimization function. When compared with a stochastic gradient descent optimization function, Adam dynamically chooses a learning rate for each parameter, whereas stochastic gradient descent uses a constant learning rate for all weights and does not change during model training. Using the Adam optimization function allows us to obtain good results much faster. Finally, we set up an earlyStopping parameter that automatically ends training if no significant improvement in accuracy is seen over a pre-specified number of training epochs, in our case, 5 epochs.

3 Implementation Details

Here we describe the process of how we implemented the techniques outlined in the previous section, as well as mention which python libraries were used.

3.1 Principal Component Analysis

We chose to implement PCA from scratch for the scope of this project, in order to show our deep understanding of the underworkings of PCA. Specifically, we utilized the Numpy libarary extensively in order to standardize our data, compute the covariance matrix of this data, and to compute the eigenvalues and eigenvectors of the covariance matrix. In order to produce a plot of cumulative explained variance, we used the Plotnine library, which is a python wrapper of the ggplot library in R.

3.2 K-Nearest-Neighbors

We wrote two functions to implement knn: 'knn.predict()', and 'knn.ktune()'. 'predict' takes in arguments: k (neighbors). distance metric, and returns the prediction accuracy conducted on the test set. predict() only uses the Numpy package. 'ktune' tunes hyperparameter k by using 3-fold cross validation to fit a model for k values in the set: $k = 1, 3, 5, 7, 9$. Odd k values were selected to avoid ties during the voting process, and 3 folds were selected to retain as much data as possible within our training and testing partitions. 'ktune' only uses sklearn to split our training set into partitions.

3.3 Support Vector Machine with Polynomial Kernel

To implement Support Vector Machine(SVM), we utilized SKLEARN library to split data into test and validation sets with a 80:20 split of train and test data. We also employed a GridSearchCV to tune the hyperparameters C (Regularization parameter) and degree (degree of the polynomial kernel function)

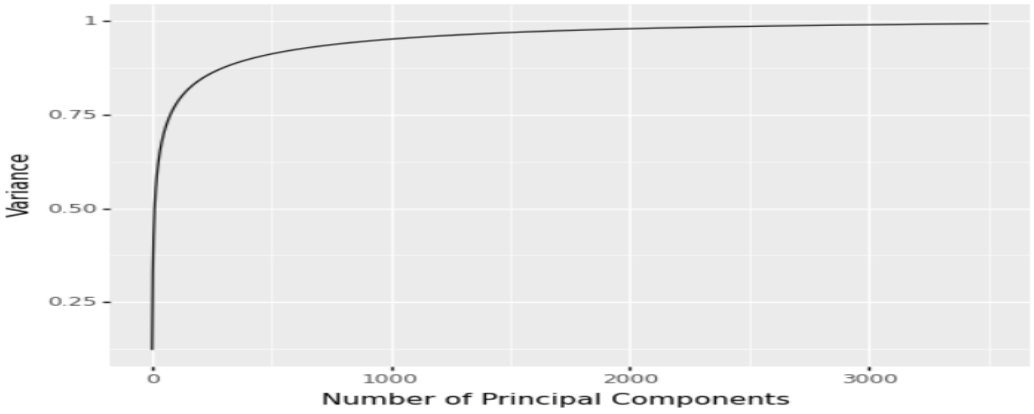
3.4 Convolutional Neural Network

Finally, we make use of a convolutional neural network (CNN) to perform both automatic feature extraction and to build a predictive model. We make extensive use of the Tensorflow and Keras libraries to build and compile the model structure. We also use the SKLEARN library to split the data into test and validation sets, utilizing an 80/20% split of training and testing data, respectively.

4 Results and Interpretation

4.1 Principal Component Analysis

The following plot shows how the proportion of explained variance changes as more components are considered.



Based on this plot, we decided it would be most appropriate to keep the first 280 components, which would account for approximately 87.07% of the variance in our subsetted data. Multiplying our original data matrix by the matrix of the first 280 principal components allowed us to reduce the number of features from 7500 to 280. Theoretically, this would mean that our models would run faster, since we reduce the total amount of data.

4.2 K-Nearest-Neighbors

The following table shows the prediction accuracy and run time from the implementation of kNN. The run time refers to the training time after hyper parameters were tuned.

Distance Metric	k	Test Accuracy	Run time
Manhattan	9	39.45%	1136.1 seconds
Euclidean	9	39.08%	1157.3 seconds

We chose k=9 because it had the highest mean prediction accuracy among k=1,3,5,7,9 across the three folds that our model was fitted per value k. This was the case for both L1 and L2 norm.

4.3 Support Vector Machine with Polynomial Kernel

After tuning the hyperparameters, the best model is $C = 10$ and degree $= 3$ for our Polynomial Kernel SVM. We were able to achieve an accuracy of 52.5% with the best model. For further details consult the ipython notebook or supplement pages.

4.4 Convolutional Neural Network

Our CNN model ran for approximately 45 minutes during training, and ran for 13 epochs. We were able to achieve a training accuracy of 69% and a validation accuracy of 58%. Further details can be seen in the figure below.

```
Epoch 1/500
631/631 [==============================] - 93s 147ms/step - loss: 1.3943 - Accuracy: 0.4227 - val_loss: 1.1847 - val_
Accuracy: 0.4979
Epoch 2/500
631/631 [==============================] - 92s 147ms/step - loss: 1.1601 - Accuracy: 0.5231 - val_loss: 1.1206 - val_
Accuracy: 0.5344
Epoch 3/500
631/631 [==============================] - 93s 147ms/step - loss: 1.0822 - Accuracy: 0.5629 - val_loss: 1.1438 - val_
Accuracy: 0.5370
Epoch 4/500
631/631 [==============================] - 94s 148ms/step - loss: 1.0180 - Accuracy: 0.5874 - val_loss: 1.0760 - val_
Accuracy: 0.5498
Epoch 5/500
631/631 [==============================] - 93s 148ms/step - loss: 0.9701 - Accuracy: 0.6076 - val_loss: 1.0496 - val_
Accuracy: 0.5683
Epoch 6/500
631/631 [==============================] - 94s 149ms/step - loss: 0.9249 - Accuracy: 0.6293 - val_loss: 1.0401 - val_
Accuracy: 0.5633
Epoch 7/500
631/631 [==============================] - 94s 149ms/step - loss: 0.8751 - Accuracy: 0.6526 - val_loss: 1.1183 - val_
Accuracy: 0.5556
Epoch 8/500
631/631 [==============================] - 94s 149ms/step - loss: 0.8524 - Accuracy: 0.6605 - val_loss: 1.0515 - val_
Accuracy: 0.5803
Epoch 9/500
631/631 [==============================] - 94s 149ms/step - loss: 0.8167 - Accuracy: 0.6747 - val_loss: 1.0517 - val_
Accuracy: 0.5718
Epoch 10/500
631/631 [==============================] - 94s 149ms/step - loss: 0.7852 - Accuracy: 0.6892 - val_loss: 1.0810 - val_
Accuracy: 0.5708
Epoch 11/500
631/631 [==============================] - 96s 152ms/step - loss: 0.7623 - Accuracy: 0.6957 - val_loss: 1.1020 - val_
Accuracy: 0.5853
<keras.callbacks.History object at 0x1711092b0>
```

5 Conclusions

The following table shows the rankings of several model assessment criterion, which describe the overall performance of our models, from most to least.

Prediction Accuracy	Run Time	Interpretability	Generalizability
CNN	KNN	KNN	SVM
SVM	SVM	SVM	KNN
KNN	CNN	CNN	CNN

As we can see our convolutional neural network (automated dimension reduction) performed the best in terms of prediction accuracy followed by SVM and kNN. This could be attributed to the fact that our SVM and CNN models were trained and validated on the whole dataset, whereas the KNN model was trained on transformed data with a fraction of the total features as the whole dataset. Also, our feature-reduced data only accounted for 87.07% of the total variance in the dataset. This means that, although we accounted for a majority of the variance, there was still some that was "left out". Therefore, this could be a possible reason that our KNN model did not perform as well with respect to accuracy.

On the other hand, our kNN model performed best in terms of run-time as compared to CNN and SVM. Perhaps this is because CNN and SVM is conducted on unreduced data with 7500 dimensions, whereas kNN is performed on PCA data that have 280 dimensions.

Based on these findings, it appears that as compared to manual dimension reduction techniques like PCA, automated dimension reduction techniques bring increased generalizability and prediction accuracy to our model at the cost of training run time.

Supplementary Materials

This section is part of the four pages allocated to supplementary material, as outlined in the report guidelines.

5.1 Results from Hyperparameter Tuning Kernel SVM

Tuning Hyperparameter with C=10 and degree from 1 to 5

```
[CV 1/5] END .......C=10, degree=1, kernel=poly;, score=0.380 total time=  29.7s
[CV 2/5] END .......C=10, degree=1, kernel=poly;, score=0.391 total time=  29.3s
[CV 3/5] END .......C=10, degree=1, kernel=poly;, score=0.401 total time=  29.2s
[CV 4/5] END .......C=10, degree=1, kernel=poly;, score=0.396 total time=  29.3s
[CV 5/5] END .......C=10, degree=1, kernel=poly;, score=0.386 total time=  29.0s
[CV 1/5] END .......C=10, degree=2, kernel=poly;, score=0.494 total time=  32.0s
[CV 2/5] END .......C=10, degree=2, kernel=poly;, score=0.498 total time=  31.7s
[CV 3/5] END .......C=10, degree=2, kernel=poly;, score=0.492 total time=  32.3s
[CV 4/5] END .......C=10, degree=2, kernel=poly;, score=0.490 total time=  32.5s
[CV 5/5] END .......C=10, degree=2, kernel=poly;, score=0.489 total time=  32.6s
[CV 1/5] END .......C=10, degree=3, kernel=poly;, score=0.512 total time=  33.7s
[CV 2/5] END .......C=10, degree=3, kernel=poly;, score=0.517 total time=  33.4s
[CV 3/5] END .......C=10, degree=3, kernel=poly;, score=0.518 total time=  33.5s
[CV 4/5] END .......C=10, degree=3, kernel=poly;, score=0.521 total time=  33.7s
[CV 5/5] END .......C=10, degree=3, kernel=poly;, score=0.497 total time=  34.1s
[CV 1/5] END .......C=10, degree=4, kernel=poly;, score=0.486 total time=  35.3s
[CV 2/5] END .......C=10, degree=4, kernel=poly;, score=0.492 total time=  35.5s
[CV 3/5] END .......C=10, degree=4, kernel=poly;, score=0.484 total time=  35.7s
[CV 4/5] END .......C=10, degree=4, kernel=poly;, score=0.485 total time=  34.5s
[CV 5/5] END .......C=10, degree=4, kernel=poly;, score=0.481 total time=  35.2s
[CV 1/5] END .......C=10, degree=5, kernel=poly;, score=0.438 total time=  36.4s
[CV 2/5] END .......C=10, degree=5, kernel=poly;, score=0.455 total time=  35.8s
[CV 3/5] END .......C=10, degree=5, kernel=poly;, score=0.446 total time=  36.3s
[CV 4/5] END .......C=10, degree=5, kernel=poly;, score=0.445 total time=  36.1s
[CV 5/5] END .......C=10, degree=5, kernel=poly;, score=0.441 total time=  35.5s
```

5.2 Sources and Acknowledgements

- https://www.baeldung.com/cs/svm-multiclass-classification
- https://cs231n.github.io/convolutional-networks/
- https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
- https://www.geeksforgeeks.org/svm-hyperparameter-tuning-using-gridsearchcv-ml/
- https://www.askpython.com/python/examples/principal-component-analysis