Coding Assignment 1
Machine Learning
Dr. M. Shamsollahi

Mohammad Hossein Shafizadegan
99104781

December 5, 2023

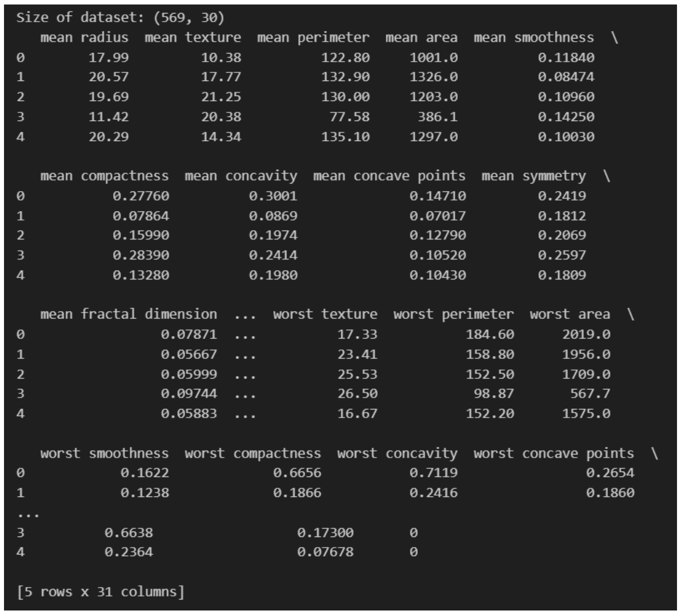
Contents

1 Question 1

1.1 Perceptron

First we simply load the asked dataset and store it in a pandas data frame format. Using the ”head()” built-in function from pandas we can observe the first 5 lines of our dataset.

```python
data = load_breast_cancer()
X = data.data
y = data.target

print(f"Size of dataset: {X.shape}")

# Create a DataFrame
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target # Adding the target column

# Display the first few rows of the dataset
print(df.head())
```



```
Size of dataset: (569, 30)
   mean radius  mean texture  mean perimeter  mean area  mean smoothness  \
0        17.99         10.38          122.80     1001.0          0.11840
1        20.57         17.77          132.90     1326.0          0.08474
2        19.69         21.25          130.00     1203.0          0.10960
3        11.42         20.38           77.58      386.1          0.14250
4        20.29         14.34          135.10     1297.0          0.10030

   mean compactness  mean concavity  mean concave points  mean symmetry  \
0           0.27760          0.3001              0.14710         0.2419
1           0.07864          0.0869              0.07017         0.1812
2           0.15990          0.1974              0.12790         0.2069
3           0.28390          0.2414              0.10520         0.2597
4           0.13280          0.1980              0.10430         0.1809

   mean fractal dimension  ...  worst texture  worst perimeter  worst area  \
0                 0.07871  ...          17.33           184.60      2019.0
1                 0.05667  ...          23.41           158.80      1956.0
2                 0.05999  ...          25.53           152.50      1709.0
3                 0.09744  ...          26.50            98.87       567.7
4                 0.05883  ...          16.67           152.20      1575.0

   worst smoothness  worst compactness  worst concavity  worst concave points  \
0            0.1622             0.6656           0.7119                0.2654
1            0.1238             0.1866           0.2416                0.1860
...
3          0.6638                 0.17300       0
4          0.2364                 0.07678       0

[5 rows x 31 columns]
```

The ”train_test_split()” function from sklearn will simply separate our data for forming the train and test data.

```python
# split the dataset to train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Now we want to implement the perceptron algorithm. First we have developed a function called ”predict_perceptron” which calculates the following:

$$y = \begin{cases} 1 & \text{if } w^T x \geq \theta \\ 0 & \text{O.W.} \end{cases}$$

```python
def predict_perceptron(X, weights, bias):
  return np.where(np.dot(X, weights) - bias >= 0, 1, 0)
```

Then another function called ”train_perceptron”, we first initialize the wights vector and the threshold randomly, then in loop for number of the asked epochs, using the delta rule in batch mode, we interatively update the value of weights and threshold after each epoch finished.

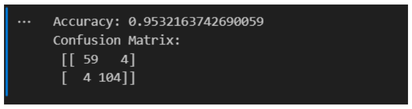
```python
def train_perceptron(X, y, learning_rate=0.01, epochs=100):
  # Initialize weights and bias
  weights = np.zeros(X.shape[1])
  bias = 0

  for epoch in range(epochs):
    # Compute predicted values
    y_pred = predict_perceptron(X, weights, bias)

    # Update weights and bias using batch mode
    weights += learning_rate * np.dot((y - y_pred), X)
    bias -= learning_rate * np.sum(y - y_pred)

  return weights, bias
```

Batch mode is generally used when you have the computational capacity to process the entire dataset at once and when the dataset is not expected to change over time. On the other hand, sample mode is used for large or continuously updating datasets where processing the entire data at once is not feasible. Hence, we decided to use the batch mode of the algorithm here.

Now using the above function, we train our network and calculate the predicted values for our test data set and assess the results by calculating the accuracy and finding the confusion matrix as follows:

```python
# Initialize and train the perceptron
trained_weights, trained_bias = train_perceptron(X_train, y_train, learning_rate=0.01, epochs=1000)

# Predictions on the test set
y_pred = predict_perceptron(X_test, trained_weights, trained_bias)

# Calculate accuracy and confusion matrix
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

# Display results
print("Accuracy:", accuracy)
print("Confusion Matrix:\n", conf_matrix)
```

The results can be seen below:

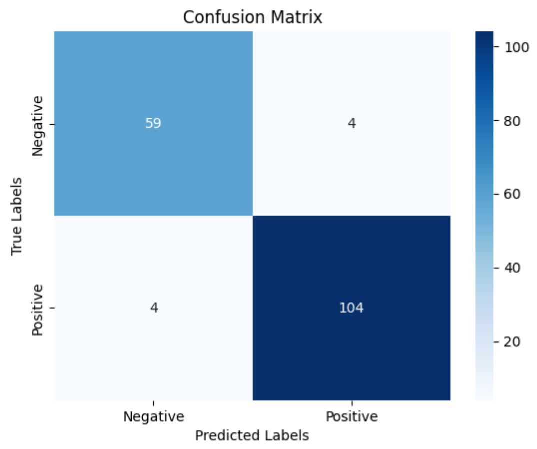


```
...  Accuracy: 0.9532163742690059
     Confusion Matrix:
      [[ 59   4]
      [  4 104]]
```

Now in order to visualize the confusion matrix in a prettier way, we have developed the following function which utilize the ”sns.heatmap” buitl-in function:

```python
def disp_conf_mat(conf_matrix):
  # Visualize the confusion matrix using a heatmap
  sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Negative', 'Positive'],
  yticklabels=['Negative', 'Positive'])
  plt.figure
  plt.xlabel('Predicted Labels')
```

```
    plt.ylabel('True Labels')
    plt.title('Confusion Matrix')
    plt.show()
```



1.2 KNN

In this section, we've provided is an implementation of the k-nearest neighbors (k-NN) algorithm for classification. Here's a brief explanation of how it works:

- predict_single
  calculates the Euclidean distance between a test example and all training examples, identifies the k nearest neighbors, and determines the most common class label among these neighbors. This label is then returned as the prediction for the test example.

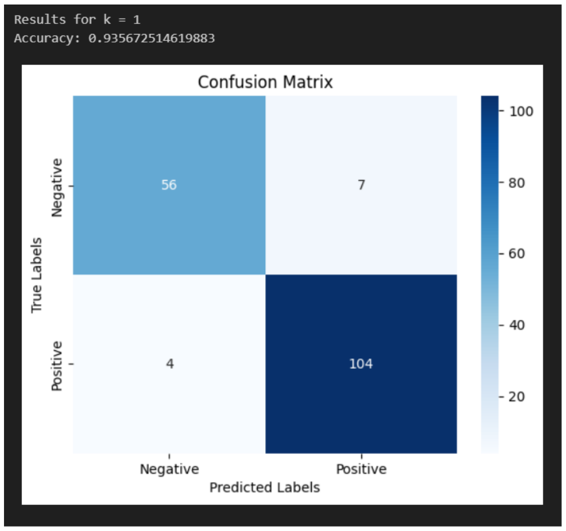
- knn_predict
  takes the training data (X_train, y_train) and test data (X_test) as inputs, along with an optional parameter k which defaults to 3. It predicts the class label for each test example by applying the "predict_single" function to each one.

```
def knn_predict(X_train, y_train, X_test, k=3):
  predictions = [predict_single(x, X_train, y_train, k) for x in X_test]
  return np.array(predictions)

def predict_single(x, X_train, y_train, k):
  # Calculate distances to all points in the training set
  distances = [np.linalg.norm(x - x_train) for x_train in X_train]

  # Get indices of k-nearest training data points
  k_neighbors_indices = np.argsort(distances)[:k]

  # Get the labels of the k-nearest training data points
  k_neighbor_labels = [y_train[i] for i in k_neighbors_indices]

  # Return the most common class label
  most_common = Counter(k_neighbor_labels).most_common(1)
  return most_common[0][0]
```

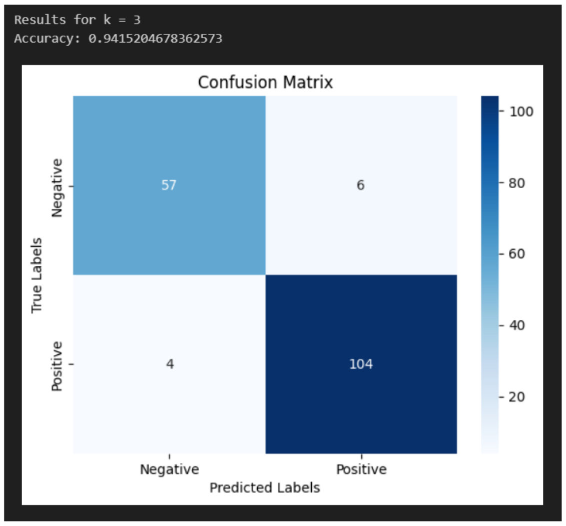
Now for different values of k, we predict the labels and display the accuracy and confusion matrix correspondingly using the following code :

```python
    # Define k values
    k_values = [1, 3, 5, 7]

    for k in k_values:
      y_pred = knn_predict(X_train, y_train, X_test, k)
      accuracy = accuracy_score(y_test, y_pred)
      conf_matrix = confusion_matrix(y_test, y_pred)

      # Display results
      print(f"\nResults for k = {k}")
      print("Accuracy:", accuracy)
      disp_conf_mat(conf_matrix)
```

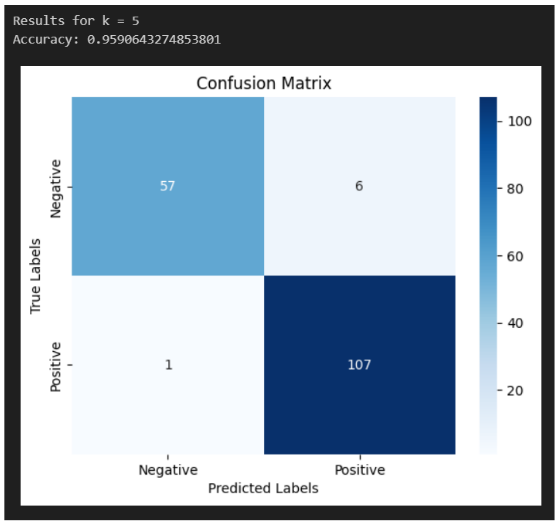
The results can be seen below:



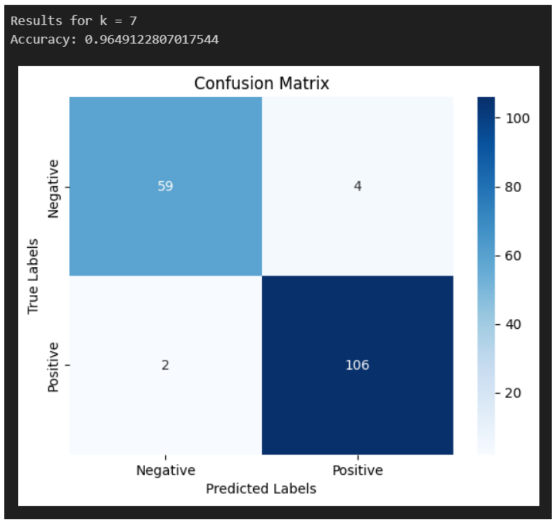
```
Results for k = 1
Accuracy: 0.935672514619883
```



```
Results for k = 3
Accuracy: 0.9415204678362573
```



```
Results for k = 5
Accuracy: 0.9590643274853801
```



```
Results for k = 7
Accuracy: 0.9649122807017544
```

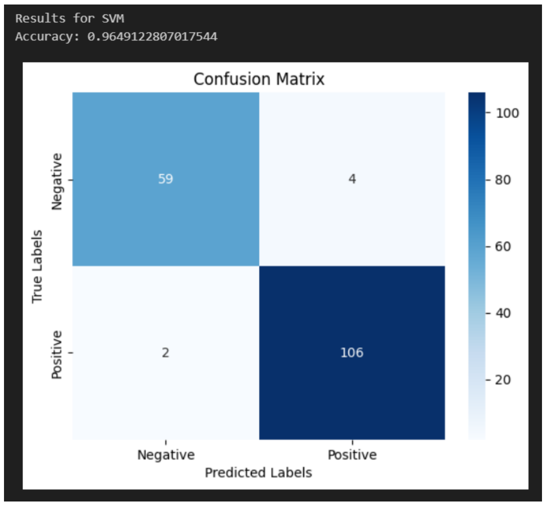
1.3 SVM

In this section we are going to use SVM algorithm for classification. We simply use the built-in functions from the ”sklearn.svm” library. The code of this section is as follows:

```
    # Initialize and train the SVM classifier
    svm_classifier = SVC(kernel='linear')
    svm_classifier.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred_svm = svm_classifier.predict(X_test)
```

Then we calculate the accuracy and display the confusion matrix for this classification.

```
    # Calculate accuracy and confusion matrix for SVM
    accuracy_svm = accuracy_score(y_test, y_pred_svm)
    conf_matrix_svm = confusion_matrix(y_test, y_pred_svm)

    # Display results for SVM
    print("\nResults for SVM")
    print("Accuracy:", accuracy_svm)

    disp_conf_mat(conf_matrix)
```



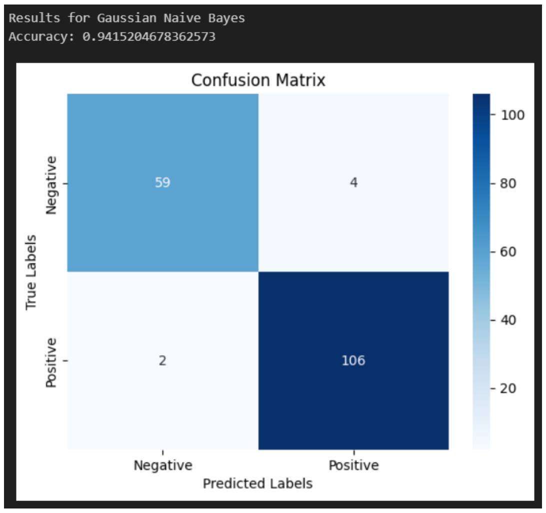
```
Results for SVM
Accuracy: 0.9649122807017544
```

1.4 Bayes Classifier

In this section, we classify our data using Bayes classifier. There are some built-in functions from ”sklearn” to do so.

```
    # Initialize and train the Gaussian Naive Bayes classifier
    nb_classifier = GaussianNB()
    nb_classifier.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred_nb = nb_classifier.predict(X_test)

    # Calculate accuracy and confusion matrix for Naive Bayes
    accuracy_nb = accuracy_score(y_test, y_pred_nb)
    conf_matrix_nb = confusion_matrix(y_test, y_pred_nb)

    # Display results for Naive Bayes
    print("\nResults for Gaussian Naive Bayes")
    print("Accuracy:", accuracy_nb)
    disp_conf_mat(conf_matrix)
```

The results for this approach cab be seen below:



```
Results for Gaussian Naive Bayes
Accuracy: 0.9415204678362573
```

1.5 Comparison

In order to compare the performance of these classifier we can assess the accuracy we calculated before.

classifier	accuracy
perceptron	95.32
knn, k = 1	93.56
knn, k = 3	94.15
knn, k = 5	95.9
knn, k = 7	96.49
SVM	96.5
Bayes	94.15

If we just consider accuracy for measuring the performance, it can be inferred that the SVM and KNN with $k = 7$ have the best performance among the models we compare.

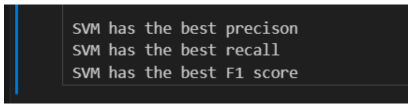
Relying solely on accuracy to assess the performance of models is not always recommended, especially for datasets with imbalanced classes or when the costs of different types of errors vary significantly.

Here we also assess precision, recall and F1 score for our models too. In the following code, we have calculated and displayed the values for precision, recall and F1 score for different methods and store them in corresponding arrays.

```
    from sklearn.metrics import precision_score, recall_score, f1_score

    y_predics = [ y_pred_nb,
                  y_pred_svm,
                  knn_predict(X_train, y_train, X_test, 7),
                  knn_predict(X_train, y_train, X_test, 5),
                     knn_predict(X_train, y_train, X_test, 3),
                     knn_predict(X_train, y_train, X_test, 1),
                     predict_perceptron(X_test, trained_weights, trained_bias)]

    titles = ["Bayes Classifier", "SVM", "KNN, k=7", "KNN, k=5",
              "KNN, k=3", "KNN, k=1", "Perceptron"]

    precisions = np.zeros(len(titles))
    recalls = np.zeros(len(titles))
    f1_scores = np.zeros(len(titles))

    for i, y_pred in enumerate(y_predics):

      precision = precision_score(y_test, y_pred, average='weighted')
      recall = recall_score(y_test, y_pred, average='weighted')
      f1 = f1_score(y_test, y_pred, average='weighted')

      precisions[i] = precision
      recalls[i] = recall
      f1_scores[i] = f1

      print(f"Metrics for {titles[i]}")
      print(f"Precision: {precision}")
      print(f"Recall: {recall}")
      print(f"F1 Score: {f1}")
      print("==========================\n")
```

Finally at the end we have clarified which model has the best performance.

```
    print(f"{titles[np.argmax(precisions)]} has the best precison ")
    print(f"{titles[np.argmax(recalls)]} has the best recall ")
    print(f"{titles[np.argmax(f1_scores)]} has the best F1 score ")
```



```
SVM has the best precison
SVM has the best recall
SVM has the best F1 score
```

2 Question 2

2.1 Parzen Estimation

First we generate the asked samples

```
    np.random.seed(42) # for reproducibility
    X = np.random.normal(0, 1, 20000)
```

First we implement the uniform kernel simply using the following functions

```
    def uniform_kernel(u):

      return np.where(np.abs(u) <= 0.5, 1, 0)
```

If $|u| \leq 0.5$, it returns 1, indicating that the point is within the kernel's window; otherwise, it returns 0.

Then we develop a function called ”parzen_window” which estimates the density at a point x given a set of sample points samples and a bandwidth h.

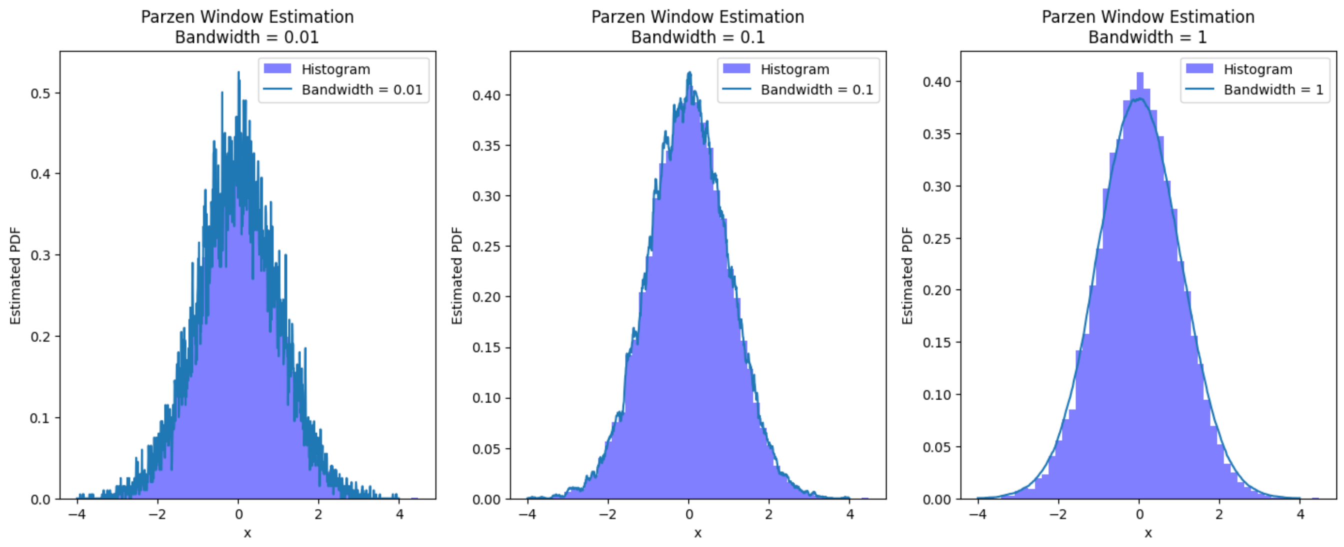
```python
def parzen_window_pdf(x_values, samples, h=1):
  pdf_values = [parzen_window(x, samples, h) for x in x_values]
  return np.array(pdf_values)
```

Finally we have developed a function called ”parzen_window”.

```python
def parzen_window(x, samples, h=1):

  total = 0
  for sample in samples:
    total += 1/h * uniform_kernel((x - sample) / h)
  return total / len(samples)
```

Now for different given values of band width, we estimate the pdf function and visualize it alongside with the data histogram which represents the true pdf function. Here is code used for this task:

```python
# Define bandwidth values
bandwidth_values = [0.01, 0.1, 1]

# Plot the estimated PDF for each bandwidth
plt.figure(figsize=(17, 6))
for i, bandwidth in enumerate(bandwidth_values, 1):
  plt.subplot(1, 3, i)
  plt.hist(X, bins=50, density=True, alpha=0.5, color='blue', label='Histogram')
  x_values = np.linspace(-4, 4, 1000)
  pdf_values = parzen_window(x_values, X, h=bandwidth)
  plt.plot(x_values, pdf_values, label=f'Bandwidth = {bandwidth}')
  plt.xlabel('x')
  plt.ylabel('Estimated PDF')
  plt.title(f'Parzen Window Estimation\nBandwidth = {bandwidth}')
  plt.legend()

plt.show()
```



A small bandwidth (e.g., 0.01) leads to a PDF that is very sensitive to the data points. This can result in an estimate that captures a lot of the data's fine structure but can also be **noisy and overfit** to the sample data, showing peaks and valleys corresponding to individual data points.

A medium bandwidth (e.g., 0.1) provides a balance between capturing the structure of the data and smoothing out noise. The estimated PDF is less sensitive to individual data points than with a smaller bandwidth but still provides a reasonable approximation of the data's distribution.

A large bandwidth (e.g., 1) produces a very smoothed PDF that may overlook important structures in the data. While it reduces the noise and variance in the estimate, it can also lead to underfitting, where the model fails to capture significant patterns in the data.