Credit Score Classification-Group 12 (19ELC304 Machine Learning)

DEEPAK SAI P -CB.EN.U4ELC20049

GOWTHAM –CB.EN.U4ELC20076

SAI PRASAD P –CB.EN.U4ELC20064

Summary CA 1

- In CA1, we formulated our problem and made assumptions that would guide the application of various machine learning techniques.
- We applied the KNN classifier algorithm on our problem to develop a classifier that used the features of the existing datasets to classify the data.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- Distance metric used was Minkowski distance.
- The dataset was split into two sets: one for training and other for testing.
- We have Encoded and filled categorical data into numbers (on the whole dataset).
- Using Pipeline() helps us to ensure all of this is this process in one section.
- We evaluated our model using resampling in cross validation.

Summary CA 2

- In CA2, Using PCA we computed the covariance matrix for our dataset before and after applying the technique. Principal component analysis (PCA) is a technique used in machine learning to reduce the dimensionality of a data set while retaining as much information as possible. It does this by identifying the directions in which the data varies the most and projecting the data onto these directions.
- To use PCA, We first standardize the data, as PCA is sensitive to the scale of the features. Then, we compute the eigenvectors and eigenvalues of the covariance matrix of the data, which will give the directions of maximum variance (the principal components) and the amount of variance captured by each component. Finally, we project the data onto the principal components by multiplying the data by the eigenvectors.
- By identifying the directions of maximum variance in the data, PCA can be used to select a small number of features that capture the most information about the data. This can be useful for reducing the complexity of a model or improving its performance.

Neural Network (10 marks)

- A neural network is a machine learning model inspired by the structure and function of the brain. It is composed of layers of interconnected "neurons," which process and transmit information.
- Neural networks are trained using large datasets, and they are able to learn and make intelligent decisions on their own by identifying patterns in the data. They are used for a wide range of applications, including image and speech recognition, natural language processing, and even playing games.
- At a high level, a neural network works by taking in input data, processing it through several hidden layers, and producing an output. Each hidden layer is made up of a set of neurons, which use weights and biases to make decisions based on the input data. The output of a neural network can be a prediction or a classification, depending on the task it is trained for.

Neural Network (10 marks)

- Neural networks are powerful machine learning models, but they can be computationally intensive and require a lot of data to train effectively. They are also prone to overfitting, which means that they may perform well on the training data but poorly on new, unseen data. To address these issues, neural networks are often combined with other techniques, such as regularization and early stopping, to improve their generalization ability.
- A binary classification problem has only two outputs. However, real-world problems are far more complex.
- We Have created a simple neural network from scratch in Python, which can solve multi-class classification problems.

Creating a Neural Network with One Hidden Layer on original dataset

Creating a Neural Network with One Hidden Layer on original dataset

```
# Load dataset
data = df

# Get features and target
X=df.drop("Credit_Score", axis=1)
y=df.Credit_Score

# Get dummy variable
y = pd.get_dummies(y).values

y[:3]

#Split data into train and test data
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=20, random_state=4)

# Initialize variables
learning_rate = 0.5
iterations = 5000
N = y_train.size

# number of input features
input_size = 24
```

- Backpropagation neural network is used to improve the accuracy of neural network and make them capable of self-learning. Backpropagation means “backward propagation of errors”. Here error is spread into the reverse direction in order to achieve better performance.
- Backpropagation is an algorithm for supervised learning of artificial neural networks that uses the gradient descent method to minimize the cost function. It searches for optimal weights that optimize the mean-squared distance between the predicted and actual labels.

```
# number of input features
input_size = 24

# number of hidden layers neurons
hidden_size = 1

# number of neurons at the output layer
output_size = 3

results = pd.DataFrame(columns=["mse", "accuracy"])

# Initialize weights
np.random.seed(10)

# initializing weight for the hidden layer
W1 = np.random.normal(scale=0.5, size=(input_size, hidden_size))

# initializing weight for the output layer
W2 = np.random.normal(scale=0.5, size=(hidden_size , output_size))

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def mean_squared_error(y_pred, y_true):
    return ((y_pred - y_true)**2).sum() / (2*y_pred.size)

def accuracy(y_pred, y_true):
    acc = y_pred.argmax(axis=1) == y_true.argmax(axis=1)
    return acc.mean()
```

BPN learns in an iterative manner. In each iteration, it compares training examples with the actual target label. target label can be a class label or continuous value. The backpropagation algorithm works in the following steps:

```
for itr in range(iterations):

    # feedforward propagation
    # on hidden layer
    Z1 = np.dot(x_train, W1)
    A1 = sigmoid(Z1)

    # on output layer
    Z2 = np.dot(A1, W2)
    A2 = sigmoid(Z2)


    # Calculating error
    mse = mean_squared_error(A2, y_train)
    acc = accuracy(A2, y_train)
    results=results.append({"mse":mse, "accuracy":acc},ignore_index=True )

    # backpropagation
    E1 = A2 - y_train
    dW1 = E1 * A2 * (1 - A2)

    E2 = np.dot(dW1, W2.T)
    dW2 = E2 * A1 * (1 - A1)


    # weight updates
    W2_update = np.dot(A1.T, dW1) / N
    W1_update = np.dot(x_train.T, dW2) / N

    W2 = W2 - learning_rate * W2_update
    W1 = W1 - learning_rate * W1_update
```

Initialize Network: BPN randomly initializes the weights.

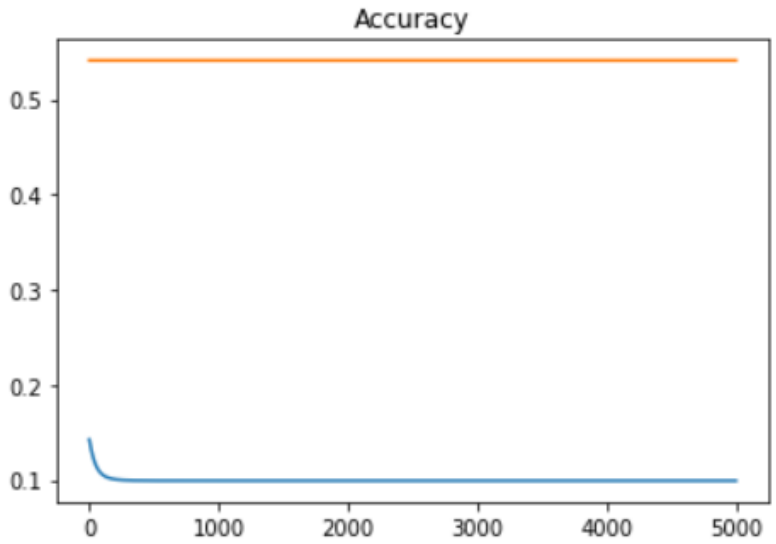
Forward Propagate: After initialization, we will propagate into the forward direction. In this phase, we will compute the output and calculate the error from the target output.

Back Propagate Error: For each observation, weights are modified in order to reduce the error in a technique called the delta rule or gradient descent. It modifies weights in a "backward" direction to all the hidden layers.

Plotting mean squared error in each iteration using pandas plot() function.

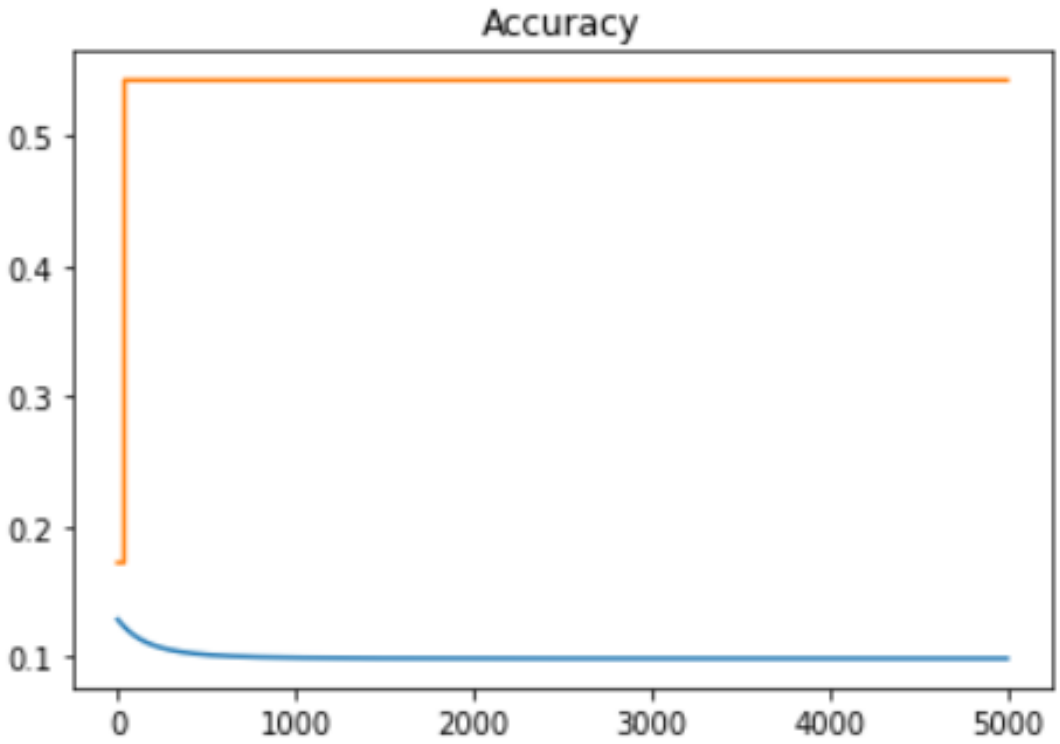
```
results.mse.plot(title="Mean Squared Error")
results.accuracy.plot(title="Accuracy")

# feedforward
Z1 = np.dot(x_test, W1)
A1 = sigmoid(Z1)

Z2 = np.dot(A1, W2)
A2 = sigmoid(Z2)

acc = accuracy(A2, y_test)
print("Accuracy: {}".format(acc))
```

```
Accuracy: 0.6
```



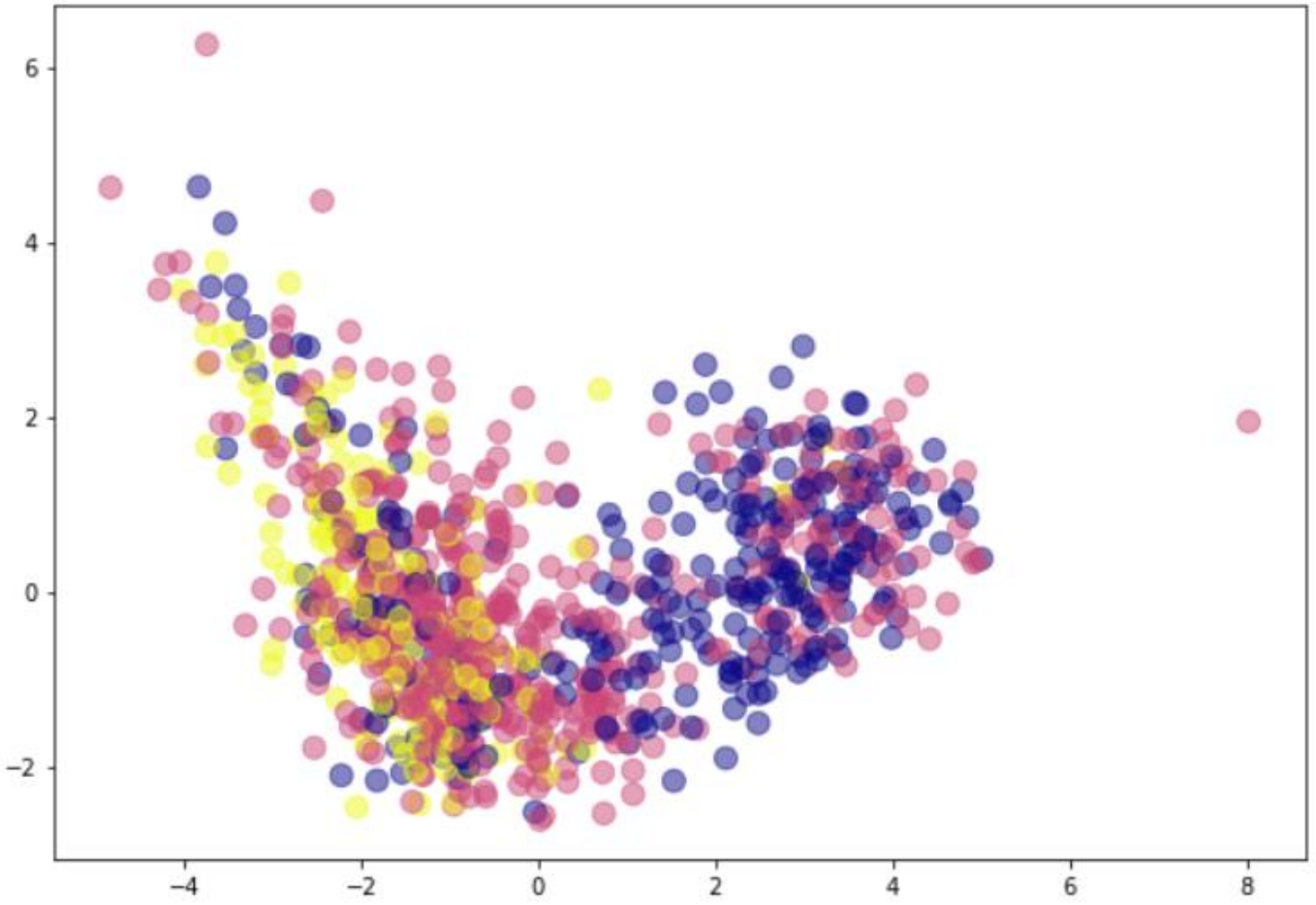
Creating a Neural Network with One Hidden Layer on standardized dataset with PCA = 2

```
Accuracy: 0.6
```



We're getting same accuracy for Original Dataset and standardized dataset with PCA = 2 in Backpropagation Neural Network. We can observe the setting of Accuracy and Mean squared error.

Creating a Neural Network with One Hidden Layer on standardized dataset with PCA = 2 (Type 2)



Creating a Neural Network with One Hidden Layer on standardized dataset with PCA = 2 : Multi-class Classification

```
Loss function value:  1208.238767947495
Loss function value:  853.4486250646517
Loss function value:  847.9783535030655
Loss function value:  846.5710285353618
Loss function value:  846.0597043379537
Loss function value:  845.8312937998378
Loss function value:  845.7075594135596
Loss function value:  845.6236745010199
Loss function value:  845.5525944728148
Loss function value:  845.4808743331962
```

Inference

- Backpropagation Neural Network is a simple and faster model compared to its earlier models. It is also a flexible and standard method. It does not need any prior knowledge for training.
- BPN performance depends upon the kind of input data is used. It is quite sensitive to noisy data. We need to use a matrix-based approach instead of a mini-batch.
- Backpropagation neural network is a method to optimize neural networks by propagating the error or loss into a backward direction. It finds loss for each node and updates its weights accordingly to minimize the loss using gradient descent.
- In this CA3, We have learned What is Backpropagation Neural Network, Backpropagation algorithm working, and Implementation from scratch in python. We plotted mean squared error in each iteration using pandas plot() function. We also got Loss Function value for 2000 epochs and can observe it setting at 845.

Inference

- Two activation functions have been used:

1. Softmax Function: f(x) = e^(xi)/ Σ e^(zk)
2. Binary Sigmoid Function: f(x) = 1/(1+e^(-x))

- **Softmax Function**: From the architecture of our neural network, we can see that we have three nodes in the output layer. We have several options for the activation function at the output layer. One option is to use sigmoid function, However, there is a more convenient activation function in the form of softmax that takes a vector as input and produces another vector of the same length as output. Since our output contains three nodes, we can consider the output from each node as one element of the input vector. The output will be a length of the same vector where the values of all the elements sum to 1. Mathematically, the softmax function can be represented as:

$$y_i(z_i) = \frac{e^{z_i}}{\sum_{k=1}^{k} e^{z_k}}$$

SVM (10 marks)

- Support Vector Machine(SVM) is a supervised machine learning algorithm used for both classification and regression. Though we say regression problems as well its best suited for classification.
- The objective of SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies the data points.
- The dimension of the hyperplane depends upon the number of features. If the number of input features is two, then the hyperplane is just a line. If the number of input features is three, then the hyperplane becomes a 2-D plane. It becomes difficult to imagine when the number of features exceeds three.

SVM (10 marks)

- Advantages of SVM:

1. Effective in high dimensional cases
2. Its memory efficient as it uses a subset of training points in the decision function called support vectors
3. Different kernel functions can be specified for the decision functions and its possible to specify custom kernels

- In this section, we are applying SVM to our dataset and displaying the
- accuracy rates when different kernels are used are additional parameters
- such as C and gamma are specified.
- Gamma indicates as to how far the influence of single training sample
- has reached, where low values mean 'far' and high values mean 'close'. C provides an adjustment for correct classification of training examples
- against maximization of the decision function's margin. Large values
- indicate a small margin and vice versa.

Applying Support Vector Machine (SVM) on standardized dataset with PCA = 2

```
from sklearn.model_selection import train_test_split

# Get features and target
X= x_pca
y=df.Credit_Score

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 20, random_state=4)
```

```
[ ] from sklearn import svm
    from sklearn.metrics import accuracy_score

    model1 = svm.SVC(kernel='linear')
    model2 = svm.SVC(kernel='rbf')
    model3 = svm.SVC(gamma=0.001)
    model4 = svm.SVC(gamma=0.001,C=0.1)
```

```
model1.fit(x_train,y_train)
model2.fit(x_train,y_train)
model3.fit(x_train,y_train)
model4.fit(x_train,y_train)

y_predModel1 = model1.predict(x_test)
y_predModel2 = model2.predict(x_test)
y_predModel3 = model3.predict(x_test)
y_predModel4 = model4.predict(x_test)

print("Accuracy of the Model 1: {0}%".format(accuracy_score(y_test, y_predModel1)*100))
print("Accuracy of the Model 2: {0}%".format(accuracy_score(y_test, y_predModel2)*100))
print("Accuracy of the Model 3: {0}%".format(accuracy_score(y_test, y_predModel3)*100))
print("Accuracy of the Model 4: {0}%".format(accuracy_score(y_test, y_predModel4)*100))
```

```
Accuracy of the Model 1: 80.0%
Accuracy of the Model 2: 80.0%
Accuracy of the Model 3: 75.0%
Accuracy of the Model 4: 60.0%
```

- For C=0.1, we are getting 60% accuracy.

```python
from sklearn import svm
from sklearn.metrics import accuracy_score

model1 = svm.SVC(kernel='linear')
model2 = svm.SVC(kernel='rbf')
model3 = svm.SVC(gamma=0.001)
model4 = svm.SVC(gamma=0.001,C=0.8)

model1.fit(x_train,y_train)
model2.fit(x_train,y_train)
model3.fit(x_train,y_train)
model4.fit(x_train,y_train)

y_predModel1 = model1.predict(x_test)
y_predModel2 = model2.predict(x_test)
y_predModel3 = model3.predict(x_test)
y_predModel4 = model4.predict(x_test)

print("Accuracy of the Model 1: {0}%".format(accuracy_score(y_test, y_predModel1)*100))
print("Accuracy of the Model 2: {0}%".format(accuracy_score(y_test, y_predModel2)*100))
print("Accuracy of the Model 3: {0}%".format(accuracy_score(y_test, y_predModel3)*100))
print("Accuracy of the Model 4: {0}%".format(accuracy_score(y_test, y_predModel4)*100))
```

```
Accuracy of the Model 1: 80.0%
Accuracy of the Model 2: 80.0%
Accuracy of the Model 3: 75.0%
Accuracy of the Model 4: 75.0%
```

From C = 0.8 and greater, we're getting 75% accuracy.

Inference

- C parameter in SVM is Penalty parameter of the error term. You can consider it as the degree of correct classification that the algorithm has to meet or the degree of optimization the the SVM has to meet.
- For greater values of C, there is no way that SVM optimizer can misclassify any single point. Yes, as you said, the tolerance of the SVM optimizer is high for higher values of C . But for Smaller C, SVM optimizer is allowed at least some degree of freedom so as to meet the best hyperplane !
- For C=0.1, we got 60% whereas for C=0.8 and greater, we got 75%.

Inference

- Linear kernel is the best function to be used for the support vector, when we have many features in the dataset.
- RBF kernels are the most generalized form of kernelization and is one of the most widely used kernels due to its similarity to the Gaussian distribution. The RBF kernel function for two points X_1 and X_2 computes the similarity or how close they are to each other.
- We got Linear and RBF kernel's Accuracy as 80%.
- Gamma and C values are key hyperparameters that can be used to train the most optimal SVM model using RBF kernel.
- The gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'
- Higher value of gamma will mean that radius of influence is limited to only support vectors. This would essentially mean that the model tries and overfit. The model accuracy lowers with the increasing value of gamma.
- The lower value of gamma will mean that the data points have very high radius of influence. This would also result in model having lower accuracy.

Miscellaneous

- While working on this project, we learnt about SVMs and Neural Networks, their models and parameters, their applications and also their Pros and Cons.
- Multiclass Classification: A classification task with more than two classes;
- We have used Softmax Function
- Cross-Entropy Function, With softmax activation function at the output layer, mean squared error cost function can be used for optimizing the cost as we did in the previous articles. However, for the softmax function, a more convenient cost function exists which is called cross-entropy.