Hyperspectral Image Analysis with Supervised Machine Learning Algorithms

Prepared by : **Betül Kul**
Student No : **090180738**
Submission Date : **June 18, 2022**

Course : **MAT 4902E**
Supervisor : **PROF. DR. Atabey Kaygun**

Table of Contents

Table of Figures

1. Introduction

Hyperspectral remote sensing is the process of analysis and evaluation of echoed radiation measures retrieved from a high number of spectral bands on a specific region of Earth called a “scene”. A hyperspectral image (HSI) has a higher spectral resolution than conventional remote sensing images, and is represented as a 3D data cube. Images are provided by imaging spectrometers that have special electromagnetic sensors [1].

Hyperspectral remote sensing analysis is used to predict spatial, and geological surface features of unknown regions in a scene. HSI images are used to classify the spatial and geographic features of a pixel that represents a location on the target surface. Due to fast technological developments in Hyperspectral imaging, there are a number of HSI datasets to be analyzed. One such dataset we used in this study is the Indian Pines dataset and is obtained from the Grupo De Intelligencia (GIC) website “Hyperspectral Remote Sensing Scenes” datasets [3].

The Indian Pine dataset is a hyperspectral remote sensing scene created using 224 spectral bands with the AVIRIS sensor [2], but is reduced to 200 bands to decrease complexity. The Indian Pines dataset that is used in this project is 145x145 pixels with 220 spectral bands. The scene contains two-thirds agriculture, and one-third forest or other natural perennial vegetation. There are two major dual-lane highways, a rail line, as well as some low-density housing, other built structures, and smaller roads.

The image (a) in Figure 1, represents the data cube shape of the HSI image of the Indian pines data. The data consist of 16 different plantation types in the region. The shapes and labels of the plantation regions is given below on the image (b).

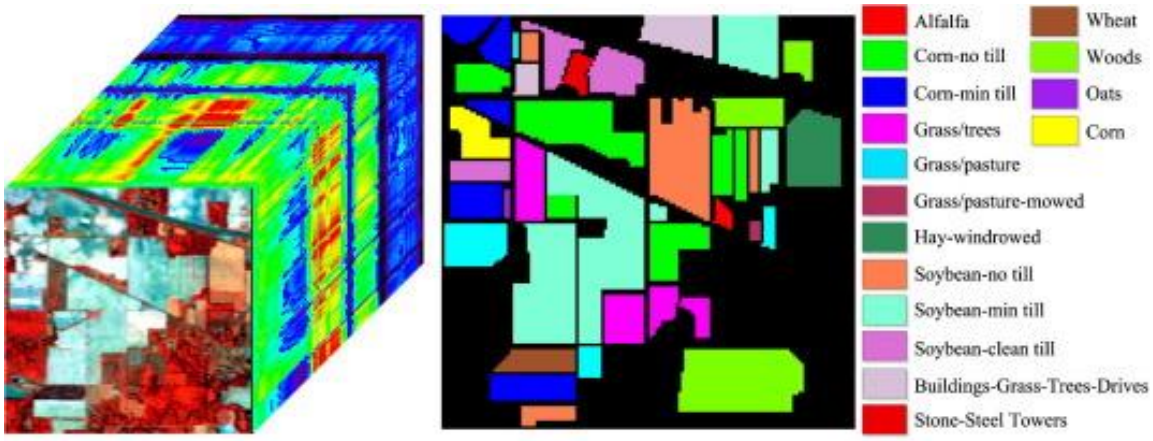

Figure 1: Indian Pines data cube and label visualization.

The 16 labels of the Indian Pines dataset with sample numbers are given below.

Groundtruth classes for the Indian Pines scene and their respective samples number

#	Class	Samples
1	Alfalfa	46
2	Corn-notill	1428
3	Corn-mintill	830
4	Corn	237
5	Grass-pasture	483
6	Grass-trees	730
7	Grass-pasture-mowed	28
8	Hay-windrowed	478
9	Oats	20
10	Soybean-notill	972
11	Soybean-mintill	2455
12	Soybean-clean	593
13	Wheat	205
14	Woods	1265
15	Buildings-Grass-Trees-Drives	386
16	Stone-Steel-Towers	93

Figure 2: Indian Pines label names and sample sizes.

The Project aims to create high accuracy machine learning models on the Indian Pines dataset using supervised machine learning methods and neural network models. The methods that will be used are

1. Support Vector Machines,
2. K-Nearest Neighbour,
3. Multinomial Logistic Regression,
4. Standard Artificial Neural Network, and
5. Convolutional Neural Networks.

The performances of the models will compared in the analysis section using the run time, accuracy score and confusion matrix metrics. In the analysis section the most efficient and best-performing methods will be determined.

The Indian Pines data includes 220 spectral bands in this Project, however some of the spectral bands are reduntant and high number of spectral bands decrease the model performance while increasing complexity. Reducing the spectral bands number without losing the most effective spectral band values can be done using the PCA method. The result of the PCA method with 3 band values will be obtained and compared with the original dataset with 220 bands. The location value of the pixels will also be considered and calculated as an input. In

this project 4 input datasets with different input shapes as variations of the Indian Pines data will be used to create the selected supervised machine learning methods.

2. Methodology

2.1 Principal Component Analysis (PCA) Method

PCA is the method used for reducing the dimensions of high dimensional datasets to the given dimension input. The method is applied by projecting high dimensional data points to a new coordinate system to maximize data variance. The axises of the new coordinate system are called “Principal Components” [4]. The number of principal components are designated by the user. The system determines the principal components by calculating the maximum variance between data points, then takes the vector that has the maximum variance between its end points as “Principal Component 1”. Each component is selected according to the variance order from largest to smallest. The intersection of each axis is the origin of the new coordinate system. An example of a two-dimensional data set with 2 principal components is given below.

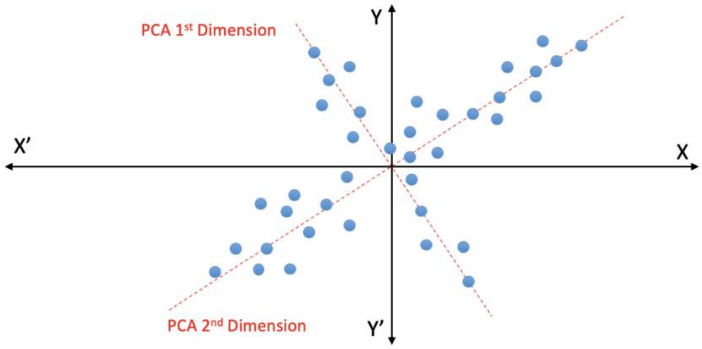

Figure 3: Principal components on 2D dataset.

After deciding the new axises on the data set, the system calculates the "loading" value of each data point. Loading value is based on the distance of each data point to each ends of the principal component. Each end of the component has different signs, the absolute value of the loading value of each data point increases when the points are closer to the ends. The array of loading values for each component is called the "eigenvector". For each dimension, the loading value of a data point on a principal component is multiplied with the measure of the dimension, then the weighted values are summed. The result is taken as the location of the data point on the principal component axis. The calculation is repeated on each component for all data points. As a result of the PCA dimensions of the data is reduced to a desired number.

2.2 Supervised Machine Learning

Supervised learning is a type of machine learning that uses labeled datasets to train algorithms that classify data and predict outcomes accurately. The models are created with

an input that is portioned from the dataset for training. Training dataset includes inputs and correct outputs, which allow the model to learn over time. The algorithm measures its accuracy through the loss function, adjusting until the error has been sufficiently minimized [5]. The algorithm then predicts the classification of the test portion of the dataset according to the trained model. The predictions are then compared with the correct labels.

The accuracy of a model is the evaluation of how the model performs. Accuracy is calculated by dividing the number of correct predictions to the total number of predictions. Common classification algorithms are support vector machines (SVM), k-nearest neighbor, logistic regression, neural networks.

2.2.1 Support Vector Machine (SVM)

Support Vector Machine (SVM) is a kernel-based algorithm that is used to solve classification, regression and novelty detection problems. In classification SVM determines a decision boundary that is used to divide the dataset into two ore more categories. Kernel functions are used to determine the linearity of the decision boundary. Radial Basis Function (RBF) kernel is a commonly used nonlinear kernel function for SVM. If the dataset is three dimensional, hyperplane that seperates the data will be a two dimensional plane, if the data is two dimensional, the decision boundary will be a line. The hyperplane is selected according to the closest data points in each category, these data points that help determine the hyperplane is called "Support Vectors". The distance between the support vectors is called the "margin". In order for the algorithm to find the optimum hyperplane, the model is first trained with the trained portion of the data. From the example given below, it can be seen that there is more than one way to drew a hyperplane that separates the data.

The SVM algorithm selects the optimum divider as the hyperplane which is the one where the margin is maximum. SVM solves the "Convex Optimization Problem" to find the maximum margin, which is why SVM algorithm is also called the "Maximum Margin Classifier". Even when the best possible hyperplane is selected, in some categories there may be data points from other categories which is called "outliers". Regularization parameters, L1 and L2 is used to diminish the detected unimportant outliers as "noise" in order to avoid misclassification. The parameter C is used to control the margin value. When the parameter C is small, the margin is smaller, hence it is algorthim is less tolerant to outliers. When the C parameter is larger, the margin gets wider and more tolerant to outliers. Larger C value may cause some misclassification, however that may be tolerable.

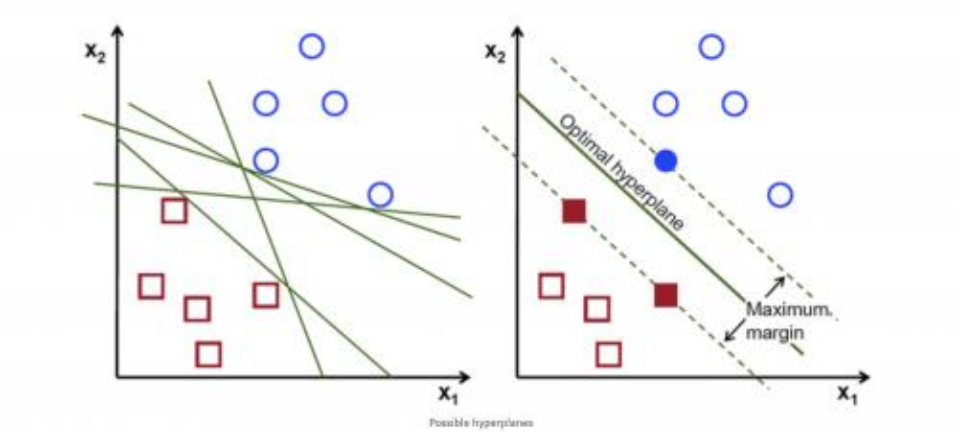

Figure 4: Hyperplane selection on 2D data.

2.2.2 K-Nearest Neighbour (KNN)

K Nearest Neighnour is a supervised classification algorithm that takes the positive integer parameter K to find k number of nearest data points with known categories, to detect the category of an unclassified data point. The classes of the nearest categories are already given to the algorthm in the training process. Then the system detects the category that has the most number of nearest neighbours as the label of the data point that belongs to the test portion of the dataset. K value must be an odd integer in order to avoid even distribution of the nearest neighbours between categories [7]. The parameter K is crucial to the model, therefore an error graph to understand how each K parameter may be drawn to help user detect the optimal K value. An example of the KNN algortihm is given below. According to the example K is designated as 5. The neighbours of the data points with unknown classes from the test sample are shown in the circles. Majority of the neighbours belong to the Class '+' in both cases, hence the test samples are labeled as Class '+'.

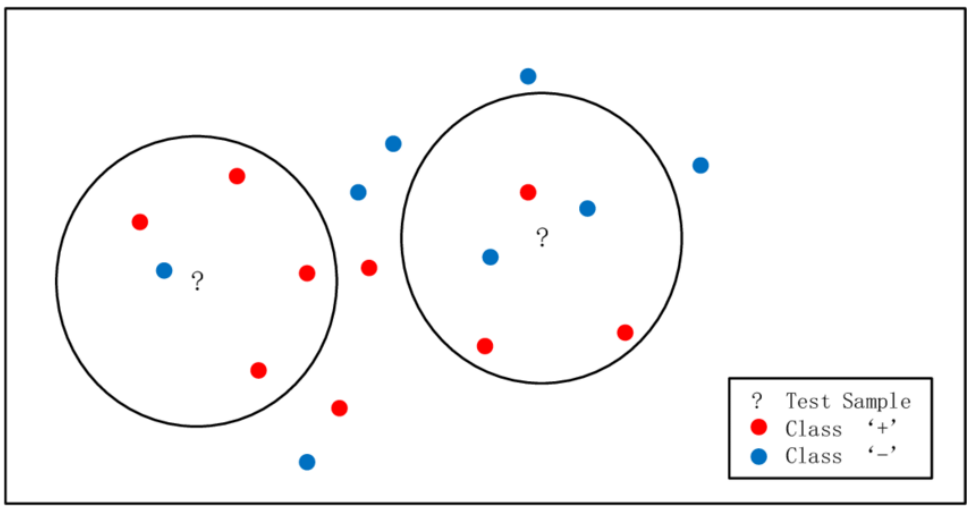

Figure 5: KNN method with K=5.

2.2.3 Logistic Regression

Logistic regression is a supervised machine learning method used for classification problems. When the dataset has 2 labels the model is called "Binary Logistic Regression". Trained data is divided to two classes according to the probability of being in one of the classes. The probability distribution of the data with 2 classes is the sigmoid function. In the image below it is shown that two classes are divided with a decision boundary of 0.5 probability. The 2 classes represented as 1 or 0. If a data point has a probability over 0.5 of being in class with 1, then it is labeled 1.

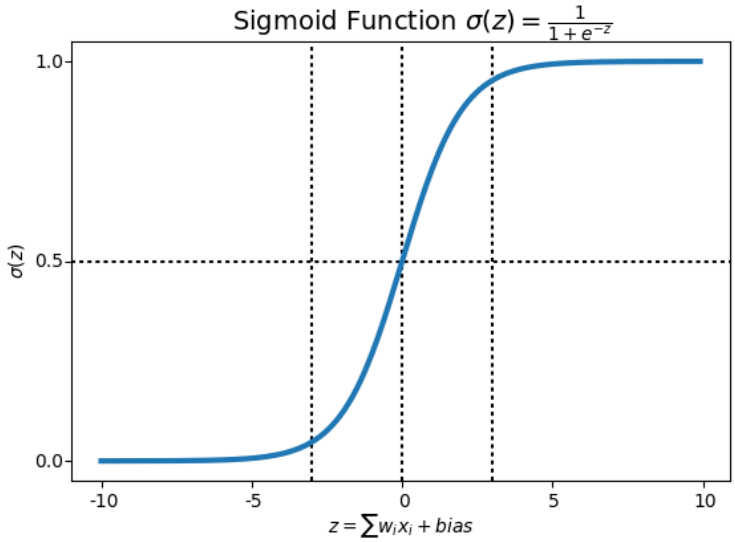

Figure 6: Binary logistics regression function.

Multinomial Logistic Regression model is used for data sets with labels more than 2. When data has multi labels, algorithm first divides the labels to 2 categories, the first category has 1 label and the others are put into the second category. Then the probability of the test sample data point belonging to each category is calculated. The calculation is repeated until the probability of belonging into each label is found. Then the data point is labeled as the class with the highest probability of including the test data point [8]. The function “proba” outputs the array of probabilities for all labels for a given data point. Penalty is a regularization parameter that defines penalization norms. Penalization is used to avoid overfitting the model to the data. The parameter "solver" represents which algorithm will be used to solve the optimization problem. Solver and penalty parameters must be compatible. The solver "saga" is the solver used for large datasets with multilabels [9].

2.2.4 Artificial Neural Networks (ANN)

Artificial Neural Networks is a supervised learning algorithm. Common application areas of ANN include facial recognition, image recognition, natural language processing and forecasting. The algorithm mimicks the mechanism of the human brain using layers with connected nodes that signals and activate one another.

The neural network structures created with "Hidden Layers" between "Input Layer" and the "Ouput Layer" [10]. The first layer that is called the Input Layer is fed with the input data separated for training. All data points are assigned to a node as a variable. The path that connects a node from the input layer to the hidden layer is called a "channel". For each channel there is a weight that is calculated and for every node of the second layer there is a treshold value called "bias". For a node to connect another node from the next layer, input value and weight is multiplied and summed with the bias of the connection node. This function is called the "Activation Function". The algorithm selects the next node to be activated based on the given treshold value of the activation function. According to an example neural network schema below the output is designated using the activation function $y_k = \sigma(a_k)$. The algorithm transforms the sum of weighted inputs and bias using the activation function that is a logistic sigmoid function according to the following formula.

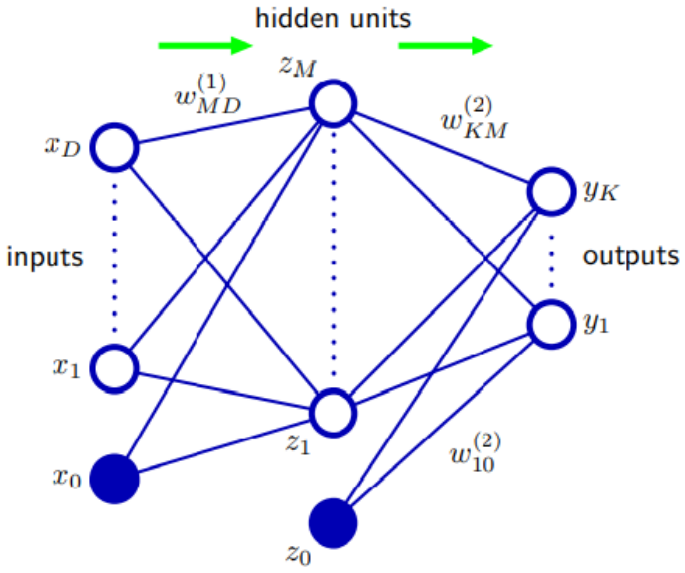

Figure 7: Neural Network model layers.

Note: From *Pattern Recognition and Machine Learning* (228), by C.M. Bishop, 2006, Springer Science+Business Media, LLC

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)} h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right) \quad \textbf{(1)}$$

After connection channel and nodes are determined for each layer the prediction is made from the output layer. The output with the highest probability is selected by the algorithm. The process of the algorithm with the direction from input layer to the output layer is called "Feedforward" or "Forward Propagation". The result of the output layer is then compared with the actual labels. The comparison is made using the Mean Squared Error algorithm also known as the "Cost Function". After the accuracy of the model is determined using the cost function, the system reassignes the weights for each channel to increase accuracy which is the process called "Backpropagation". Selection of the activation function changes according to the problem that needs to be solved and the function highly affects the performance of the model. The rectified linear activation function or "ReLU" is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero [11].

2.2.5 Convolutional Neural Networks (CNN)

Convolutional Neural Network (CNN) is a type of artificial neural network used for pattern recognition on matrix type datasets. The procedure differs from standard neural network algorithm because of the calculations done between the Input and Output layers. Such extra calculations are done by Convolution Layers and Pooling Layers [12].

In a CNN usually the first layer that comes after the input layers is the Convolutional Layer. In this step the algorithm recognizes edges, shapes and specific patterns in the input with the use of a kernel filter. A kernel filter is a small matrix with random values, the size of the filter is given by the user. The algorithm slides the filter matrix across the image and calculates the dot product of the filter and the part of the input matrix the filter is on. The weighted values are all stored on a new matrix. The process of the filter going across all of the input values and calculating the weighted values is called "Convolving". An example of applying the filter to an input matrix is given below.

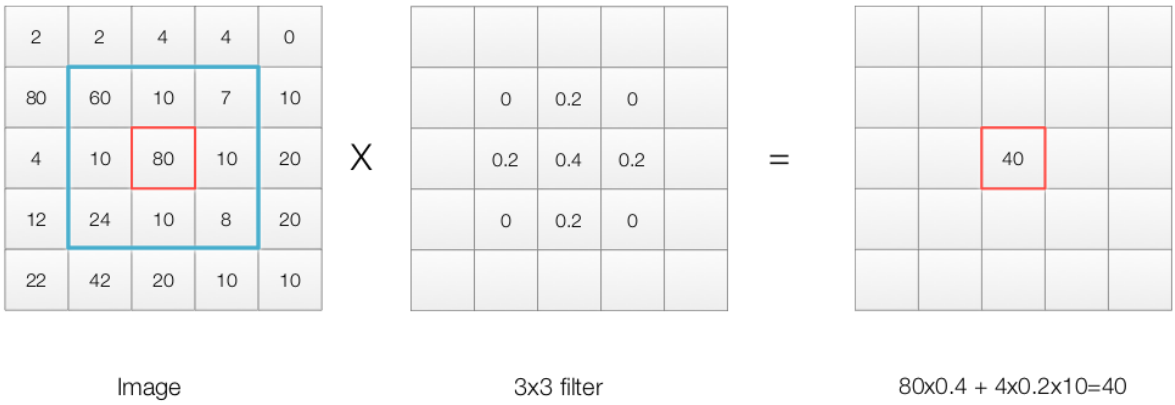

Figure 8: Example of a filter application.

Note: From *From Bits to Brains Machines that can see: Convolutional Neural Networks*, 2016 (https://shafeentejani.github.io/2016-12-20/convolutional-neural-nets/)

There can be a multiple number of filters used on a convolution layer where each filter detects a different type of pattern on the input data. The output of the Convolving process is called a "Feature Map". A result of a typical feature maps with 4 layered CNN is given below.

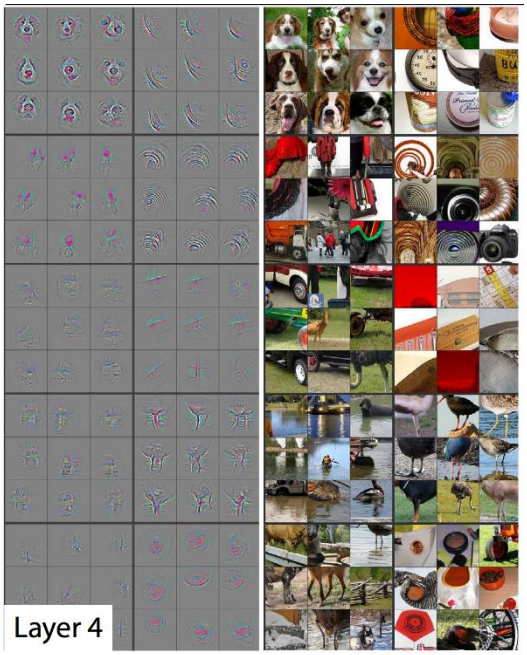

Figure 9: Example feature maps.

Padding describes the addition of empty pixels around the edges of an image. The purpose of padding is to preserve the original size of an image when applying a convolutional filter and enable the filter to perform full convolutions on the edge pixels [13]. “Max pooling” method takes the maximum value in a given pool matrix and derives a new matrix. Flattening is used to convert the 2D array as a result of pooled feature maps into a single continuous linear vector.

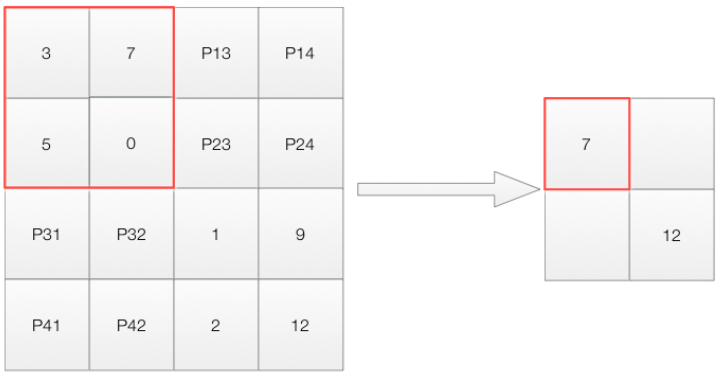

Figure 10: Example of the max pooling method.

In the Fully connected layer, the algorithm classifies the image according to the detected patterns. For example if an image contains two eyes, a nose, sharp teeth, and a body with stripes, it is classified as a picture of a tiger. Convolution and Pooling are linear operations. In

real world applications, the model needs to learn non-linearity [14]. In order to introduce non-linearity to the network, ReLu function that returns 1 for positive values and returns 0 for negative values is used. Fully connected layer is usually applied with softmax activation function for classification. Softmax function produces probabilities from 0 to 1.

3. Experiments

In this section, the contents, shapes and the meaning of the labeling of Indian Pines data has been analyzed. After understanding the data, it is reshaped and resized to better apply the Classical Machine Learning Classification and Neural Network methods that are introduced.

3.1 Data Preparation

Python sklearn, pandas, scipy, numpy libraries must be installed and imported for the following data manipulation applications. The data files retrieved from the source are in Matlab file type. The “loadmat” function of the package loadmat from scipy library is used to load the data files into variables. The multi-dimensional dataset array “pine” has the shape (145,145,220). The (145,145) dimensional part holds the information of x and y coordinates of the point in the Indian pines area. For each of the 220 spectral reflectance bands there is a measured value collected from the AVIRIS sensor. Label data has been loaded as the “pine_gt” array, which has shape (145,145,1). Again the (145, 145)-dimensional part represent the coordinates of each point in the area, however there is only 1 label value for each point. In order to better understand the labels, shapes and borders of the plantations, a figure of the labels are plotted below.

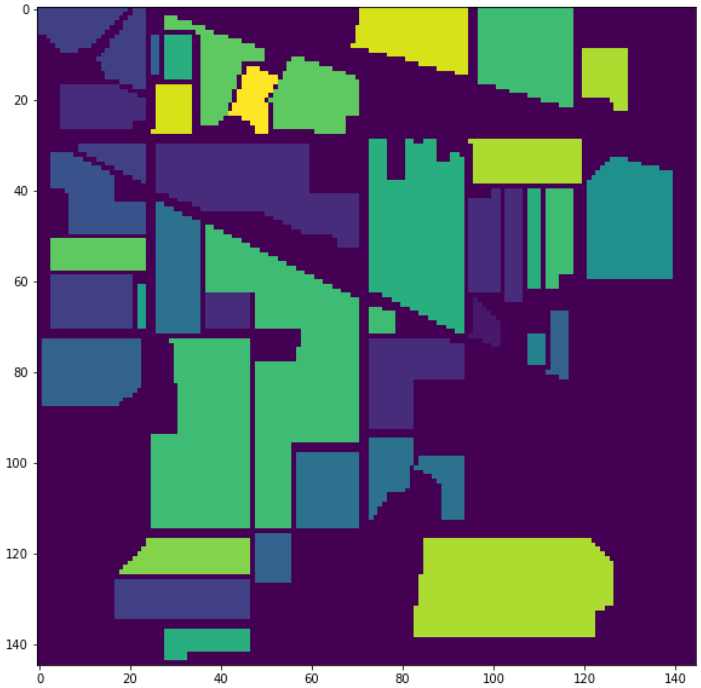

Figure 11: Indian Pines image with labeled regions.

All points from the 144th row of the ground truth array pine_gt are represented with zeros, which can also be observed from the plantation figure. The 0 labeled areas are unclassified. Using the unique function of the Python numpy library, it is possible to see that there are 17 unique labels in the pine_gt data. These labels are named after the implanted plants on these areas according to the source documents. A variable called “plants” is created to hold the original label names. As stated above the AVIRIS sensor captures 220 band measures for each points. However to classify the points, putting all of these band values to the calculation increases complexity. In order to examine how only 3 band values, resized using the PCA method, will effect the accuracy of the experiments, “pine” array must be reshaped to (145*145,220) to correctly apply the PCA method. Also the array with label values is reshaped to (145*145,1) to apply Laber Binarizer operation. Label Binarization creates a column for each possible label and the rows hold the probability of having that specific label. Since we only have 2 possible outcome of having a label for each point, which are 1 and 0, the rows only have the value 1 for their column label and 0 for others. Before decreasing the number of bands with PCA method, the x and y coordinates of each point from the pine array has been calculated and added to a list. The list is transformed to an array with shape (145*145, 222) named “pine_222”. This array will be used to compare how the original 220 banded data with location will perform in comparison with other arrays in the models. Locations are added due to the assumption that each plant is in an area where same type of plants are nearby. The PCA method is applied to pine_220 array with 3 components. Now the “pine_3” array has shape (145*145,3) and 220 bands are desized to 3. The x and y coordinates of each data is added to the pine_3 array and another array with shape of (145*145,5) called “pine_5” is obtained.

3.2 Classical Machine Learning Classification Methods

After importing the StandardScaler package of the scikitlearn library, the band data is standardized between 0 and 1 in order to prevent outliers in the data. In order to apply the classical classification methods that are further explained in the Methodology section, the band data of “pine_3”, “pine_5”, “pine_220”, “pine_222” arrays and label data with one column is splitted into two parts, train and test with 0.25 ratio for the test size. Remaining 0.75 ratio of the data will be used to train the model for the prediction procedure. Binarized label array with 17 columns is not used because binarization process is implemented to the models in the background.

3.2.1 Support Vector Machine (SVM)

The SVM function that returns a dictionary type of output is defined. Standard Scaler application may be applied before splitting the data, however in this case multiple data sets will be used for comparison, therefore writing the process inside the SVM function is more practical. SVM is defined with “rbf” kernel. Since the output is a dictionary object, the label predictions of the model, model accuracy and the confusion matrix that is created after

importing the confusion_matrix package can be called using the “SVM_Pred”, “SVM_Acc” and “SVM_cm” keywords.

3.2.2 K-Nearest Neighbour (KNN)

“KNN” function is defined to create a KNN model. Parameters are the train and test portions of the data and n-parameter. The parameter n indicates the input of the “n_neighbors” parameter of KNN model. In order to determine which n input gives the best accuracy a “knn_error” function is defined, the mean error graph of the four data arrays are calculated for n values between 1 to 40. In order to standardize the accuracy scores of KNN model with an n value, the graphs are examined and “n_neighbors” is determined as 5.

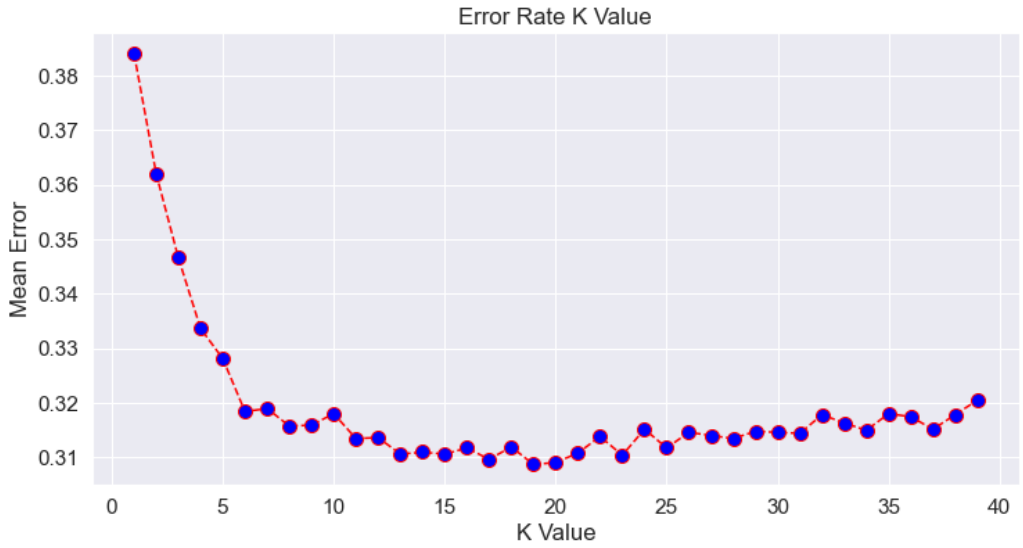

Figure 12: pine_3 error graph of KNN method.

3.2.3 Multinomial Logistic Regression

A function named "LR" that takes the train and test samples as input is defined in order to create multinomial logistic regression model with the four input datasets. Solver is selected as "saga" and penalty is "L1" due to their compability with the "multinomial" selection of the “multi_class” parameter. The parameter “max_iter” defines the maximum number of iterations taken for the solvers to converge. The default number for “max_iter” is 100. However, the input datasets that will be used in the model requires a higher number of iterations. Therefore it is determined as 2500. If the “max_iter” parameter is not changed then a “max_iter exceeded” error will be given. C is an arbitrary parameter in order to regulate model fitting. Model is fitted with train data and predictions are made with “model.predict” function on the test sample. “Proba” function gives the probabilities of all the predictions on each layers. “Proba” results are also added to the dictionary type output of the function “LR” with the model accuracy score and the variable for the confusion matrix.

3.3 Neural Network Methods

The input datasets must be prepared for neural network models. The first step is to use Standard Scaler method on the input datasets. In data preparation, label binarization is made on the label data. For neural network models, binarized labels is used. Then input datasets is split to train and test samples with 0.25 ratio.

3.3.1 Standard Neural Networks

Keras software library that provides a Python interface for artificial neural networks, will be used for neural network models. Keras acts as an interface for the TensorFlow library. TensorFlow library is used to create deep learning algorithms and neural network models with multi-dimensional arrays.

In order to create Standard neural network models with different input dataset, the function “NN” is defined with five parameters. The first four parameters are where the train and test sample are given. The parameter "dimension" is where the user must enter the column numbers, in this case, how many band spectrums that will be included. First linear stack of layers is created and added to the model with keras “Sequential” function. Model layers will be created and added to the Sequential class. In order to add a layer that is densely connected to the NN layers the function “ks.layers.Dense” will be used from keras. For the first layer, in order to introduce non-linearity to the model, “relu” is selected as the activation function. “input_dim” parameter is the dimension parameter that user has entered. There are 17 output labels in the Indians Pines dateset, therefore an output layer with shape 17 is added to the model with “sigmoid” activation function. The sigmoid function is also used in the logistic regression model. In order to compile the model; optimizer, loss and metrics parameters must be selected. Optimizers are the algorithms used to change the attributes of the neural network models, such as weights and learning rate in order to reduce the losses [15]. The “adam” algorithm is used in the “NN” model due to its fast computation time. The loss function in a neural network quantifies the difference between the expected outcome and the outcome produced by the machine learning model [16]. The metrics functions are used to evaluate the performance of the model. Since the labels are binarized in this model, “binary-accuracy” is selected as the metrics function in order to calculate how often predictions match the labels.

In order to train the model with “model.fit” function, batch size and epoch parameters must be given. “batch_size” is the term that defines the number of samples that will be propagated through the network in one iteration. The number of epochs defines the number of times the learning algorithm will work through the entire training dataset.
Training is done with a loop over the number of epochs where each loop proceeds over the training dataset. Within this loop there is another nested loop that iterates over each batch of samples that is specified with the “batch_size” parameter [17].

After the model is trained, predictions can be done with “model.predict” function on the test sample. However the predictions for each data point returns an array with 17 columns that are probabilities of each label. In order for the algorithm to assign the label with highest probability to the test data points, 17 columns is reduced to 1, the label with the highest probability using argmax and axis=1. The confusion matrix and accuracy score of the model along with predictions is added to the dictionary for the output of the “NN” function.

3.3.2 Convolutional Neural Networks

In order to create models with different input datasets with different shapes, the function “CNN” is defined with 6 parameters. Train and test samples of input datasets is required for the first four parameters. m and n parameters are the y and z coordinates of the input data shape. All of the input sets are reshaped to 2D, therefore the value n will be 1 and the m will change for each input according to the column numbers.

A sequential class intance is created to add labels to the model. The first layer is the convolutional layer that is added with Conv1D function which has a 1D filter with 3x1 shape. Padding is selected “same” meaning that the feature map created as a result of the filtering process will have the same shape as the input matrix. The activation function that is selected for the Convolution layer is “relu” since the function adds non-linearity to the model. Pooling is done with “MaxPooling1D” function that is selected with a pool matrix with size. The resultant feature map is flattened with “model.Flatten” function in order to create a 1D long linear vector. Another layer is added to the convolutional neural network with relu activation function to increase accuracy and the fully connected layer is added with the accuracy function “sigmoid” with 17 outputs. Model is compiled with “adam” optimizer using “binary-crossentropy” loss function. “binary-accuracy” metrics is selected to evaluate the performance of the training.

Model is fitted with “model.fit” function with 10 epochs that contains 64 batches. The predictions are made with “model.predict” function on the test sample and binarized predictions are reduced to a single column that contains labels that have maximum probability using argmax. Accuracy scores and confusion matrix is calculated with “accuracy_score” and “confusion_matrix” functions and added to the dictionary output of the “CNN” function.

4. Analysis

Accuracy scores is calculated by dividing the number of correct predictions to the number of total predictions. In all of the functions defined to create models, variables are added to the output of the function to hold value of the accuracy score to be used for analysis. Run time is the time passed in seconds for compiling the model. Run time is a value that is calculated using the time library of Python and "%timeit" function. The run time results will be used to evaluate the performance of the models in the analysis. Variables for the confusion matrices are created with "confusion_matrix" function that summarizes the performance of the models by comparing the number of predictions with original values. Confusion matrices are added to all of the functions that has been defined to create models. Confusion matrices will be plotted using the "CM" function which is defined to create a heatmap of the confusion matrix variable, obtained as a result of the models. Output variables are called using the ".get" function. In this analysis confusion matrices of the SVM, KNN, LR, NN, CNN methods are analyzed based on the input values "pine_5" data; the data points with 3 spectral band values that are retrieved from PCA method with added locations, and "pine_222" data; original data points with 220 spectral band values with added locations. In the confusion matrices, the values on the y axis are the original values and the values on the x axis are the predictions of the model.

4.1 SVM Confusion Matrices

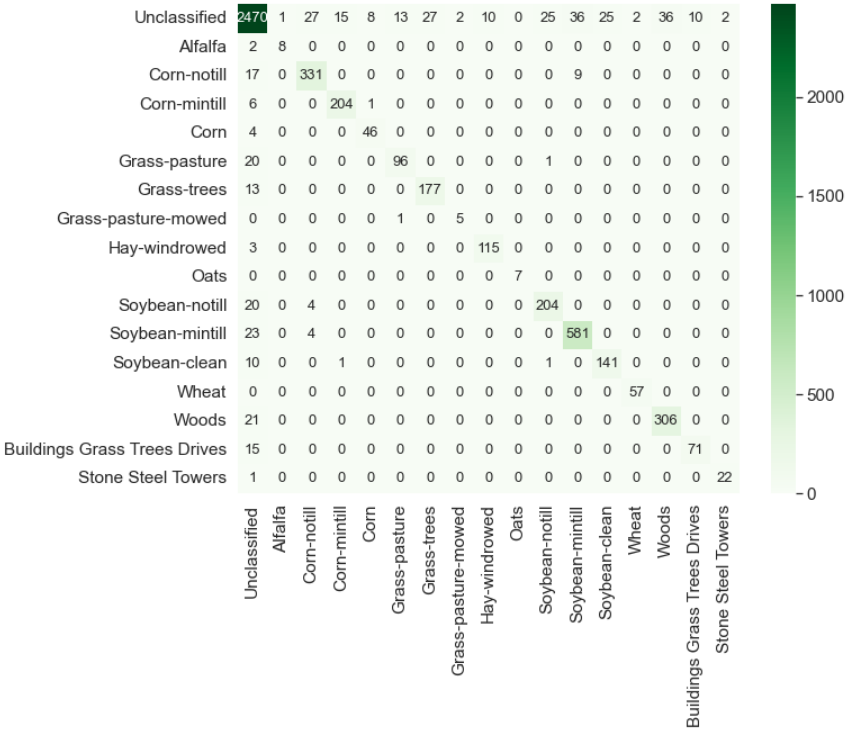

Figure 13: pine_5 confusion matrix of the SVM method.

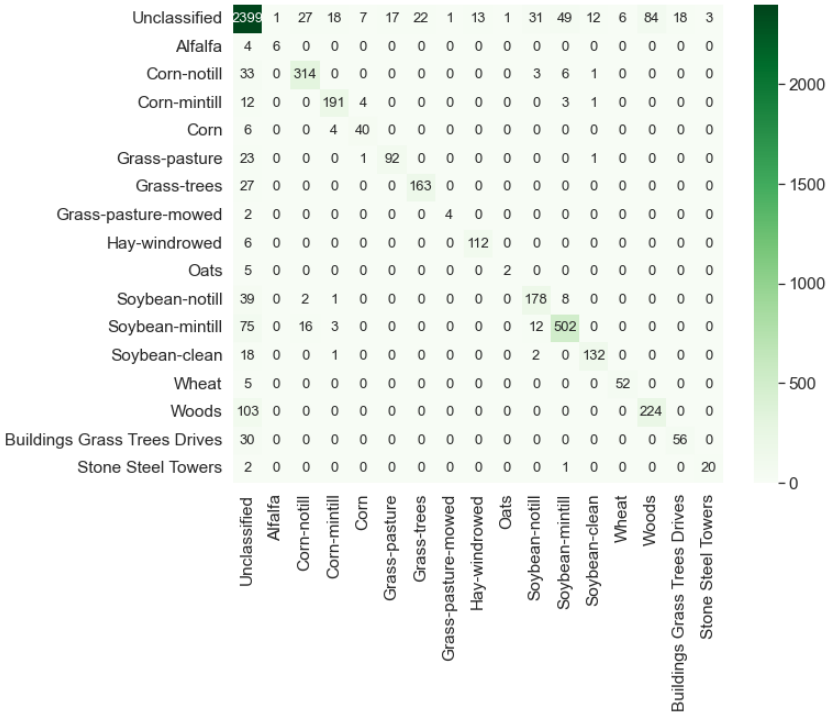

Figure 14: pine_222 confusion matrix of the SVM method.

From the Figure 13, it can be seen that 2470 points are correctly labeled as Unclassified. However the model mislabeled the rest with other labels. Although, none of the plants belong to the Grass-pasture-mowed, Oats, and Wheat classes are mislabeled as Unclassified. The values on the diogonal are the correct predictions. In this model, the plants belong to the Oats are all correctly labeled. All 57 values of the Wheat are correctly labeled, however, 2 unclassified points are mislabeled as Wheat. In Figure 14, it can be seen that almost 32% of the Woods points are mislabeled as Unclassified. In the “pine_222” SVM model, there are no 100% correct predictions of a class. The accuracy scores for Figure 13 and Figure 14 are 0.92 and 0.85. The time it takes to run the model with “pine_222” input is almost 12 times of the model with “pine_5” input. It is clear that the SVM model is more efficient with “pine_5”.

4.2 KNN Confusion Matrices

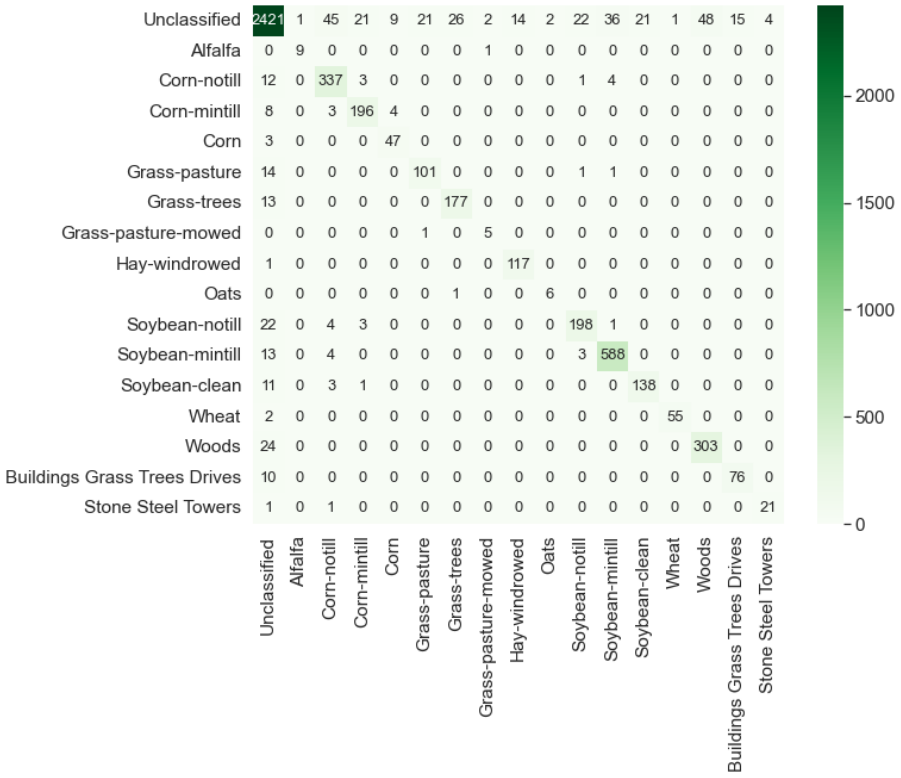

Figure 15: pine_5 confusion matrix of the KNN method.

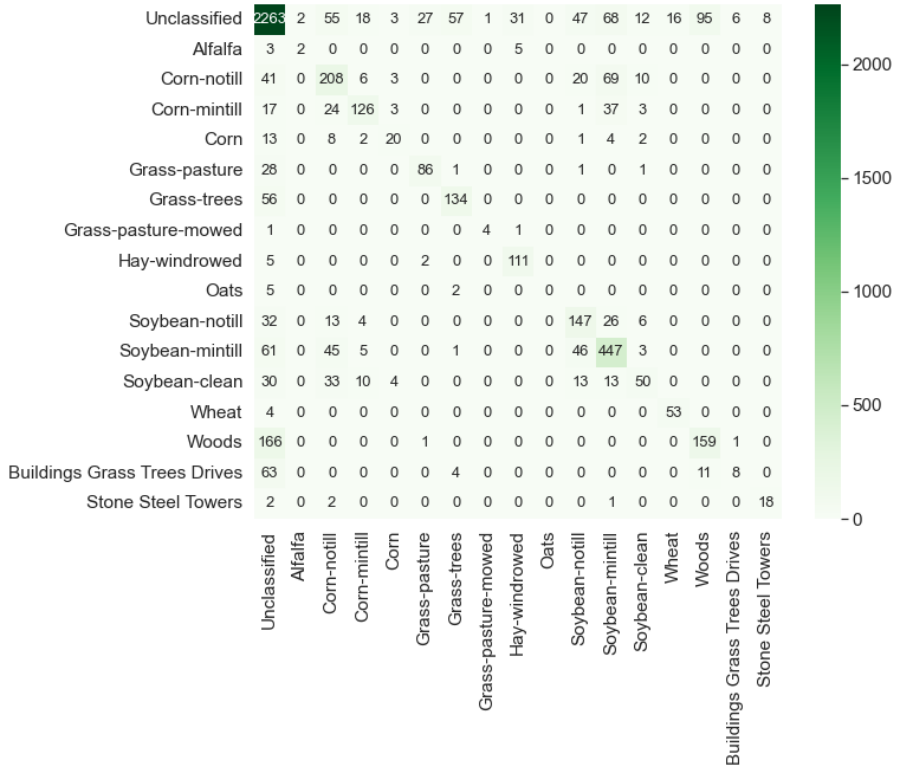

Figure 16: pine_222 confusion matrix of the KNN method.

In Figure 15, KNN method is applied to the data with 5 columns. There is no 100% predictions in any of the classes. In Figure 16, “pine_222” confusion matrix, none of the classes are 100% correctly predicted. Also, all of the Oats plants are mislabeled. Also from the distribution of the predictions it can be said that mislabeling has increased with the “pine_222” data. Accuracy scores of the two models are 0.91 and 0.72. It can be said that there is a major decrease in the accuracy score of the model with “pine_222” input. Even though it takes 12 times more for the “pine_222” KNN model to run than the “pine_5” model, the efficiency is lessened.

4.3 LR Confusion Matrices

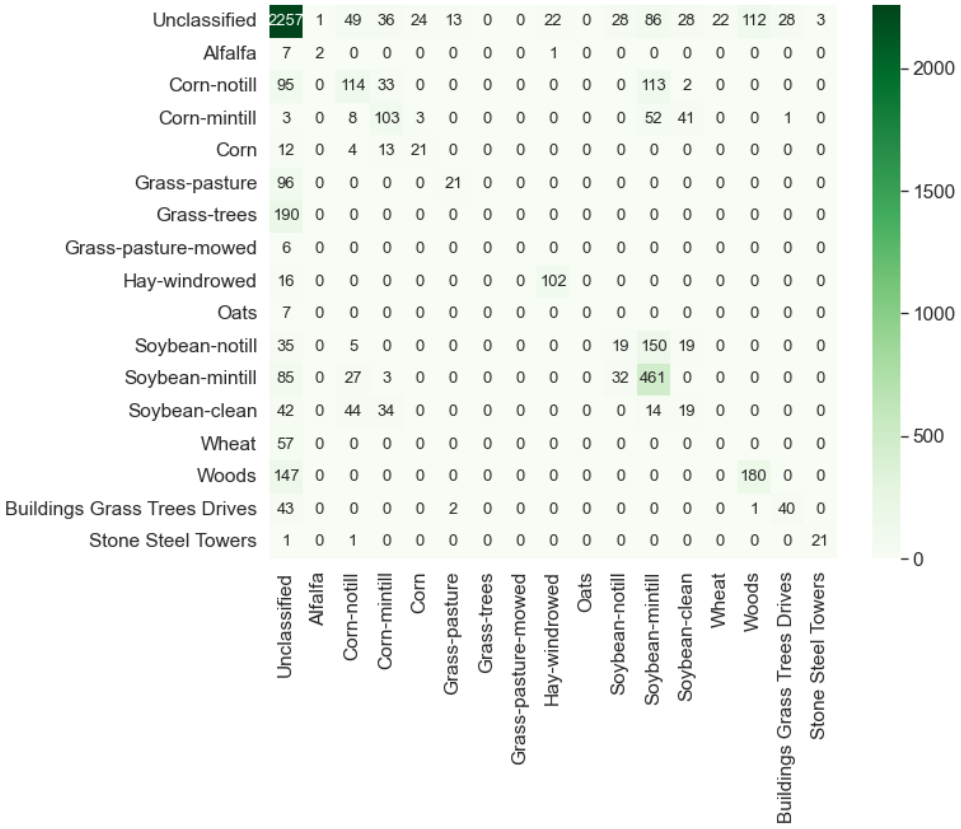

Figure 17: pine_5 confusion matrix of the LR method.

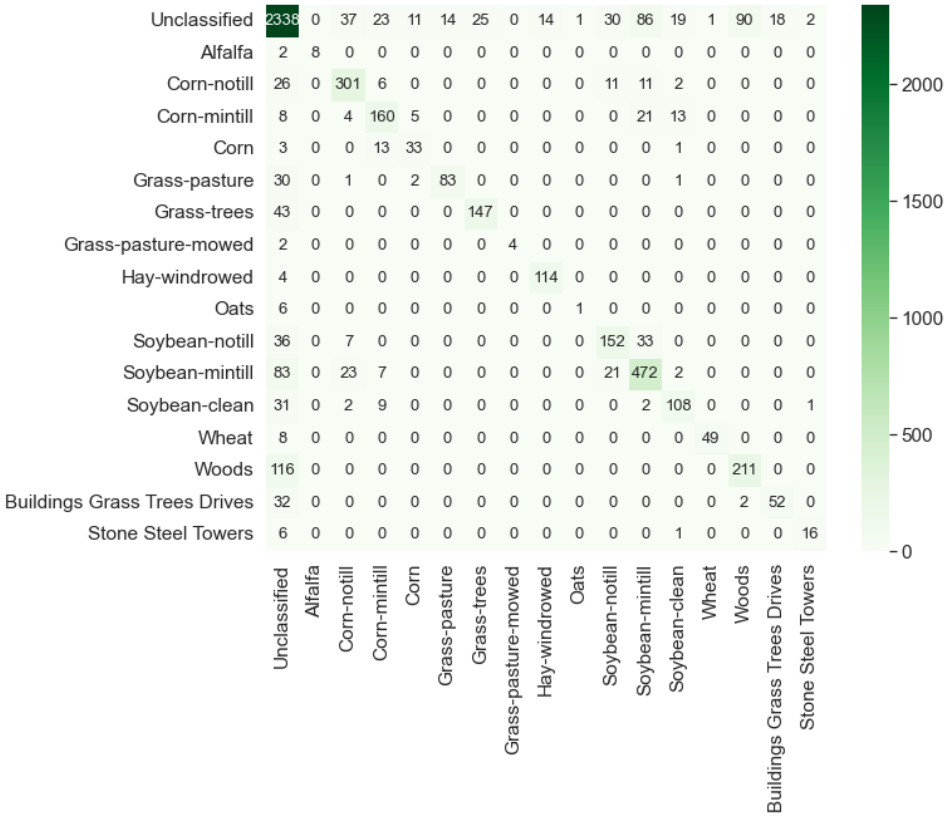

Figure 18: pine_222 confusion matrix of the LR method.

From Figure 17, the Multinomial Logistic Regression method performs very poorly on the "pine_5" data. It is clear that all none of the plants in Oats, Grass-trees, Grass-pasture-mowed, and Wheats classes are labeled correctly. Besides, the model never predicted any of the points as Oats, Grass-trees, and Grass-pasture-mowed. In Figure 18, with "pine_222" input, it can be seen that the model has correctly labeled all of the classes at least once. The accuracy scores are 0.63 and 0.80. LR model with "pine_222" data has taken 90 times more to run than the "pine_5" input. Even though the accuracy has greatly increased, there is a major time difference between the models which decreases the efficiency of the LR model.

4.4 NN Confusion Matrices

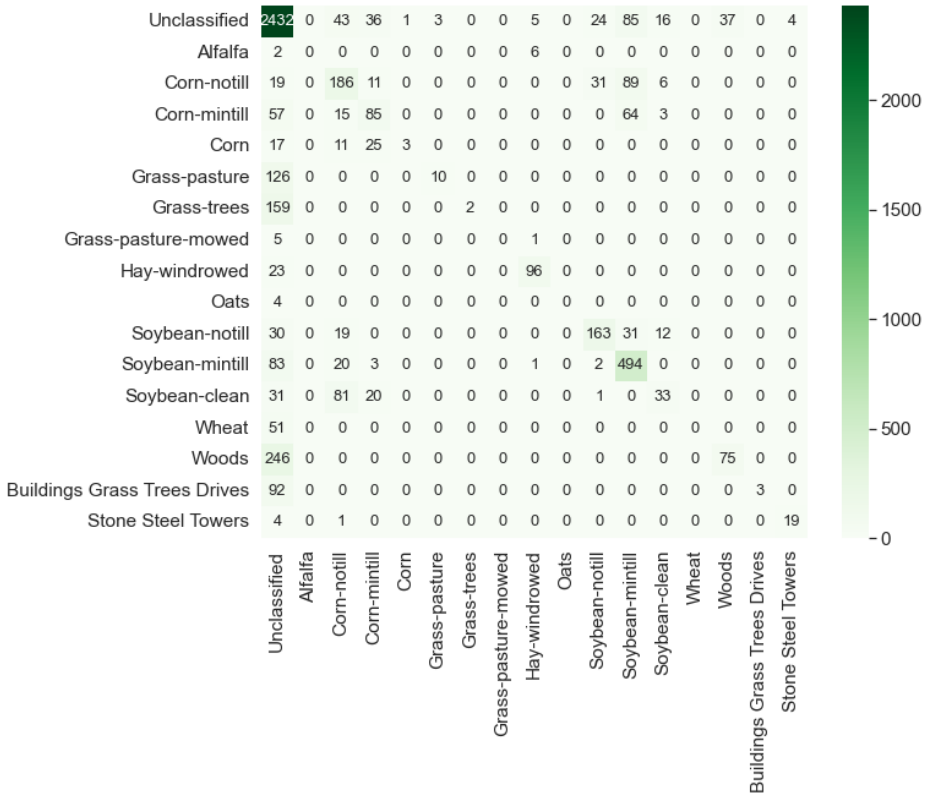

Figure 19: pine_5 confusion matrix of the NN method.

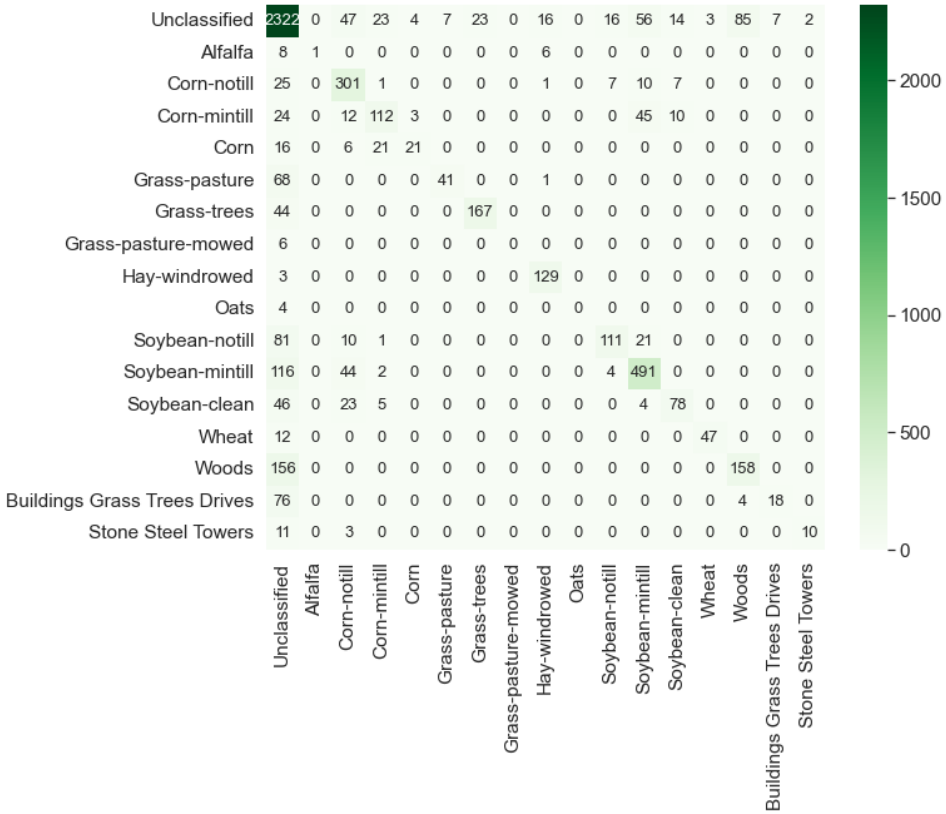

Figure 20: pine_222 confusion matrix of the NN method.

In Figure 19, NN model is applied to the “pine_5” input. It can be seen from the diogonal values that none of the Alfalfa, Grass-pasture-mowed, Oats, and Wheat plants are correctly labeled. Also, the model never labeled any of the points as Grass-pasture-mowed, Oats, and Wheat. In Figure 20, from the confusion matrix of the NN model with “pine_222” data, it can be seen that there are zero correct predictions of the plants in Grass-pasture-mowed and Oats. Besides, in this model, the system never labeled any of the data points as Grass-pasture-mowed and Oats. The accuracy scores are 0.68 and 0.76. Although the run times of the models are almost the same, the NN model has performed better with “pine_222” data.

4.5 CNN Confusion Matrices

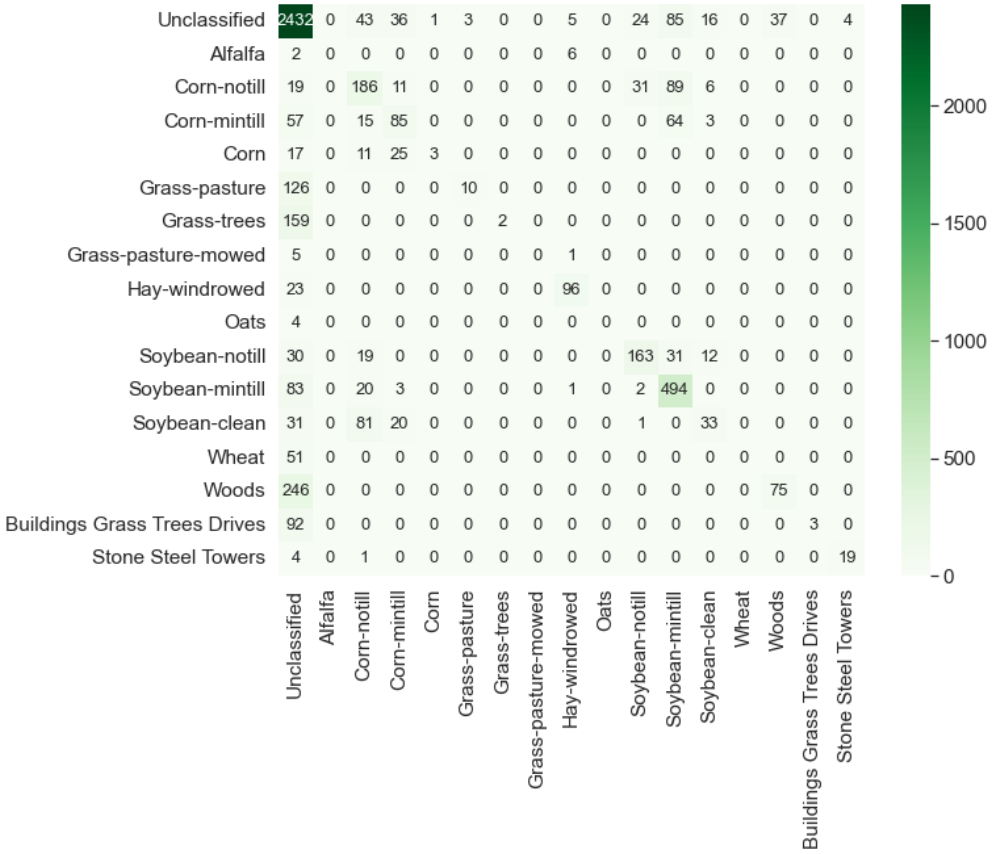

Figure 21: pine_5 confusion matrix of the CNN method.

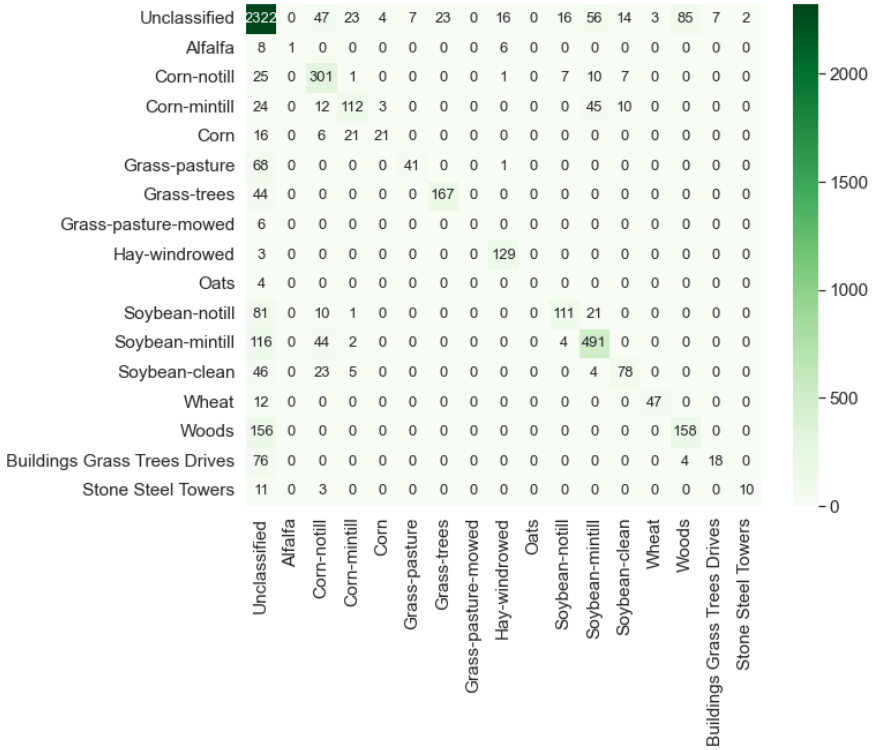

Figure 22: pine_222 confusion matrix of the CNN method.

In the confusion matrix shown as Figure 21, CNN method is applied to the “pine_5” data. The result show that the system never recognized any of the Alfalfa, Oats, Grass-pasture-mowed, and Wheat classes. Almost 77% of the Woods plants and 98% of the Grass-trees plants are mislabeled as Unclassified. From the Figure 22, the confusion matrix of the CNN method with “pine_22” data, it can be seen that the system never recognized any point as Grass-pasture-mowed and Oats. The false mislabeling of the Woods as Unclassified has decreased to 50%. The accuracy scores of the models are 0.79 and 0.83. Although the model with “pine_222” is more accurate, the run time of the system is almost 7 times the run time of the model with “pine_5” input.

4.6 Table of Accuracy and Run-Time Values

	Models	Pine_3	Pine_3 (time)	Pine_5	Pine_5 (time)	Pine_220	Pine_220 (time)	Pine_222	Pine_222 (time)
0	SVM	0.685943	413.878290	0.920867	28.426225	0.800457	389.337662	0.853529	335.896444
1	KNN	0.671866	0.355220	0.912117	0.444140	0.676051	3.865428	0.729694	4.568690
2	LR	0.603386	23.447888	0.639148	18.026950	0.779342	1939.173289	0.808256	1605.705084
3	NN	0.612707	2.674989	0.684991	3.020515	0.698497	2.707859	0.762222	2.740029
4	CNN	0.634582	4.275728	0.794560	5.827198	0.763173	40.982290	0.833365	40.007848

Figure 23: Table of run time and accuracy score values of all models.

From this table, it can be observed that in all of the methods the accuracy results are higher on the models where the location of the pixels are added to the data. The most accurate model

of the methods created for the “pine_5” data is SVM method. However, the run time of the SVM model is 70 times larger than the KNN method which has a very close accuracy. Therefore, it can be said that the most efficient model with “pine_5” data is the KNN method with 5 nearest neighbours. Between the models with 222 columns, the SVM method has the highest accuracy score and the run time of the system is the 4th fastest among all models. The closest accuracy score to the SVM model with “pine_222” is the CNN method where the system runs 8 times faster than the SVM model. The slowest model is the Multinomial Logistic Regression method with large inputs, the models take almost 26 minutes to run, which shows that even though the accuracy score of the model is 0.80, which is the 5th highest score in all of the models, the run time decreases the efficiency greatly. For SVM and KNN methods, data prepared with PCA to reduce the spectral band number to 3, has increased the accuracy. However, with LR, NN, and CNN models, the accuracy increased when original spectral bands were used. In conclusion, for the HSI analysis of the Indian Pines dataset, the most accurate predictions are obtained with SVM method, however the best performing method is KNN with K=5 and for logistic regression and neural network models, PCA method effects the model accuracy negatively.

5. Future Work

The parameters of the models can be readjusted to get better results to update the fitting of the models. For multinomial logistics regression, the reason why the model takes so long to run for large datasets should be investigated. Different layers can be added to the neural networks to increase performance. The filter number can be increased for the Convolutional Layer of the CNN model. The same data preparation and modeling steps can be applied to different HSI datasets to compare the results to better choose the most efficient model for HSI analysis.

6. References

[1] Pan, E., Ma, Y., Fan, F., Mei, X., & Huang, J. (2021, April 26). Hyperspectral image classification across different datasets: A generalization to unseen categories. MDPI. Retrieved from https://www.mdpi.com/2072-4292/13/9/1672/htm

[2] Scheidt, C. (n.d.). *Hyperspectral Remote Sensing Applications*. GFZ. Retrieved from https://www.gfz-potsdam.de/en/section/remote-sensing-and-geoinformatics/topics/hyperspectral-remote-sensing-applications

[3] *Hyperspectral remote sensing scenes*. Hyperspectral Remote Sensing Scenes - Grupo de Inteligencia Computacional (GIC). (n.d.). Retrieved from https://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes

[4] *Principal Component Analysis (PCA) - syllabus.cs.manchester.ac.uk*. (n.d.). Retrieved from http://syllabus.cs.manchester.ac.uk/pgt/2018/COMP61021/lectures/PCA.pdf

[5] IBM Cloud Education. (n.d.). *What is supervised learning?* IBM. Retrieved from https://www.ibm.com/cloud/learn/supervised-learning#:~:text=Supervised%20learning%2C%20also%20known%20as,data%20or%20predict%20outcomes%20accurately.

[6] *Support Vector Machine*. semantic. (n.d.). Retrieved from http://semantic-portal.net/base-machine-learning-in-python-svm

[7] *What is the K-nearest neighbors algorithm?* IBM. (n.d.). Retrieved from https://www.ibm.com/topics/knn

[8] Starkweather, J., & Moske, A. K. (n.d.). *Binary logit regression binary logistic regression analysis-multinomial logistic regression*. StuDocu. Retrieved from https://www.studocu.com/row/document/ambo-university/statistics-for-research/binary-logit-regression-binary-logistic-regression-analysis/8399966

[9] *Sklearn.linear_model.logisticregression*. scikit. (n.d.). Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[10] IBM Cloud Education. (n.d.). *What are neural networks?* IBM. Retrieved from https://www.ibm.com/cloud/learn/neural-networks

[11] Brownlee, J. (2020, August 20). *A gentle introduction to the rectified linear unit (ReLU)*. Machine Learning Mastery. Retrieved from https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/#:~:text=The%20rectified%20linear%20activation%20function,otherwise%2C%20it%20will%20output%20zero

[12] IBM Cloud Education. (n.d.). *What are convolutional neural networks?* IBM. Retrieved June 15, 2022, from https://www.ibm.com/cloud/learn/convolutional-neural-networks

[13] "Convolutional Neural Networks (CNN) tutorial". (n.d.). Retrieved June 15, 2022, from https://jhui.github.io/2017/03/16/CNN-Convolutional-neural-network/

[14] From bits to brains. RSS. (n.d.). Retrieved June 15, 2022, from https://shafeentejani.github.io/2016-12-20/convolutional-neural-nets/

[15] A comprehensive guide on Deep learning optimizers. Analytics Vidhya. (2022, May 24). Retrieved June 15, 2022, from https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/#:~:text=An%20optimizer%20is%20a%20function,loss%20and%20improve%20the%20accuracy.%5D

[16] Seb. (2022, April 4). An introduction to neural network loss functions. Programmathically. Retrieved June 15, 2022, from https://programmathically.com/an-introduction-to-neural-network-loss-functions/#:~:text=The%20loss%20function%20in%20a,all%20losses%20constitutes%20the%20cost.

[17] Brownlee, J. (2019, October 25). Difference between a batch and an epoch in a neural network. Machine Learning Mastery. Retrieved June 15, 2022, from https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch

7. Appendix

Appendix A: Data preparation code
Appendix B-1: Code for splitting the data to apply classical classification methods
Appendix B-2: SVM function code for Support Vector Machine method
Appendix B-3: KNN function code for K Nearest Neighbour method
Appendix B-4: knn_error function code for K Nearest Neighbour method
Appendix B-5: LR function code for Multinomial Logistic Regression method
Appendix C-1: Code for splitting the data to apply neural network methods
Appendix C-2: NN function code for Neural Network method
Appendix C-3: CNN function code for Convolutional Neural Network method
Appendix D-1: CM function code for plotting confusion matrices.
Appendix D-2: Code for obtaining time, accuracy score, confusion matrices of the SVM function.
Appendix D-3: Code for obtaining time, accuracy score, and confusion matrices of the KNN function.
Appendix D-4: Code for obtaining time, accuracy score, and confusion matrices of the LR function.
Appendix D-5: Code for obtaining time, accuracy score, and confusion matrices of the NN function.
Appendix D-6: Code for obtaining time, accuracy score, and confusion matrices of the CNN function.
Appendix D-7: Code for creating the comparison table.

Appendix A

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import keras as ks
import seaborn as sn
import time
from sklearn.svm import SVC
from scipy.io import loadmat
from pandas import read_csv
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report
from sklearn.decomposition import PCA
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from keras.layers import Dense, Conv1D, Dropout, Flatten, MaxPooling1D, LSTM, Embedding

pine = loadmat('Indian_pines.mat')['indian_pines']
pine_gt = loadmat('Indian_pines_gt.mat')['indian_pines_gt']

fig, (ax1) = plt.subplots(1, figsize=(20,10))
img= ax1.imshow(pine_gt)
np.unique(pine_gt)

plants = ['Unclassified', 'Alfalfa','Corn-notill', 'Corn-mintill','Corn','Grass-pasture','Grass-trees','Grass-
pasture-mowed','Hay-windrowed','Oats','Soybean-notill','Soybean-mintill',
'Soybean-clean', 'Wheat','Woods','Buildings Grass Trees Drives','Stone Steel Towers']

pine_220 = pine.reshape((145*145,220))
y_1 = pine_gt.reshape(145*145)

labeler = LabelBinarizer()
y_17 = labeler.fit_transform(y_1)
yy_17=y_17.astype(np.float32)
```

```
Pine_222 =[]
for i in range (145):
    for j in range (145):
        x = pine[i,j].tolist()
        x.extend([i,j])
        # k = pine_gt[i,j]
        # x.extend([i,j,k])
        Pine_222.append(x)
pine_222 = np.asarray(Pine_222).astype(np.float32)
pine_222.shape

model_pca = PCA(n_components=3)
pine_3 = model_pca.fit_transform(pine_220)
pine_3.shape

Pine_3 = pine_3.reshape(145,145,3)

Pine_5 =[]
for i in range (145):
    for j in range (145):
        x = Pine_3[i,j].tolist()
        x.extend([i,j])
        # k = pine_gt[i,j]
        # x.extend([i,j,k])
        Pine_5.append(x)

pine_5 = np.asarray(Pine_5).astype(np.float32)
pine_5.shape
```

Appendix B-1

```
sc = StandardScaler()

pine_3_sc = sc.fit_transform(pine_3)
pine_5_sc = sc.fit_transform(pine_5)
pine_220_sc = sc.fit_transform(pine_220)
pine_222_sc = sc.fit_transform(pine_222)

pine_3_Train,  pine_3_Test, pine_3_Y_Train, pine_3_Y_Test = train_test_split(pine_3_sc, y_1, test_size = 0.25, random_state = 0)
pine_5_Train,  pine_5_Test, pine_5_Y_Train, pine_5_Y_Test = train_test_split(pine_5_sc, y_1, test_size = 0.25, random_state = 0)
pine_220_Train,  pine_220_Test, pine_220_Y_Train, pine_220_Y_Test = train_test_split(pine_220_sc, y_1, test_size = 0.25, random_state = 0)
pine_222_Train,  pine_222_Test, pine_222_Y_Train, pine_222_Y_Test = train_test_split(pine_222_sc, y_1, test_size = 0.25, random_state = 0)
```

Appendix B-2

```
def SVM(x_train,x_test,y_train,y_test):
    svm_model =  SVC(C = 100, kernel = 'rbf', cache_size = 10*1024)
    svm_model.fit(x_train, y_train)
    y_pred = svm_model.predict(x_test)
    svm_acc = accuracy_score(y_test,y_pred)
    cm_svm = confusion_matrix(y_test, y_pred)

    return {'SVM_Acc': svm_acc, 'SVM_Pred': y_pred, 'SVM_cm': cm_svm}
```

Appendix B-3

```
def KNN(x_train,x_test,y_train,y_test,n):
    classifier_knn = KNeighborsClassifier(n_neighbors=n)
    classifier_knn.fit(x_train, y_train)
    knn_pred = classifier_knn.predict(x_test)
    cm_knn = confusion_matrix(y_test, knn_pred)
    knn_acc = accuracy_score(y_test,knn_pred)
    return {'KNN_Acc': knn_acc, 'KNN_Pred': knn_pred, 'KNN_cm': cm_knn}
```

Appendix B-4

```
def knn_error(x_train,x_test,y_train,y_test):
    KNN_error = []

    for i in range(1, 40):
        model_error = KNeighborsClassifier(n_neighbors=i)
        model_error.fit(x_train, y_train)
        pred_i = model_error.predict(x_test)
        KNN_error.append(np.mean(pred_i != y_test))

    plt.figure(figsize=(12, 6))
    plt.plot(range(1, 40), KNN_error, color='red', linestyle='dashed', marker='o',
            markerfacecolor='blue', markersize=10)
    plt.title('Error Rate K Value')
    plt.xlabel('K Value')
    plot_error = plt.ylabel('Mean Error')
    return plot_error

knn_error(pine_3_Train,  pine_3_Test, pine_3_Y_Train, pine_3_Y_Test)
```

Appendix B-5

```
def LR(x_train,x_test,y_train,y_test):
    lr_model = LogisticRegression(C=17,
                                  penalty='l1',
                                  solver='saga',
                                  multi_class='multinomial',
                                  max_iter=2500)
    lr_model.fit(x_train,y_train)
    lr_proba = lr_model.predict_proba(x_test)
    y_pred = lr_model.predict(x_test)
    cm_lr = confusion_matrix(y_test, lr_model.predict(x_test))
    lr_acc = accuracy_score(y_test,y_pred)
    return {'LR_Acc': lr_acc,
            'LR_Pred': y_pred,
            'LR_cm': cm_lr,
            'LR_proba': lr_proba}
```

Appendix C-1

```
sc = StandardScaler()

pine_3_sc = sc.fit_transform(pine_3)
pine_5_sc = sc.fit_transform(pine_5)
pine_220_sc = sc.fit_transform(pine_220)
pine_222_sc = sc.fit_transform(pine_222)

pine_3_train, pine_3_test, pine_3_y_train, pine_3_y_test = train_test_split(pine_3_sc, y_17)
pine_5_train, pine_5_test, pine_5_y_train, pine_5_y_test = train_test_split(pine_5_sc, y_17)
pine_220_train, pine_220_test, pine_220_y_train, pine_220_y_test = train_test_split(pine_220_sc, y_17)
pine_222_train, pine_222_test, pine_222_y_train, pine_222_y_test = train_test_split(pine_222_sc, y_17)
```

Appendix C-2

```
def NN(x_train,x_test,y_train,y_test, dimension):
    nn_model = ks.models.Sequential()
    nn_model.add(ks.layers.Dense(64, activation='relu', input_dim=dimension))
    nn_model.add(ks.layers.Dense(17,activation='sigmoid'))
    nn_model.compile(optimizer='adam',
                     loss='binary_crossentropy',
                     metrics=['binary_accuracy'])
    nn_model.fit(x_train,y_train,epochs=10,batch_size=100)
    y_pred =  nn_model.predict(x_test)
    yy_pred = np.argmax(y_pred,axis=1)
    yy_test = np.argmax(y_test,axis=1)
    nn_acc = accuracy_score(yy_test,yy_pred)
    cm_nn = confusion_matrix(yy_test,yy_pred)
    return {'NN_Acc': nn_acc, 'NN_Pred': y_pred, 'NN_cm': cm_nn}
```

Appendix C-3

```
def CNN(x_train,x_test,y_train,y_test, m,n):
    model_cnn = ks.models.Sequential()
    model_cnn.add(Conv1D(64,
                         3,
                         padding="same",
                         activation='relu',
                         input_shape=(m,n)))
    model_cnn.add(MaxPooling1D(pool_size=2))
    model_cnn.add(Flatten())
    model_cnn.add(Dense(128, activation = 'relu'))
    model_cnn.add(Dense(17, activation = 'sigmoid'))
    model_cnn.compile(optimizer='adam',
                      loss='binary_crossentropy',
                      metrics=['binary_accuracy'])
    model_cnn.fit(x_train, y_train, batch_size=64, epochs=10)
    y_pred = model_cnn.predict(x_test)
    ycc_pred = np.argmax(y_pred,axis=1)
    ycc_test = np.argmax(y_test,axis=1)
    cm_cnn = confusion_matrix(ycc_test,ycc_pred)
    cnn_acc = accuracy_score(ycc_test,ycc_pred)
    return {'CNN_Acc': cnn_acc, 'CNN_cm': cm_cnn}
```

Appendix D-1

```
def CM(cm):
    df_cm = pd.DataFrame(cm[0:17, 0:17],
                         columns= plants[0:17],
                         index= plants[0:17])
    plt.figure(figsize = (10,8))
    sn.set(font_scale=1.4) #for label size
    sn.heatmap(df_cm,
               cmap="Greens",
               annot=True,
               annot_kws={"size": 13},
               fmt='d')
    plot = plt.savefig('cmap.png', dpi=300)
    return plot
```

Appendix D-2

```
SVM_Outputs_3 = SVM(pine_3_Train,  pine_3_Test, pine_3_Y_Train, pine_3_Y_Test)
SVM_3 = SVM_Outputs_3.get("SVM_Acc")
SVM_t3 = %timeit -n1 -r1 -o SVM(pine_3_Train,  pine_3_Test, pine_3_Y_Train, pine_3_Y_Test)
SVM_t3 = SVM_t3.average

SVM_Outputs_5 = SVM(pine_5_Train,  pine_5_Test, pine_5_Y_Train, pine_5_Y_Test)
SVM_5 = SVM_Outputs_5.get("SVM_Acc")
SVM_t5 = %timeit -n1 -r1 -o SVM(pine_5_Train,  pine_5_Test, pine_5_Y_Train, pine_5_Y_Test)
SVM_t5 = SVM_t5.average
SVM_cm5 = CM(SVM_Outputs_5.get("SVM_cm"))

SVM_Outputs_220 = SVM(pine_220_Train,  pine_220_Test, pine_220_Y_Train, pine_220_Y_Test)
SVM_220 = SVM_Outputs_220.get("SVM_Acc")
SVM_t220 = %timeit -n1 -r1 -o SVM(pine_220_Train,  pine_220_Test, pine_220_Y_Train, pine_220_Y_Test)
SVM_t220 = SVM_t220.average

SVM_Outputs_222 = SVM(pine_222_Train,  pine_222_Test, pine_222_Y_Train, pine_222_Y_Test)
SVM_222 = SVM_Outputs_222.get("SVM_Acc")
SVM_t222 = %timeit -n1 -r1 -o SVM(pine_222_Train,  pine_222_Test, pine_222_Y_Train, pine_222_Y_Test)
SVM_t222 = SVM_t222.average
SVM_cm222 = CM(SVM_Outputs_222.get("SVM_cm"))
```

Appendix D-3

```
KNN_Outputs_3 = KNN(pine_3_Train,  pine_3_Test, pine_3_Y_Train, pine_3_Y_Test,5)
KNN_3 = KNN_Outputs_3.get("KNN_Acc")
KNN_t3 = %timeit -n1 -r1 -o KNN(pine_3_Train,  pine_3_Test, pine_3_Y_Train, pine_3_Y_Test,5)
KNN_t3 = KNN_t3.average

KNN_Outputs_5 = KNN(pine_5_Train,  pine_5_Test, pine_5_Y_Train, pine_5_Y_Test,5)
KNN_5 = KNN_Outputs_5.get("KNN_Acc")
KNN_t5 = %timeit -n1 -r1 -o KNN(pine_5_Train,  pine_5_Test, pine_5_Y_Train, pine_5_Y_Test,5)
KNN_t5 = KNN_t5.average
KNN_cm5 = CM(KNN_Outputs_5.get("KNN_cm"))

KNN_Outputs_220 = KNN(pine_220_Train,  pine_220_Test, pine_220_Y_Train, pine_220_Y_Test,5)
KNN_220 = KNN_Outputs_220.get("KNN_Acc")
KNN_t220 = %timeit -n1 -r1 -o KNN(pine_220_Train,  pine_220_Test, pine_220_Y_Train, pine_220_Y_Test,5)
KNN_t220 = KNN_t220.average
KNN_cm220 = CM(KNN_Outputs_220.get("KNN_cm"))

KNN_Outputs_222 = KNN(pine_222_Train,  pine_222_Test, pine_222_Y_Train, pine_222_Y_Test,5)
KNN_t222 = %timeit -n1 -r1 -o KNN(pine_222_Train,  pine_222_Test, pine_222_Y_Train, pine_222_Y_Test,5)
KNN_t222 = KNN_t222.average
KNN_222 = KNN_Outputs_222.get("KNN_Acc")
KNN_cm222 = CM(KNN_Outputs_222.get("KNN_cm"))
```

Appendix D-4

```
LR_Outputs_3 = LR(pine_3_Train,  pine_3_Test, pine_3_Y_Train, pine_3_Y_Test)
LR_3 = LR_Outputs_3.get("LR_Acc")
LR_t3 = %timeit -n1 -r1 -o LR(pine_3_Train,  pine_3_Test, pine_3_Y_Train, pine_3_Y_Test)
LR_t3 = LR_t3.average

LR_Outputs_5 = LR(pine_5_Train,  pine_5_Test, pine_5_Y_Train, pine_5_Y_Test)
LR_5 = LR_Outputs_5.get("LR_Acc")
LR_t5 = %timeit -n1 -r1 -o LR(pine_5_Train,  pine_5_Test, pine_5_Y_Train, pine_5_Y_Test)
LR_t5 = LR_t5.average
LR_cm5 = CM(LR_Outputs_5.get("LR_cm"))

LR_Outputs_220 = LR(pine_220_Train,  pine_220_Test, pine_220_Y_Train, pine_220_Y_Test)
LR_220 = LR_Outputs_220.get("LR_Acc")
LR_t220 = %timeit -n1 -r1 -o LR(pine_220_Train,  pine_220_Test, pine_220_Y_Train, pine_220_Y_Test)
LR_t220 = LR_t220.average

LR_Outputs_222 = LR(pine_222_Train,  pine_222_Test, pine_222_Y_Train, pine_222_Y_Test)
LR_222 = LR_Outputs_222.get("LR_Acc")
LR_t222 = %timeit -n1 -r1 -o LR(pine_222_Train,  pine_222_Test, pine_222_Y_Train, pine_222_Y_Test)
LR_t222 = LR_t222.average
LR_cm222 = CM(LR_Outputs_222.get("LR_cm"))
```

Appendix D-5

```
NN_Outputs_3 = NN(pine_3_train, pine_3_test, pine_3_y_train, pine_3_y_test, 3)
NN_3 = NN_Outputs_3.get("NN_Acc")
NN_t3 = %timeit -n1 -r1 -o NN(pine_3_train, pine_3_test, pine_3_y_train, pine_3_y_test, 3)
NN_t3 = NN_t3.average

NN_Outputs_5 = NN(pine_5_train, pine_5_test, pine_5_y_train, pine_5_y_test, 5)
NN_5 = NN_Outputs_5.get("NN_Acc")
NN_t5 = %timeit -n1 -r1 -o NN(pine_5_train, pine_5_test, pine_5_y_train, pine_5_y_test, 5)
NN_t5 = NN_t5.average
NN_cm5 = CM(NN_Outputs_5.get("NN_cm"))

NN_Outputs_220 = NN(pine_220_train, pine_220_test, pine_220_y_train, pine_220_y_test, 220)
NN_220 = NN_Outputs_220.get("NN_Acc")
NN_t220 = %timeit -n1 -r1 -o NN(pine_220_train, pine_220_test, pine_220_y_train, pine_220_y_test, 220)
NN_t220 = NN_t220.average

NN_Outputs_222 = NN(pine_222_train, pine_222_test, pine_222_y_train, pine_222_y_test, 222)
NN_222 = NN_Outputs_222.get("NN_Acc")
NN_t222 = %timeit -n1 -r1 -o NN(pine_222_train, pine_222_test, pine_222_y_train, pine_222_y_test, 222)
NN_t222= NN_t222.average
NN_cm222 = CM(NN_Outputs_222.get("NN_cm"))
```

Appendix D-6

```
CNN_Outputs_3 = CNN(pine_3_train, pine_3_test, pine_3_y_train, pine_3_y_test, 3,1)
CNN_3 = CNN_Outputs_3.get("CNN_Acc")
CNN_t3 = %timeit -n1 -r1 -o CNN(pine_3_train, pine_3_test, pine_3_y_train, pine_3_y_test, 3,1)
CNN_t3 = CNN_t3.average

CNN_Outputs_5 = CNN(pine_5_train, pine_5_test, pine_5_y_train, pine_5_y_test, 5,1)
CNN_5 = CNN_Outputs_5.get("CNN_Acc")
CNN_t5 = %timeit -n1 -r1 -o CNN(pine_5_train, pine_5_test, pine_5_y_train, pine_5_y_test, 5,1)
CNN_t5 = CNN_t5.average
CNN_cm5 = CM(NN_Outputs_5.get("NN_cm"))

CNN_Outputs_220 = CNN(pine_220_train, pine_220_test, pine_220_y_train, pine_220_y_test, 220,1)
CNN_220 = CNN_Outputs_220.get("CNN_Acc")
CNN_t220 = %timeit -n1 -r1 -o CNN(pine_220_train, pine_220_test, pine_220_y_train, pine_220_y_test, 220,1)
CNN_t220 = CNN_t220.average

CNN_Outputs_222 = CNN(pine_222_train, pine_222_test, pine_222_y_train, pine_222_y_test, 222,1)
CNN_222 = CNN_Outputs_222.get("CNN_Acc")
```

```
CNN_t222 = %timeit -n1 -r1 -o CNN(pine_222_train, pine_222_test, pine_222_y_train, pine_222_y_test, 222,1)
CNN_t222 = CNN_t222.average
CNN_cm222 = CM(NN_Outputs_222.get("NN_cm"))
```

Appendix D-7

```
data = [['SVM', SVM_3, SVM_t3, SVM_5, SVM_t5, SVM_220, SVM_t220, SVM_222, SVM_t222], ['KNN', KNN_3, KNN_t3, KNN_5,KNN_t5, KNN_220 ,KNN_t220, KNN_222, KNN_t222], ['LR', LR_3 , LR_t3, LR_5, LR_t5, LR_220 , LR_t220, LR_222, LR_t222],
      ['NN', NN_3 , NN_t3, NN_5, NN_t5, NN_220 , NN_t220, NN_222, NN_t222], ['CNN', CNN_3, CNN_t3, CNN_5, CNN_t5, CNN_220, CNN_t220, CNN_222, CNN_t222]]

df = pd.DataFrame(data, columns = ['Models', 'Pine_3', 'Pine_3 (time)', 'Pine_5', 'Pine_5 (time)','Pine_220','Pine_220 (time)', 'Pine_222','Pine_222 (time)'])
```