

Discriminative and Generative Algorithms Performance on Breast Cancer Diagnosis.

María de los Ángeles Contreras Anaya A01700284

Abstract— This document presents the implementation of the K-Nearest Neighbor and the Naive Bayes Classifier algorithm "by-hand" to diagnose Breast Cancer based on characteristics of the cell nucleus of the patients, furthermore the performance of each algorithm is compared against the fed data to determine if the generative or the discriminative algorithm was better.

I. Introduction

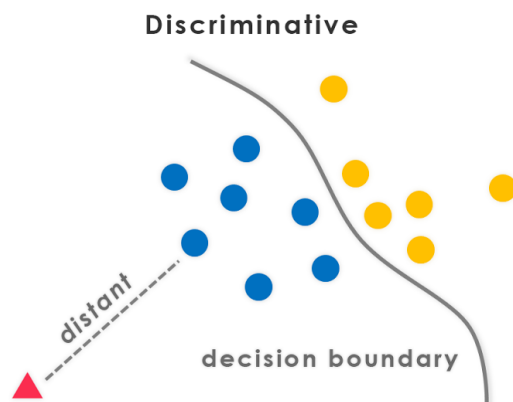
Breast cancer is a disease in which cells in the breast mutate and grow in an uncontrolled way, thus creating a mass of tissue called a tumor. A malignant tumor left unchecked could eventually spread beyond the original location to other parts of the body. According to the World Health Organization, breast cancer is the most common cancer among women and worldwide. In 2020, more than 2.3 million women were diagnosed with breast cancer worldwide and about 685,000 died,

turning this type of cancer into one of the leading causes of death across women of ages 35 to 54. Moreover, young adult females of ages 15 to 39 are less likely to be diagnosed at an early stage of breast cancer (47% of cases in this age group) compared to women older than 65 (68% of cases in this age group). Artificial Intelligence can be of great use in this field to improve the accuracy or to reduce the time that it takes to get breast cancer diagnosed. ML algorithms are not supposed to replace pathologists, but to support them in breast cancer identification, since there are many variations of the disease that make it "unique" to each person who is diagnosed with it.

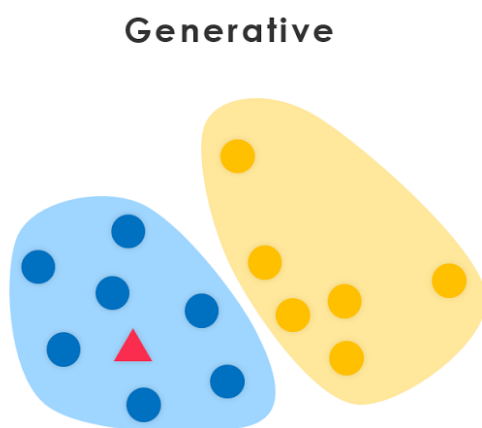
II. State of the Art

Machine learning algorithms can be classified into two types of models, discriminative and generative models. A discriminative model can be used for both classification and regression problems, the model makes predictions on the unseen

data based on conditional probability, thus it focuses on separating one class from another.



On the other hand, a generative model focuses on the distribution of a dataset to return a probability for a given example using probability estimates to model data points and distinguish between class labels in a dataset. These types of models are capable of generating new data points and can tackle more complex tasks.



In sum, discriminative models try to define boundaries in the data space and focuses on distinguishing the label class, while generative

algorithms model the underlying distribution of the data throughout the space and focus on explaining how data was populated. Having said that, discriminative models are said to be more accurate in predictions for classification tasks, which is the kind of task that breast cancer prediction lies in.

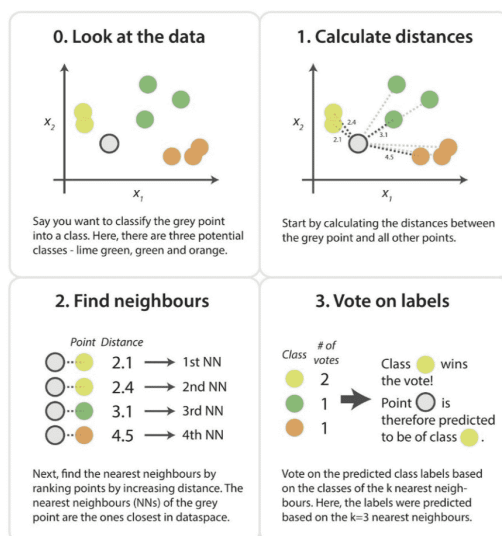
The selected algorithms for this classification task were K-nearest neighbor (discriminative) and Naive Bayes Classifier (generative).

K-Nearest Neighbor

The method for pattern classification known as K-Nearest Neighbor was introduced in an unpublished US Air Force School of Aviation Medicine report in 1951 by Fix and Hodges. This algorithm falls under the supervised learning category, and it can be used either for regression or classification tasks. KNN is known as a lazy learning and non-parametric algorithm because it does not perform any training when data is supplied, it trains once a query is performed, and it does not make any assumptions about the distribution of the supplied data.

The algorithm works by finding the distances between a query and all the labels in the dataset, then it selects the defined number (K) of

labels closest to the query, and finally it predicts or classifies the queried data. For classification tasks, like the one performed in this document, the algorithm uses a voting mechanism for the most frequent label using the mode and for regression tasks it calculates the average of the labels.



K-Nearest Neighbor algorithm uses different types of metrics for finding the distance between the training values and the queried one, for this particular case the euclidean distance was used, which is calculated with the formula:

$$d(x, y) = \sqrt{\sum_{i=1}^n (y_i - x_i)^2}$$

Naive Bayes Classifier

The Naive Bayes Classifier is a probabilistic machine learning algorithm that falls under the supervised learning category. The classifier is based on the Bayes theorem, which is used for calculating conditional probabilities. Conditional probability is a measure that represents the probability of A happening, given that B has occurred, and it is stated mathematically as:

LIKELIHOOD
the probability of "B" being TRUE given that "A" is TRUE

PRIOR
the probability of "A" being TRUE

POSTERIOR
the probability of "A" being TRUE given that "B" is TRUE

The probability of "B" being TRUE

$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$

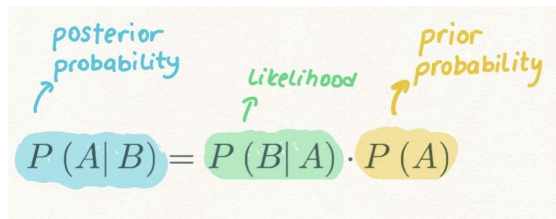
@luminousmen.com

Where:

- **P(B|A) Scaler or Likelihood:** Scales up or down the prior probability based on the observed evidence.
- **P(A) Prior:** The Unscaled probability of an event happening with a randomly chosen set of features.
- **P(B) Normalizer:** Adjusts the calculated probability based on the rarity of the evidence.

In classification tasks, the normalizer is a constant that is omitted for

predictions, so the formula is simplified and ends up being stated mathematically as follows:



The diagram shows the formula $P(A|B) = P(B|A) \cdot P(A)$ with three colored boxes and arrows pointing to them: a blue box for $P(A|B)$ labeled "posterior probability", a green box for $P(B|A)$ labeled "likelihood", and a yellow box for $P(A)$ labeled "prior probability".

Where A is the class variable and B = (b1, b2, b3, ..., bn) represents the features.

The algorithm is called "naive" because it makes the assumption that all features of the given dataset are independent and of equal importance to the outcome. The classifier has limited options for parameter tuning, which allows the model to generalize easily.

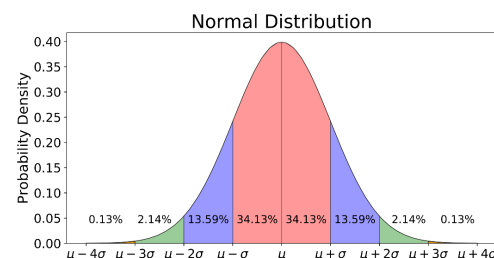
There are five types of Naive Bayes classifiers, the classifiers vary in the nature of the input they receive and on which they perform better:

- Gaussian Naive Bayes: Used when feature values are continuous and the classifier assumes that the aforementioned follow a Gaussian (normal) distribution.
- Multinomial Naive Bayes: Used when feature vectors represent the frequency with which certain target values

have been generated, and it is widely used for document classification tasks.

- Bernoulli Naive Bayes: Used when features are binary variables describing the inputs rather than the frequency of the features, and it is also widely used in document classification.
- Complement Naive Bayes: Used when feature values represent the frequency with which the complement of the target value has been generated.
- Categorical Naive Bayes: Used for data sets that have categorical features instead of frequencies or continuous values.

For the given dataset, **Gaussian Naive Bayes** algorithm was preferred since the feature from the dataset has continuous values.



Using the Gaussian Naive Bayes algorithm, the likelihood of the features is assumed to be:

$$f(x | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

So, basically the algorithm works by calculating the prior probability for all given class labels (benign/malign), then finds the likelihood for each class and each given attribute, then it performs the Bayes Formula to obtain the posterior probability and returns the class with a higher probability.

III. Dataset

The dataset used in this implementation was found on Kaggle, it contains data computed from digitized images of FNA of breast mass describing the characteristics of the present cell nucleus. The set consists of 10 real-valued columns and 20 calculated columns that have the mean and standard error of those 10 real-value features. That leaves us with a set of 33 columns and 569 entries of type float that document the relationship between the cell nucleus characteristics and the mass tissue (tumor) benign or malign. The dataset was downloaded as a .csv

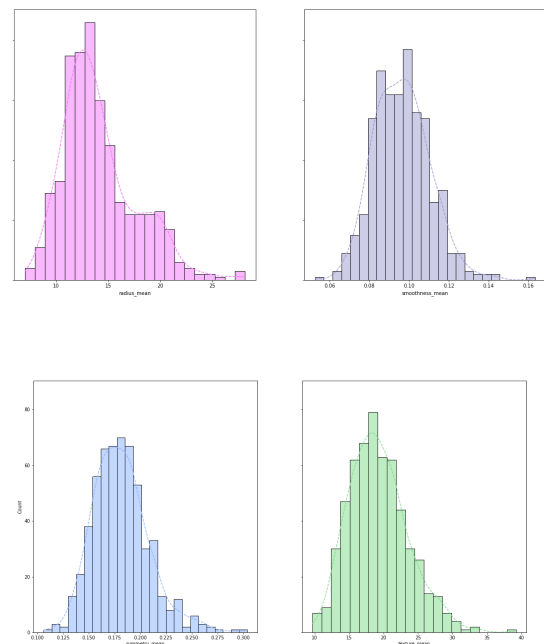
file and read using pandas library on python.

The class distribution of the dataset is of 357 benign tumors and 212 malignant tumors, therefore knowing that our dataset is imbalanced we should focus on our models predicting the minority class (malignant tumors) correctly.

IV. Implementation

• Data Exploration

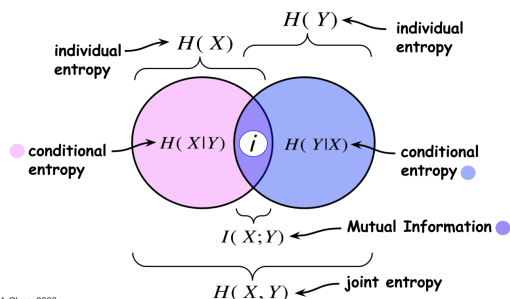
Plotting some features to see the distribution of the data, since the distribution is close to the normal distribution, Gaussian Naive Bayes algorithm is expected to have a good performance.



• Feature Selection

Feature selection is one of the most important steps before feeding data into machine learning algorithms, the right selection of features can increase the predictive power of the algorithm by removing both redundant data (noise) and misleading data.

There are two popular feature selection techniques that can be used for numerical input data and a categorical target variable, but for these implementations, the feature selection technique used was mutual information.



Sean McClure, 2020

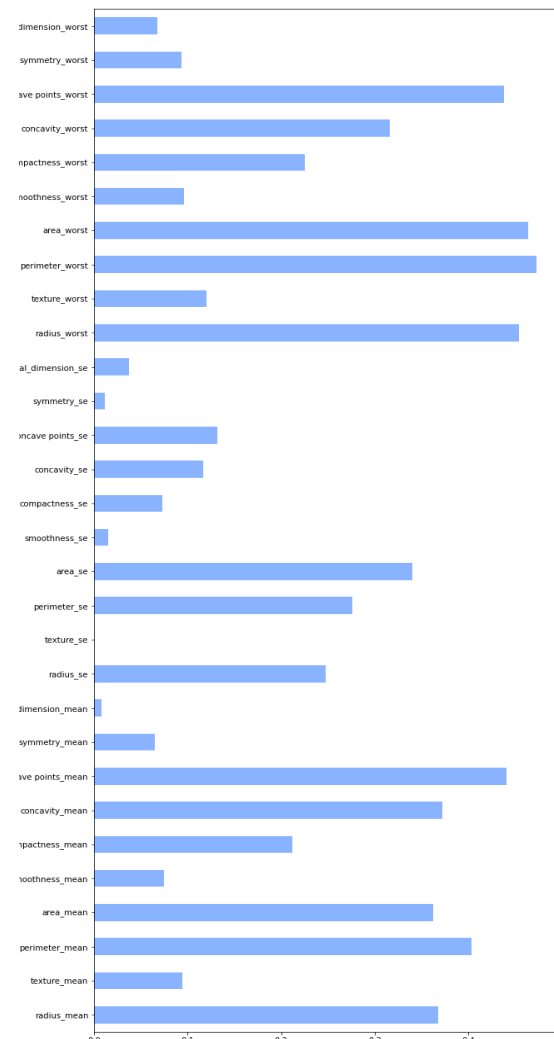
This technique is the application of information gain to feature selection, the metric measures the reduction in uncertainty for one variable given a known value of the other variable, and it is stated by the following formula:

$$I(X; Y) = H(X) - H(X|Y)$$

Where $I(X; Y)$ is the mutual information for X and Y, $H(X)$ is the

entropy for X and $H(X|Y)$ is the conditional entropy for X given Y.

The following plot displays the mutual information metric for the features of the dataset with respect to the "diagnosis" variable. From the graph the selected features were: radius_worst, perimeter_worst and area_worst since they have the biggest values which means they provide more information about the target variable and thus improving the algorithm's performance.



IMG1. graph displayed by the function using mutual information

• Preprocessing

Before performing any prediction, the dataset was preprocessed, in this case the categorical column "diagnosis" was transformed using a scratch function that simulates Label Encoding.

Label Encoding is an encoding technique that converts categorical variables into a numeric form so they can be read and processed. After label encoding was used on the aforementioned column, the "m" values for malignant tumors was replaced by 0 and the "b" values for benign ones by 1.

Furthermore, an empty column and the "id" column were dropped since they do not provide any useful information.

```
RangeIndex: 569 entries, 0 to 568
Data columns (total 33 columns):
#   Column                                Non-Null Count  Dtype
---  ---                                ---
0   id                                    569 non-null    int64
1   diagnosis                             569 non-null    object
2   radius_mean                           569 non-null    float64
3   texture_mean                           569 non-null    float64
4   perimeter_mean                         569 non-null    float64
5   area_mean                             569 non-null    float64
6   smoothness_mean                       569 non-null    float64
7   compactness_mean                      569 non-null    float64
8   concavity_mean                        569 non-null    float64
9   concave points_mean                   569 non-null    float64
10  symmetry_mean                         569 non-null    float64
11  fractal_dimension_mean                569 non-null    float64
12  radius_se                             569 non-null    float64
13  texture_se                             569 non-null    float64
14  perimeter_se                           569 non-null    float64
15  area_se                               569 non-null    float64
16  smoothness_se                         569 non-null    float64
17  compactness_se                        569 non-null    float64
18  concavity_se                          569 non-null    float64
19  concave points_se                     569 non-null    float64
20  symmetry_se                           569 non-null    float64
21  fractal_dimension_se                  569 non-null    float64
22  radius_worst                          569 non-null    float64
23  texture_worst                         569 non-null    float64
24  perimeter_worst                       569 non-null    float64
25  area_worst                           569 non-null    float64
26  smoothness_worst                      569 non-null    float64
27  compactness_worst                     569 non-null    float64
28  concavity_worst                       569 non-null    float64
29  concave points_worst                  569 non-null    float64
30  symmetry_worst                        569 non-null    float64
31  fractal_dimension_worst                569 non-null    float64
32  Unnamed: 32                           0 non-null      float64
```

IMG2. raw data information retrieved with pandas

```
RangeIndex: 569 entries, 0 to 568
Data columns (total 31 columns):
#   Column                                Non-Null Count  Dtype
---  ---                                ---
0   diagnosis                             569 non-null    int64
1   radius_mean                           569 non-null    float64
2   texture_mean                           569 non-null    float64
3   perimeter_mean                         569 non-null    float64
4   area_mean                             569 non-null    float64
5   smoothness_mean                       569 non-null    float64
6   compactness_mean                      569 non-null    float64
7   concavity_mean                        569 non-null    float64
8   concave points_mean                   569 non-null    float64
9   symmetry_mean                         569 non-null    float64
10  fractal_dimension_mean                569 non-null    float64
11  radius_se                             569 non-null    float64
12  texture_se                             569 non-null    float64
13  perimeter_se                           569 non-null    float64
14  area_se                               569 non-null    float64
15  smoothness_se                         569 non-null    float64
16  compactness_se                        569 non-null    float64
17  concavity_se                          569 non-null    float64
18  concave points_se                     569 non-null    float64
19  symmetry_se                           569 non-null    float64
20  fractal_dimension_se                  569 non-null    float64
21  radius_worst                          569 non-null    float64
22  texture_worst                         569 non-null    float64
23  perimeter_worst                       569 non-null    float64
24  area_worst                           569 non-null    float64
25  smoothness_worst                      569 non-null    float64
26  compactness_worst                     569 non-null    float64
27  concavity_worst                       569 non-null    float64
28  concave points_worst                  569 non-null    float64
29  symmetry_worst                        569 non-null    float64
30  fractal_dimension_worst                569 non-null    float64
```

IMG3. data information retrieved with pandas after pre-processing step

• Implementation

To start off both implementations the dataset was divided into training (80%) and testing (20%) data.

For KNN, the split method tried to mimic the one from scikit learn that returns four arrays with the corresponding features and label splits.

```
def split(X, y, train_size):
    shuffle_x = X.sample(frac=1, random_state=2)
    shuffle_y = y.sample(frac=1, random_state=2)

    train_size = int(train_size * len(dataset))

    X_train = shuffle_x[:train_size].values
    X_test = shuffle_x[train_size:].values
    y_train = shuffle_y[:train_size].values
    y_test = shuffle_y[train_size:].values

    return X_train, y_train, X_test, y_test
```

For Naive Bayes the split method simply returns two dataframes

containing the testing and training data.

```
def split(dataset, train_size):
    shuffle_dataset = dataset.sample(frac=1, random_state=2)

    train_size = int(train_size * len(dataset))

    train_set = shuffle_dataset[:train_size]
    test_set = shuffle_dataset[train_size:]

    return train_set, test_set
```

Naive Bayes specific

For Naive Bayes implementation the next step was making the predictions, to make a prediction the first step was to create a method that would calculate the prior probability.

The prior probability is obtained by dividing the total number of events by the number of possible outcomes.

```
def get_prior_probability(train_set, y):
    categories = sorted(train_set[y].unique())
    probability = []
    for i in categories:
        probability.append(len(train_set[train_set[y]==i])/len(train_set))
    return probability
```

The next step is to calculate the likelihood for each variable, using the gaussian distribution formula that involves getting the mean and standard deviation, hence the need for creating a "get_stats" method.

```
def get_stats(train_set, feature):
    mean = train_set[feature].mean()
    std = train_set[feature].std()
    return mean, std

def get_gaussian_distribution(train_set, feature, feature_value, y,
category):
    train_set = train_set[train_set[y]==category]
    mean, std = get_stats(train_set, feature)
    probability = (1 / (np.sqrt(2 * np.pi) * std)) * np.exp(-
((feature_value-mean)**2 / (2 * std**2)))
    return probability
```

Finally, the ***predict*** method was constructed to calculate the posterior probability for each label and returning a list of all the predictions, meaning the label with the highest probability, for each entry.

```
def predict(train_set, X_test, y):
    features = list(train_set.columns)[1:]

    prior_prob = get_prior_probability(train_set, y) # fitting the
model

    # predicting
    y_pred = []
    for x in X_test:
        categories = sorted(train_set[y].unique())
        likelihood = [1]*len(categories)

        for j in range(len(categories)):
            for i in range(len(features)):
                likelihood[j] *= get_gaussian_distribution(train_set,
features[i], x[i], y, categories[j])

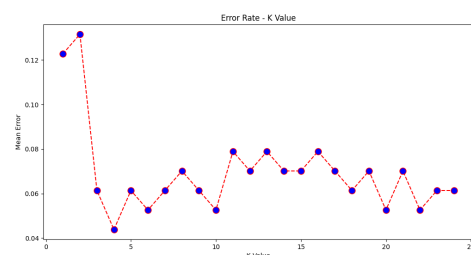
        posterior_prob = [1]*len(categories)

        for j in range(len(categories)):
            posterior_prob[j] = likelihood[j] * prior_prob[j]

        y_pred.append(np.argmax(posterior_prob))
    return y_pred
```

K-Nearest Neighbor specific

For KNN implementation the next step was tuning the **K** parameter in order to improve the algorithms performance on the given dataset. In order to do this an error plot was used, the error plot displays the relationship between the k value and the error rate associated to it. The best value for the KNN algorithm in this case is the one with the minimum error rate.



IMG4. error plot to find optimal k value for KNN algorithm

From the plot, you can see that the smallest error was with a K=4.

To make a prediction using the KNN algorithm the following step was obtaining the nearest neighbors, for this two functions were implemented, one that calculated the distance between two given points using an Euclidean measure and the other that listed, sorted and selected the K nearest neighbors and the predicted values for those.

```
def get_distance(p1,p2):
    return np.sqrt(np.sum((p1-p2)**2))

def get_neighbors(X_train, test_row, k):
    neighbors = []

    for i in range(len(X_train)):
        distances = get_distance(np.array(X_train[i,:]), test_row)
        neighbors.append(distances)
    neighbors = np.array(neighbors)

    k_nearest_neighbors = np.argsort(neighbors)[:k]
    return k_nearest_neighbors
```

Finally, the ***predict*** method was constructed, this method made use of the ones described above and then perform the votation to get the prediction for the given input using the mode. A "get_mode" function was created since the one from the *statistics* library did not support having multiple values as mode and the multimode variation returned a list instead of a single value.

```
# returns the smallest number from the found mode values
def get_mode(values):
    modes = []
    counts = {k:values.count(k) for k in set(values)}
    for key, val in counts.items():
        if val == max(counts.values()):
            modes.append(key)
    modes.sort()
    return modes[0]

def predict(X_train, y_train, X_test, k):
    classification = []

    for elem in X_test:
        k_nearest_neighbors = get_neighbors(X_train, elem, k)

        labels = y_train[k_nearest_neighbors]
        classification.append(get_mode(labels.tolist()))

    return classification
```

● Evaluating models

To evaluate the predictions of both models a confusion matrix, which is a technique used to summarize the performance of a classification algorithm, was chosen since it provides insight into how and why is the classification model is getting "confused".

Furthermore, the following metrics were used to compare the algorithms performance:

- Accuracy (The ratio of correct predictions to total predictions made)
- Recall (The measure of our model correctly identifying True Positives)
- Precision (Measure of the quality of a positive prediction made by the model)

All this evaluating tools were made from scratch trying to mimic the ones from scikit learn.

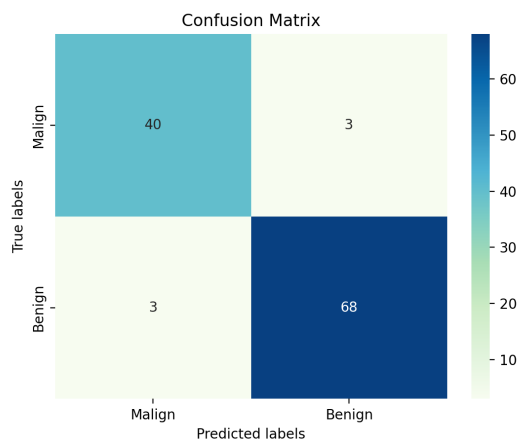
V. Tests

Scenario A.

For this case scenario, the feature selection step was skipped on both algorithms, that means that the predictions are made taking into account all columns from the dataset except for the label one as features (30 features).

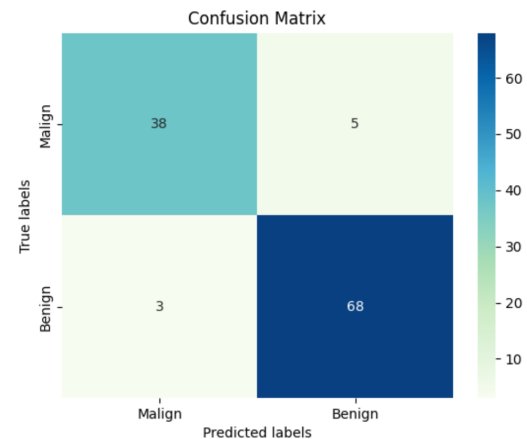
K-Nearest Neighbor

For K we used the optimal value obtained from the error plot which is 4.



Metrics for the K-nearest neighbor algorithm with K = 4		
Accuracy	Precision	Recall
94.74 %	93.02 %	93.02 %

Gaussian Naive Bayes



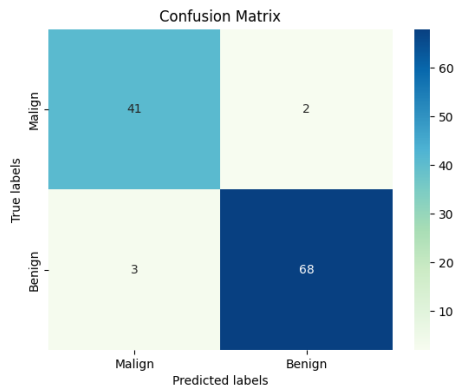
Metrics for the Naive Bayes algorithm with the Gaussian Distribution		
Accuracy	Precision	Recall
92.98 %	92.68 %	88.37 %

Scenario B.

In this case scenario, feature selection was performed using the mutual information technique, the 3 best variables (the ones with the highest information gain value) were selected: concave points_worst, perimeter_worst, concave points_mean and used in both models.

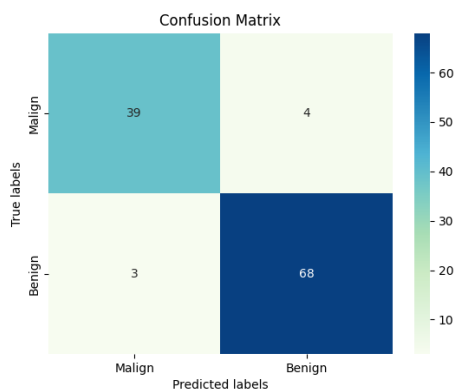
K-Nearest Neighbor

For K we used the optimal value obtained from the error plot which is 4.



Metrics for the K-nearest neighbor algorithm with K = 4		
Accuracy	Precision	Recall
95.61 %	93.18 %	95.35 %

Gaussian Naive Bayes



Metrics for the Naive Bayes algorithm with the Gaussian Distribution		
Accuracy	Precision	Recall
92.98 %	92.68 %	88.37 %

VI. Conclusions

From the implementation of the two algorithms in order to tackle the task of predicting breast cancer by classifying tumors from patients as malignant or benign and the conducted tests, we learned the following:

1. **Feature selection improved the algorithm's performance almost to the same percentage in both the generative and the discriminative algorithm** from using all dataset columns as features (30) to using the best features (3) recall percentage improved by over 2% in both KNN and Naive Bayes, namely both algorithms managed to predict correctly one false positive prediction.
2. **Discriminative models are in fact more accurate in classification tasks**, which is the kind of task that breast cancer prediction lies into. Using the best features obtained with the mutual information technique, KNN algorithm had an accuracy of 95.61% and Naive Bayes an accuracy of 93.86% that means that KNN is over 1.5% more accurate than the latter.
3. **K-Nearest Neighbor algorithm is better suited to predict breast cancer than Gaussian Naive Bayes.** This assumption is made since the recall metric of the KNN model is greater than the one from Naive Bayes. Recall is chosen as the preferred metric since we are dealing with the prediction of

a medical disease, thus it is far more important to diagnose all positive values than all the negative ones. Furthermore, the dataset is slightly imbalanced and the positive class is the minority class, hence predicting the positive values gains even more importance. KNN algorithm had a recall of 95.35% while Naive Bayes had a recall of 90.70%, meaning that the KNN algorithm identifies a larger number of malignant tumors than Naive Bayes.

VII. References

- World. (2021, March 26). *Breast cancer*. Who.int; World Health Organization: WHO.
<https://www.who.int/news-room/fact-sheets/detail/breast-cancer>
- Breast Cancer Facts and Statistics*. (2022). Breastcancer.org.
<https://www.breastcancer.org/facts-statistics>
- Fumo, D. (2017, June 15). *Types of Machine Learning Algorithms You Should Know*. Medium; Towards Data Science.
[https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861#:~:text=Supervised%20learning%20algorithm](https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861#:~:text=Supervised%20learning%20algorithm%20try%20to,from%20the%20previous%20data%20sets.)
- Raj, A. (2021, April 8). *Introduction to Classification Using K Nearest Neighbours*. Medium; Towards Data Science.
<https://towardsdatascience.com/getting-acquainted-with-k-nearest-neighbors-ba0a9ecf354f>
- Gandhi, R. (2018, May 5). *Naive Bayes Classifier - Towards Data Science*. Medium; Towards Data Science.
<https://towardsdatascience.com/naive-bayes-classifier-81d512f50a7c>
- Naive Bayes Classifiers - GeeksforGeeks*. (2017, March 3). GeeksforGeeks.
<https://www.geeksforgeeks.org/naive-bayes-classifiers/>
- UCI Machine Learning. (2016). *Breast Cancer Wisconsin (Diagnostic) Data Set*. Kaggle.com.
<https://www.kaggle.com/datasets/uci/ml/breast-cancer-wisconsin-data>
- McClure, S. (2020, November 7). *A Deep Conceptual Guide to Mutual Information - The Startup - Medium*. Medium; The Startup.
<https://medium.com/swlh/a-deep-conceptual-guide-to-mutual-information-a5021031fad0>
- Yadav, D. (2019, December 6). *Categorical encoding using Label-Encoding and One-Hot-Encoder*.

Medium; Towards Data Science.

<https://towardsdatascience.com/categorical-encoding-using-label-encoding-and-one-hot-encoder-911ef77fb5bd>

Zhu, A. (2021, June 29). *Select Features for Machine Learning Model with Mutual Information*. Medium; Towards Data Science.

<https://towardsdatascience.com/select-features-for-machine-learning-model-with-mutual-information-534fe387d5c8>