K-Nearest Neighbor Algorithm and SVM algorithm

To what extent is the K-Nearest Neighbor algorithm more accurate than the SVM algorithm in detecting breast cancer in patients?

Word Count: 3519

Table of Contents

1.INTRODUCTION

The K-Nearest Neighbor (KNN) and Support Vector Machine (SVM) algorithms are both extremely important in the medical field, used to detect many forms of cancer with very high accuracy. Many people rely on these algorithms, which brings up the question of which is better? In this paper I will be focusing on the accuracy of these two algorithms in the detection of breast cancer in patients. Accuracy in these algorithms is extremely im portant. In Canada, 5000 women pass away due to breast cancer each year . A patient being misdiagnosed or undiagnosed may cause them their life. A 1% difference in the accuracy of one algorithm compared to another can be an issue of life or death for some people. For example, if algorithm 1 can detect breast cancer with 98% accuracy and algorithm 2 can detect breast cancer with 99%, using algorithm 2 can help save up to 50 more lives . This is why we need to use the most accurate algorithms at all times to maximize the amount of people we can help. In this paper I will be explaining how both the KNN and SVM algorithms work, writing a program which will measure the accuracy of both these algorithms using the same set of data and finally explaining what aspects of the algorithm may make it more accurate then the other. Through the use of SVM and KNN algorithms we are able to detect breast cancer in patients, but which algorithm is more accurate and effective in completing this task?

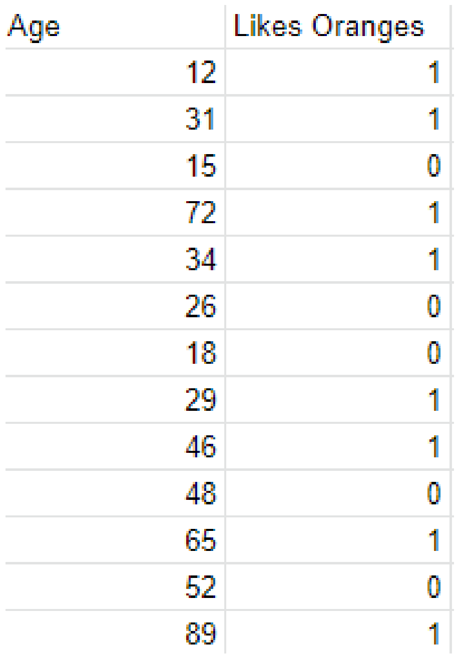
2.BACKGROUND INFORMATION

Machine learning is “the use and development of computer systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyze and draw inferences from patterns in data” (Oxford). In this paper we will be using machine learning to detect breast cancer, more specifically using the KNN and SVM algorithms.

2.1. SUPERVISED AND UNSUPERVISED ALGORITHMS

Firstly, I would like to explain what a supervised machine learning algorithm is, since both KNN and SVM are supervised. A supervised machine learning algorithm, uses labeled input data to learn a function that delivers suitable output when additional unlabeled data is presented. You can try to better understand this by imagining a child which you are teaching what a dog looks like. You then show them a set of images with a bunch of animals in it. Everytime you show them the dog you say “dog” and everytime its another animal you say “not dog”. After a while the child (algorithm) will understand what a dog looks like. You then show them a new set of images where they have to distinguish between dogs and other animals. They will mostly be able to distinguish between dogs and animals like pigs, but they may get confused with animals like wolves. This shows the importance of the data used in the training of the algorithm.

Usually supervised machine learning algorithms are used to solve classification and regression problems. The outcome of a classification task is a discrete value. "Likes oranges" and "does not like oranges," for example, are distinct. There is no such thing as a middle ground. Another example of a classification challenge is the aforementioned analogy of training a child to recognise a dog.

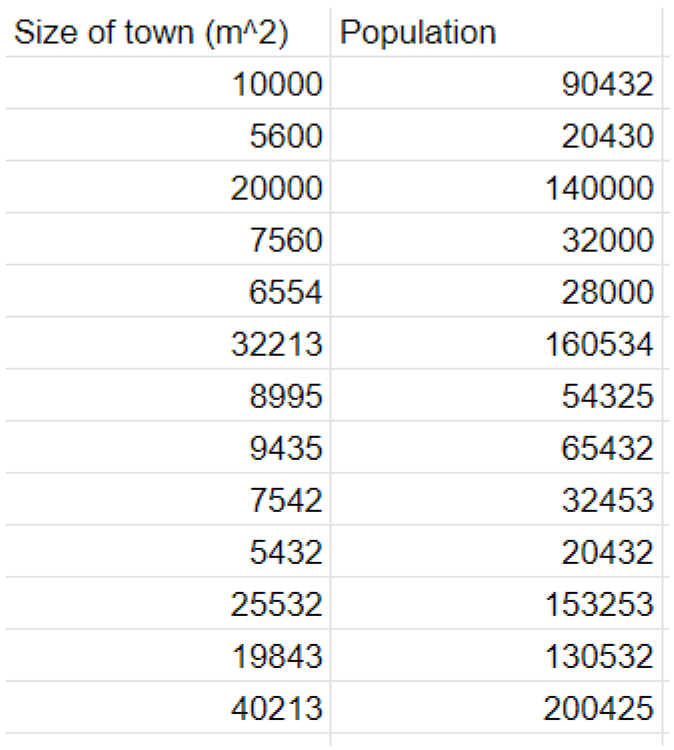


Age	Likes Oranges
12	1
31	1
15	0
72	1
34	1
26	0
18	0
29	1
46	1
48	0
65	1
52	0
89	1

Figure 1: Example of data used for KNN algorithm

In the data used for classification algorithms there are predictors and a label. In this set of data we are able to predict whether they like (1) or dislike (0) oranges based on how old they are (predictor). The output (label) of a classification method is typically represented as an integer number such as 1, -1, or 0. These figures are solely symbolic in this situation. They should not be subjected to mathematical processes since they would be nonsensical.

Next, KNN can be used for a regression algorithm. The outcome of a regression problem is a real number (a number with a decimal point). A dependent variable (the thing we are trying to guess given our independent variables) and an independent variable (or set of independent variables) are both present .



Size of town (m^2)	Population
10000	90432
5600	20430
20000	140000
7560	32000
6554	28000
32213	160534
8995	54325
9435	65432
7542	32453
5432	20432
25532	153253
19843	130532
40213	200425

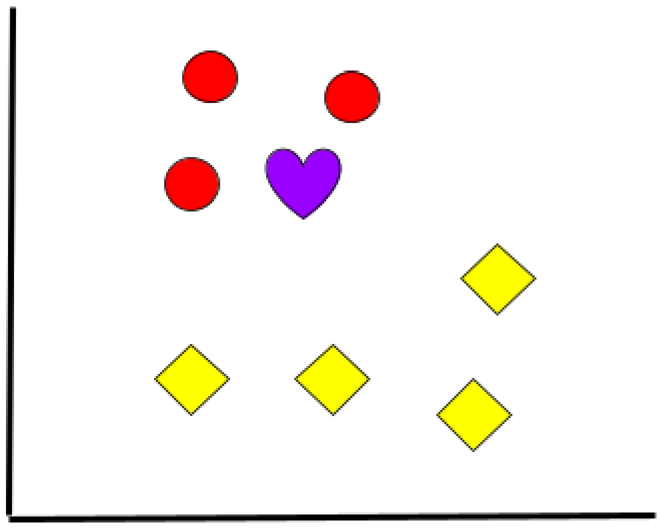
Figure 2: Size of town and population

In this case the population (dependent variable) is dependent on the size of the town (independent variable). In addition, each row is commonly referred to as an example, observation, or data point, but each column (without the label/dependent variable) is commonly referred to as a predictor, dimension, independent variable, or feature.

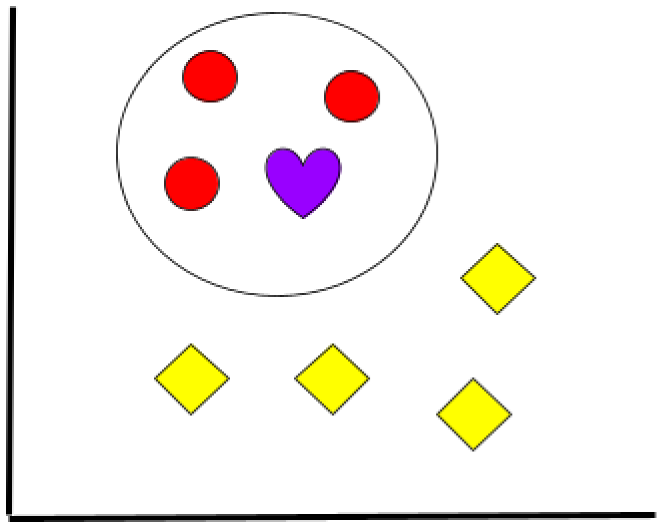
Supervised learning algorithms are opposed to unsupervised learning algorithms. Unsupervised algorithms uses input data without any labels. Referring to our previous example, it would be like teaching the child what a dog is, without you telling it if it's right or wrong, causing it to self correct. Unlike supervised learning, which aims to learn a function so that it may make predictions given new unlabeled data, unsupervised learning aims to understand the data's core structure in order to provide additional insight.

2.2 KNN ALGORITHM

The KNN (K-Nearest Nearest Neighbor) algorithm assumes that similar things are always near each other. To simply explain how the algorithm works we can draw a diagram.



In this diagram you are trying to classify the blue heart. In the algorithm the blue heart can only be classified as a red circle or a yellow rhombus, there is no middle ground. To do this we have to set a value for K. K is basically the amount of nearest neighbors we wish to vote from. In this case the three closest neighbors are the three red circles. So I will draw a circle encapsulating them.



In this set of data since the K value is 3 we can easily classify the blue heart as a red circle since it is surrounded by them. This equally shows the importance of the K value in your program as it can make or break the algorithm. So how do you choose the best K value for your algorithm? You can separate the training and validation datasets from the initial dataset to acquire the best K value. Next draw the validation error curve, the curve is made by attempting a different value of K starting and increasing by 1. After plotting points and making a line of best fit, you can predict the best value of K. For example, you input multiple K values and create a table using the accuracy output. It may look like the following table.

K Value	*Accuracy (%)*
1	50
2	44
3	35
4	29
5	25
6	21
7	18
8	16
9	14
10	10

14	10.4
16	10.7
18	11
20	12
22	13
24	14

Figure 3: K value table example

If we graph this table using desmos we can observe trends in the function.

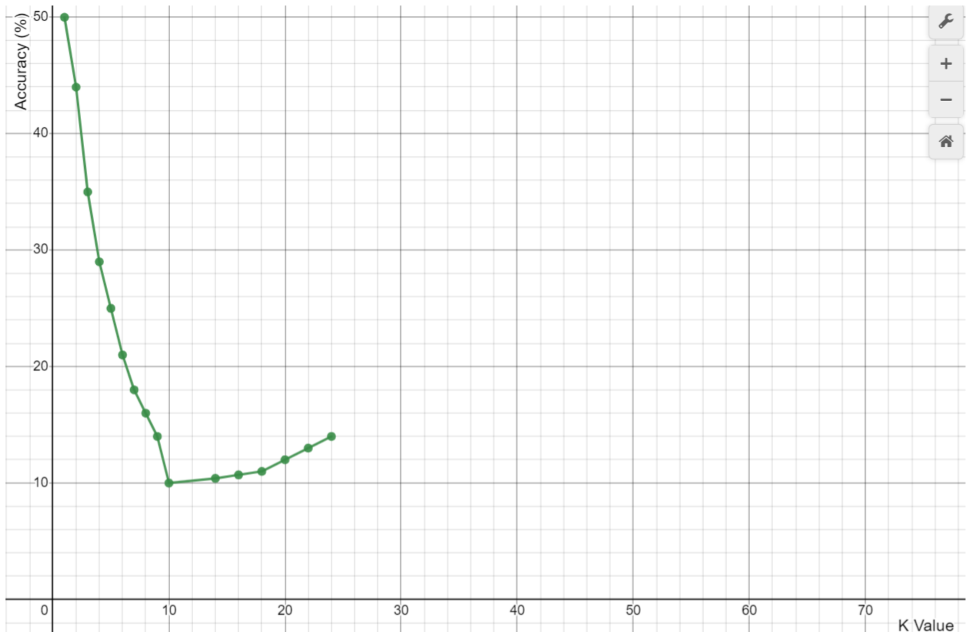


Figure 4: K-value vs Accuracy

The dependent variable is accuracy and the independent variable is the K value, hence the accuracy is on the y axis and the k value is on the x axis. By observing this graph, we can see that the accuracy will rapidly decrease until it hits its lowest accuracy, then it will exponentially increase for each additional K value. This means that, in this case, the K value of 10 will be the most accurate.

2.3 SUPPORT VECTOR MACHINE (SVM)

Just like KNN, the Support Vector Machine can be used to solve both classification and regression problems. When using the SVM algorithm, each data point is represented as a point in n-dimensional space (where n is the number of features you have), with each feature's value being the value of a certain coordinate. Next, we perform classification by identifying the hyper-plane that effectively distinguishes the two classes

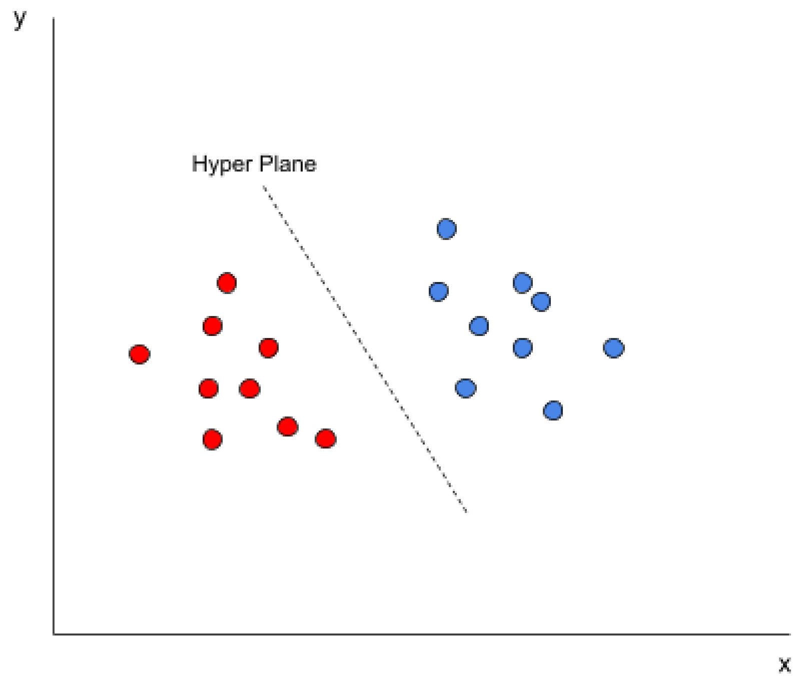


Figure 5: SVM example

Support vectors are data points that are closer to the hyperplane and have an impact on the hyperplane's position and orientation. We increase the classifier's margin using these support vectors. The hyperplane's location will vary if the support vectors are deleted. These are the ideas that aid in the development of our SVM. A feature of the SVM algorithm allows one to disregard outliers and locate the hyper-plane with the largest margin. In a case where no straight line can be created to distinguish between the two points, you must introduce higher dimensions so you can segregate them.

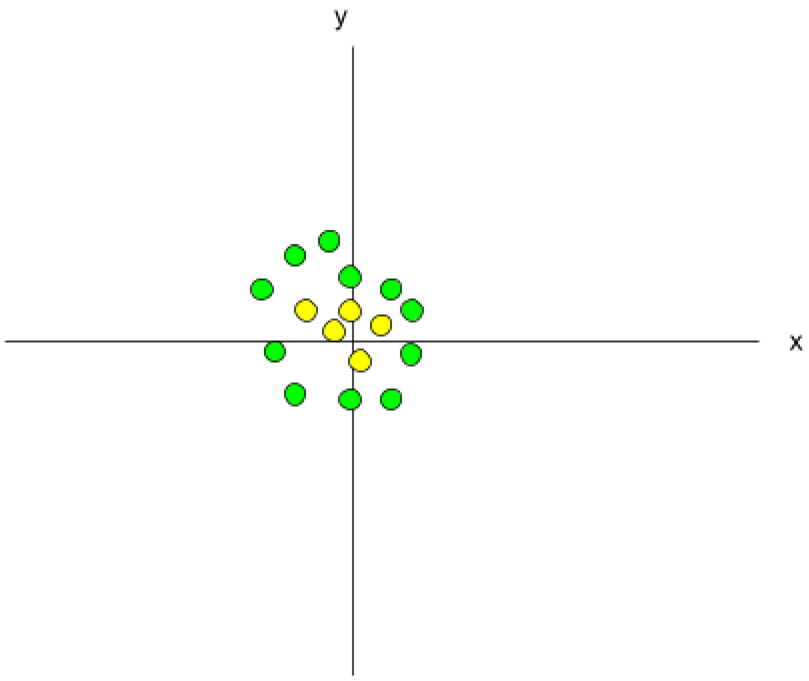
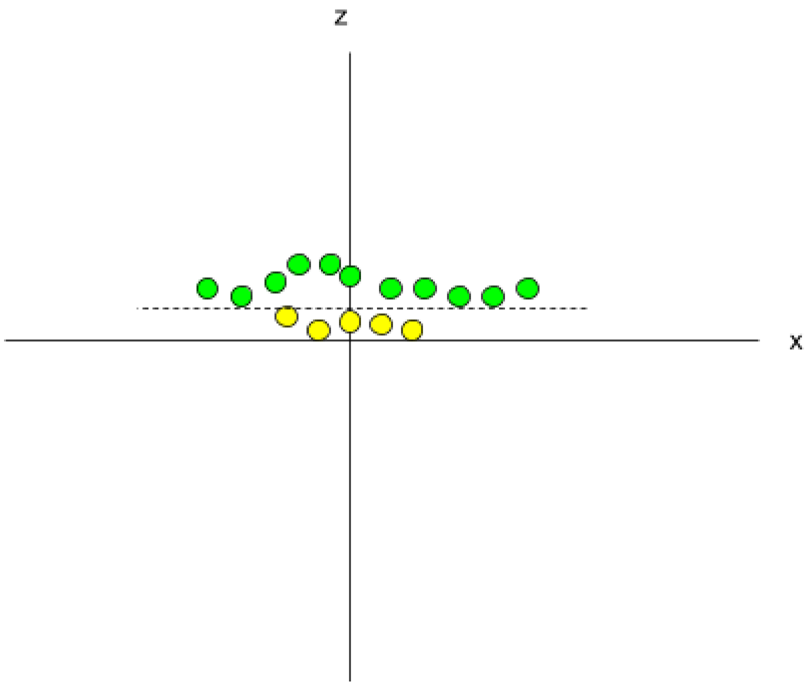
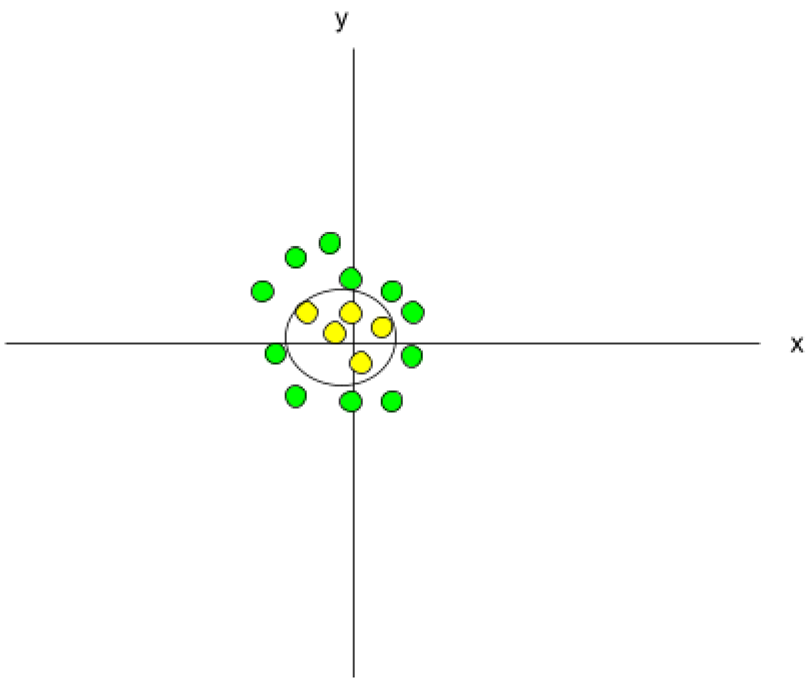


Figure 6: SVM with no linear hyperplane in the second dimension

To draw a linear hyperplane we can introduce a third feature, z. We can say that $z = y^2 + x^2$, since we square both y and x the sum of both will be positive, meaning z would be positive. This will cause the plane to look something like this.



With this we can draw the hyperplane. Fortunately we will not be needing to do this since SVM has a built-in technique called the kernel. The SVM kernel is a function that transforms non separable problems into separable problems by taking a low-dimensional input space and raising its dimension. It is most helpful in problems with non-linear separation. Simply said, it performs a number of incredibly sophisticated data transformations before determining how to split the data in accordance with the labels or outputs you've specified. So if we put the hyperplane in the original input it will look like this.



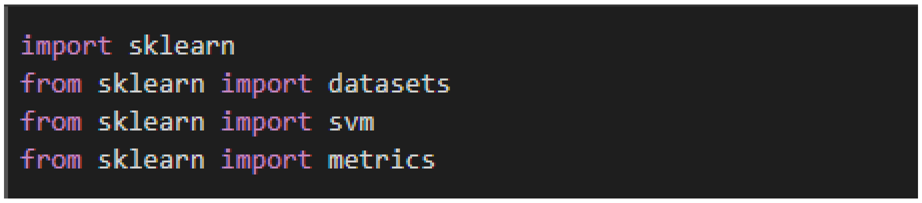
3. METHODOLOGY

3.1. DATASET

The dataset that we will be using comes from the California Irvine ML repository. This data set has 30 different features. Using a digital image of a fine needle aspirate (FNA) of a breast mass, features are calculated. They describe details of the cell nuclei. The thirty features described in the data set are based off the, radius (mean of distances from center to points on the perimeter), texture (standard deviation of gray-scale values), perimeter area, smoothness (local variation in radius lengths), compactness (perimeter^2 / area - 1.0), concavity (severity of concave portions of the contour), concave points (number of concave portions of the contour), symmetry and fractal dimension ("coastline approximation" - 1). The above-mentioned separating plane was obtained using the Multisurface Method-Tree (MSM-T), a classification method that builds a decision tree using linear programming (K. P. Bennett, "Decision Tree Construction Via Linear Programming," Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society, pp. 97-101, 1992). An exhaustive search in the domain of 1-4 features and 1-3 separation planes was used to select relevant features. According to [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets," Optimization Methods and Software 1, 1992, 23–34], the actual linear programme that was utilised to acquire the separation plane in the three-dimensional space is described. The program will be programmed on google collab, a google website which allows you to write and execute Python in your browser. Using this allows me to program on the go, without having to download and install tensor flow and pycharm.

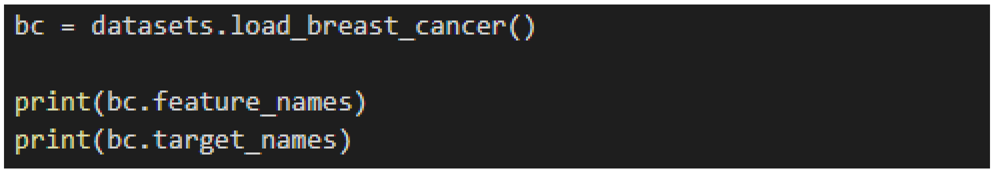
3.2. IMPLEMENTATION OF SVM

I was assisted in the programming of and understanding of both the SVM and KNN algorithm by Nicholas Singh, an AI and Software Engineer who I came into contact with after reading upon his article on classifying breast cancer using KNN. Implementing SVM is actually fairly easy. Firstly, we have to import sklearn. The scikit-learn (sklearn) library is a free software machine learning library for Python. It has many features, such as various classification, regression and clustering algorithms. From sklearn I then imported their datasets, SVM and metrics.



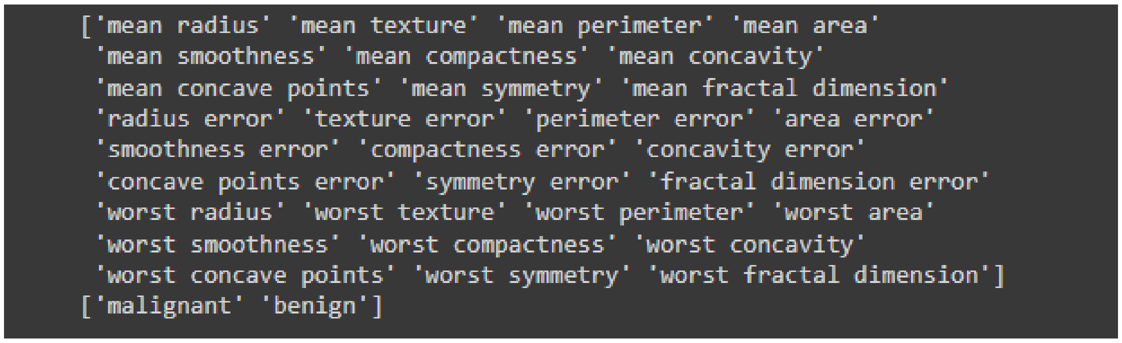
```
import sklearn
from sklearn import datasets
from sklearn import svm
from sklearn import metrics
```

To avoid having to import pandas (a dataset tool), I imported the data straight from sklearn as it also uses the data from the California Irvine ML repository. Importing SVM allows us to utilize the algorithm. Finally we import metrics, which implements several loss, score, and utility functions to measure classification performance. We will be using this to find the accuracy of the program. Next we have to load in the dataset, we can do this by using the function, datasets.learn_breast_cancer().



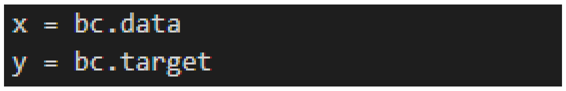
```
bc = datasets.load_breast_cancer()

print(bc.feature_names)
print(bc.target_names)
```

We assign the dataset to the variable bc (breast cancer) so we may call upon it for other various functions. For example we can use it to print the feature names and target names.



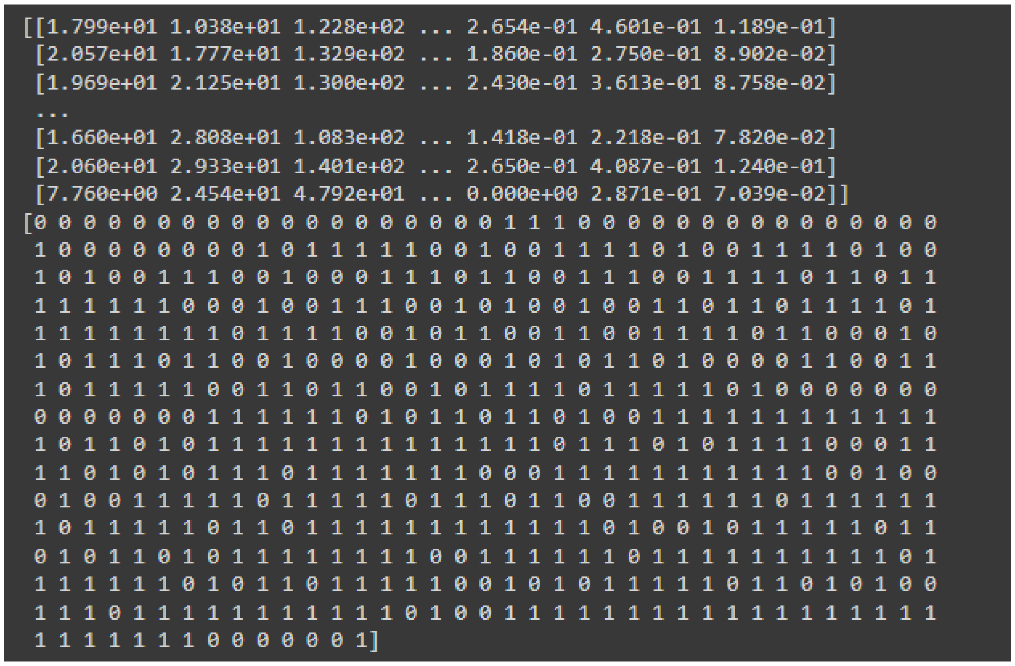
```
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension']
['malignant' 'benign']
```

As you can see, the algorithm is using these 30 features to make a prediction on whether the cell is malignant (has breast cancer) or benign (no breast cancer), with malignant and benign being the target names. Next we have to assign variables to the data and the targets of the data.



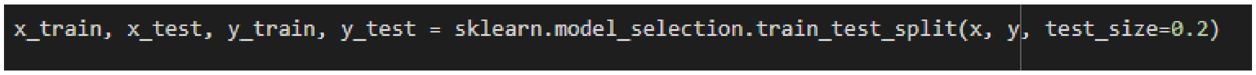
```
x = bc.data
y = bc.target
```

The variable x will hold the values of the features for the cells, and variable y will hold whether the cell is malignant or benign by using 0's and 1's (0 = benign, 1 = malignant). If we print the values of x and y we can see the values of the features of the cells in the data set and if they're malignant or benign.



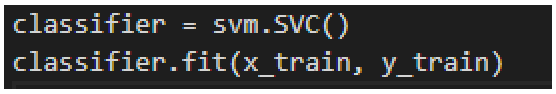
```
[[1.799e+01 1.038e+01 1.228e+02 ... 2.654e-01 4.601e-01 1.189e-01]
 [2.057e+01 1.777e+01 1.329e+02 ... 1.860e-01 2.750e-01 8.902e-02]
 [1.969e+01 2.125e+01 1.300e+02 ... 2.430e-01 3.613e-01 8.758e-02]
 ...
 [1.660e+01 2.808e+01 1.083e+02 ... 1.418e-01 2.218e-01 7.820e-02]
 [2.060e+01 2.933e+01 1.401e+02 ... 2.650e-01 4.087e-01 1.240e-01]
 [7.760e+00 2.454e+01 4.792e+01 ... 0.000e+00 2.871e-01 7.039e-02]]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 0 0 1 0 0 1 1 1 1 0 1 0 0 1 1 1 1 0 1 0 0
 1 0 1 0 0 1 1 1 0 0 1 0 0 0 1 1 1 0 1 1 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 1 1
 1 1 1 1 1 1 0 0 0 1 0 0 1 1 1 0 0 1 0 1 0 0 1 0 0 1 1 0 1 1 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 1 0 1 1 0 0 1 1 0 0 1 1 1 1 0 1 1 0 0 0 1 0
 1 0 1 1 1 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 1 0 1 1 0 1 0 0 0 0 1 1 0 0 1 1
 1 0 1 1 1 1 1 0 0 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 0 0 1 1
 1 1 0 1 0 1 0 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0 0
 0 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1
 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 0 1 1 1 1 1 0 1 1
 0 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 0 1 0 1 1 0 1 1 1 1 1 0 0 1 0 1 0 1 1 1 1 1 0 1 1 0 1 0 1 0 0
 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 0 0 0 0 0 0 1]
```

Now we have to separate the data into training and testing data. We do this by utilizing a sklearn function called the model selection train test split. This function basically allows us to set a percentage of the data which will be used in the testing phase where we predict the accuracy of the algorithm. In this case we will use 20% of the data to test the algorithm and 80% to train it. These are good margins because the training data is large, which will increase the accuracy of the algorithm and the testing data is not too small so the accuracy will be more precise.



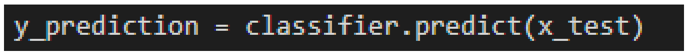
```
x_train, x_test, y_train, y_test = sklearn.model_selection.train_test_split(x, y, test_size=0.2)
```

This line of code splits the data into training and testing data sets. Where x_train and y_train are the respective training data for the features and targets, and where x_test and y_test are the respective testing data for the features and targets. Inside the parameters of the function, *sklearn.model_selection.train_test_split()*, is *(x, y, test_size=0.2)*, this is essentially what splits the dataset. Making 20% of the whole data set the test size and the other 80% the training size. Splitting the data is essential as it allows for you to effectively test the accuracy of the program since it makes sure that the algorithm is not just memorizing the answers. Next we have to implement our classifier.



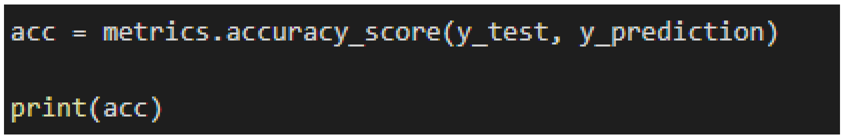
```
classifier = svm.SVC()
classifier.fit(x_train, y_train)
```

The first line of code essentially creates the classifier by using the function *svm.SVC()*. SVC stands for support vector classifier, it basically just creates the best hyperplane for your set of data. *Classifier.fit()* is what trains your algorithm so it can predict whether the cell is malignant or benign. Since this is a supervised learning algorithm, there are two arrays in the parameters. Now that we have trained the algorithm, we can test it using the predict function and by using the data we separated earlier.



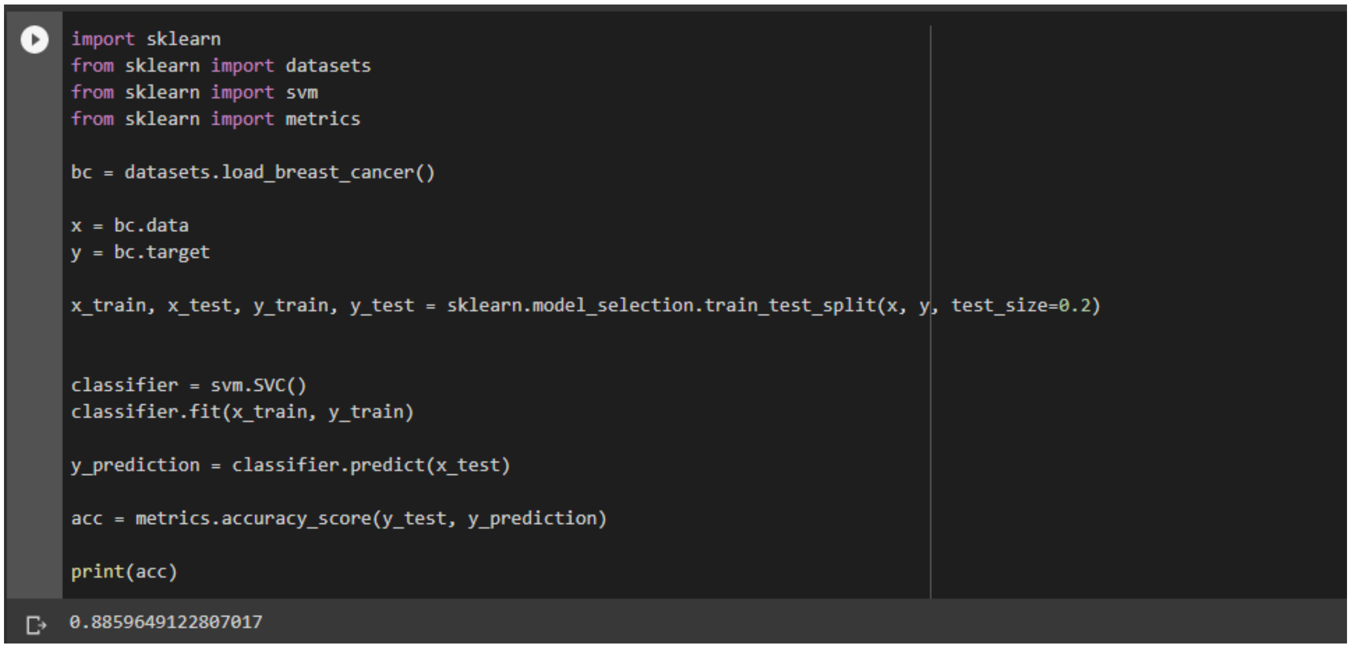
```
y_prediction = classifier.predict(x_test)
```

Essentially, this line of code uses the data we separated into the x_test variable, and tests the algorithm using it. The algorithm responds with whether the cell is malignant or benign and that information is stored within the variable y_prediction. We can then compare the answers of the algorithm to the correct answers and measure its accuracy.



```
acc = metrics.accuracy_score(y_test, y_prediction)

print(acc)
```

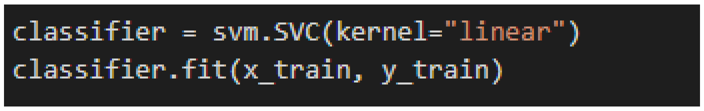
The first line of code uses the *metrics.accurace_score()* function from sklearn to compare the answers from the algorithm to the correct answers and compute the accuracy of the algorithm. We then assign this value to the variable acc and print the value.



```
import sklearn
from sklearn import datasets
from sklearn import svm
from sklearn import metrics

bc = datasets.load_breast_cancer()

x = bc.data
y = bc.target

x_train, x_test, y_train, y_test = sklearn.model_selection.train_test_split(x, y, test_size=0.2)


classifier = svm.SVC()
classifier.fit(x_train, y_train)

y_prediction = classifier.predict(x_test)

acc = metrics.accuracy_score(y_test, y_prediction)

print(acc)
```

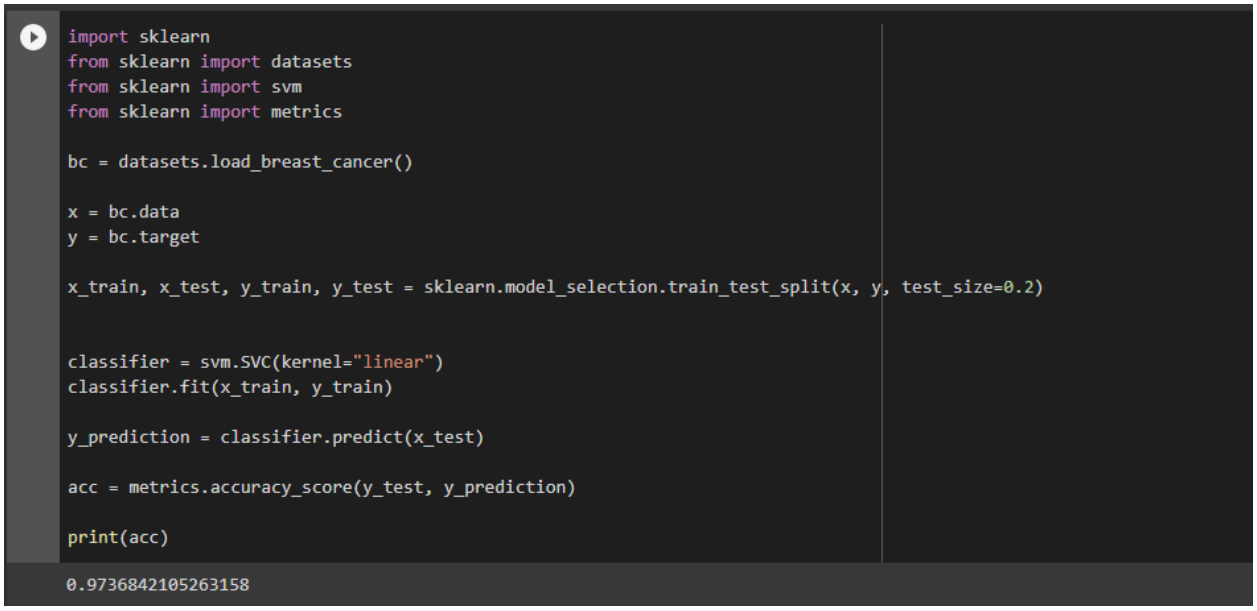
```
0.8859649122807017
```

The accuracy of the algorithm right now is 88.6% to 3 significant figures. As you probably have observed, this value is not very high. However this is because we did not add a kernel. We can add the kernel in the parameters of the svm.SVC() function. Kernel's may range from linear all the way to sigmoid. With each increase resulting in a higher accuracy. However due to the computing power needed for even polynomials, we will resort to only a linear kernel as it is the most efficient kernel for our dataset.



```
classifier = svm.SVC(kernel="linear")
classifier.fit(x_train, y_train)
```

As you can see, in the parameters of the *svm.SVC()* function we added a linear kernel. This will allow the algorithm to predict whether the cell is malignant or benign with way more accuracy.



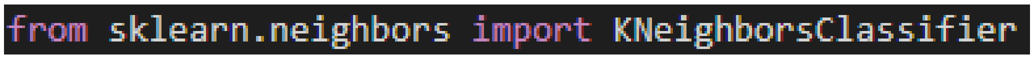
```
import sklearn
from sklearn import datasets
from sklearn import svm
from sklearn import metrics

bc = datasets.load_breast_cancer()

x = bc.data
y = bc.target

x_train, x_test, y_train, y_test = sklearn.model_selection.train_test_split(x, y, test_size=0.2)


classifier = svm.SVC(kernel="linear")
classifier.fit(x_train, y_train)

y_prediction = classifier.predict(x_test)

acc = metrics.accuracy_score(y_test, y_prediction)

print(acc)
```

```
0.9736842105263158
```

The algorithm can now detect whether the cell is malignant or benign with about 97.4% accuracy. This greatly increased the accuracy from before by about 9%. This shows how a simple addition to a program can make it way more efficient. But now how does SVM compare to KNN in the detection of malignant or benign cells?

3.3. IMPLEMENTATION OF KNN

To find the accuracy of KNN is quite simple. All we need to do is make some slight adjustments to our previous code for SVM. Firstly we have to import KNN from sklearn.We can import KNN by using the following command.

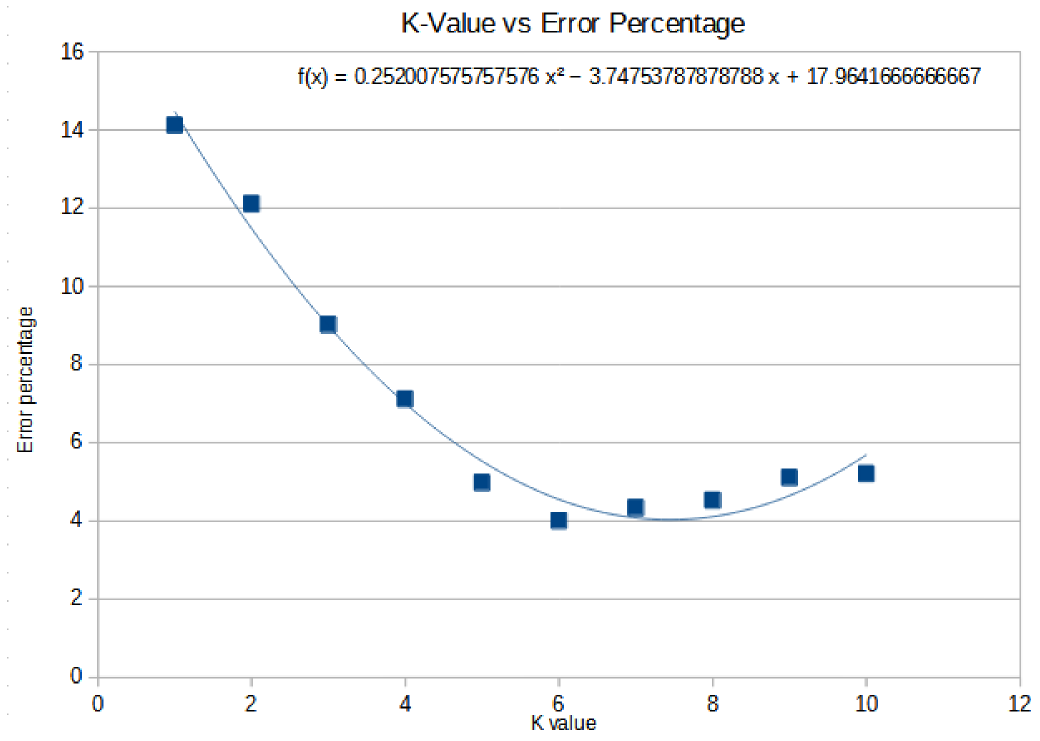


```
from sklearn.neighbors import KNeighborsClassifier
```

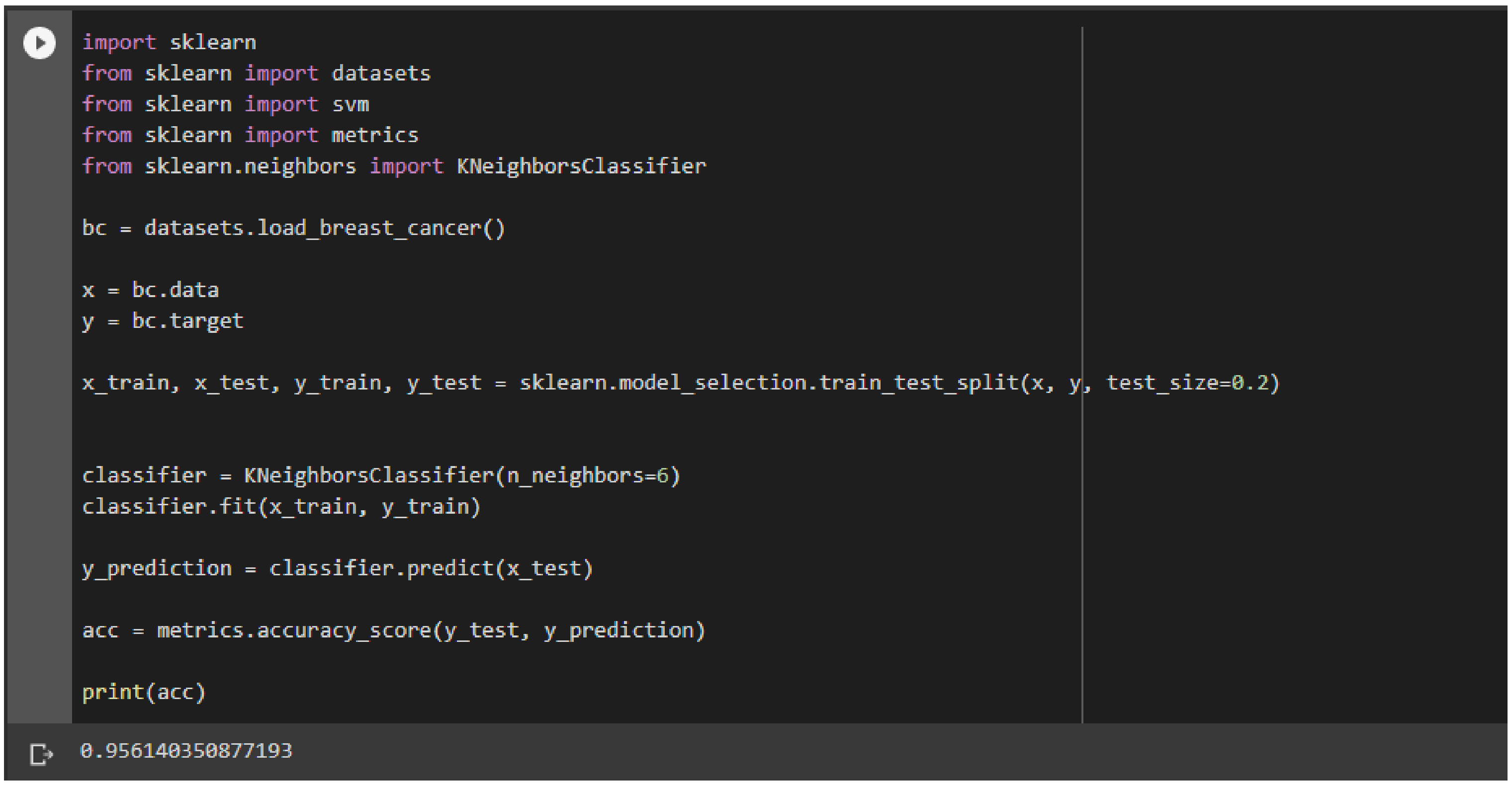
Now all we have to do is change the classifier from svm to KNN. This can be done by removing the previous line of code (*classifier = svm.SVC(kernel="linear"))* with *classifier = KNeighborsClassifier(n_neighbors=1)*, where n_neighbors equals the K value. Now to get the most accurate prediction, we need to find the K value with the least error percentage. I experimentally deduced a table of values with the K value and the respective error percentage.

K VALUE	**ERROR (%)**
1	14.12
2	12.11
3	9.02
4	7.12
5	4.98
6	4.40
7	4.51
8	4.53
9	5.11
10	5.21

We can then plot this data on a chart and get an equation for the line of best fit to solve for the error% for any K value.



If we further examine this graph we can see that the K value with the least error percentage is 6. We can then plug this value into the program and print the accuracy.



```
import sklearn
from sklearn import datasets
from sklearn import svm
from sklearn import metrics
from sklearn.neighbors import KNeighborsClassifier

bc = datasets.load_breast_cancer()

x = bc.data
y = bc.target

x_train, x_test, y_train, y_test = sklearn.model_selection.train_test_split(x, y, test_size=0.2)


classifier = KNeighborsClassifier(n_neighbors=6)
classifier.fit(x_train, y_train)

y_prediction = classifier.predict(x_test)

acc = metrics.accuracy_score(y_test, y_prediction)

print(acc)
```

```
0.956140350877193
```

As you can see, KNN was able to predict whether the cell was malignant or benign with about 95.6% accuracy. Therefore, we can conclude from this experiment that KNN is about 1.8% less accurate than SVM (*97.4%-95.6%=1.8%)*, but why may this be the case?

4. EVALUATION

The reason why SVM is more accurate than KNN in this case is due to the sheer amount of features and the size of the data. Since KNN needs to store the training data in order to function, it can be computationally expensive in terms of both time and storage if the data set is very vast. Usually, alternative supervised learning models, like SVM don't work like this.
Since KNN relies on computing distances, it can be quite sensitive to the size of the input. The computed distances for features with a higher scale can be very high and could result in subpar results. Scaling the data before executing the KNN algorithm is therefore advised, however in our case, it will be very difficult to scale the data due to the amount of data and features. SVM on the other hand is very good with a large set of features. The large amount of features creates more dimensions, which SVM is able to handle easily with the kernel feature. However SVM undeperforms when the number of samples is larger than the number of features, and vice versa for the KNN algorithm. This is what makes the margin of error between SVM and KNN only 1.8%. The data set we are using has both disadvantages and advantages for both algorithms, however it pairs better with SVM, shown by its higher accuracy.

5. CONCLUSION

To conclude, both the KNN and SVM algorithms are both very accurate algorithms in the detection of breast cancer. We went over how both the SVM and KNN algorithms work, and how we can implement them. We found that the SVM algorithm has a greater ability to detect whether a cell is malignant or benign with more accuracy than KNN. We explored how we can use kernels with the SVM algorithm to further increase the accuracy, and how we can find the most optimal K value for the KNN algorithm by plotting points on a graph and observing the trends. For the set of data we used, the SVM algorithm was more accurate than the KNN algorithm, with a 1.8% more predictive accuracy. I believe that in the medical field we should be using the SVM algorithm due to its ability to handle the amount of features that are present in a cell by using its kernel function. Overall, the experiment was a success and we were able to deduce the best algorithm to use to predict whether a cell is malignant or benign.

6.WORKS CITED

-Singh, Nicholas. “Classifying Breast Cancer Using Knns.” *Medium*, The Startup, 18 Jan. 2021, https://medium.com/swlh/classifying-breast-cancer-using-knns-75ac5cbad3eb.

-Ray, Sunil. “SVM: Support Vector Machine Algorithm in Machine Learning.” Analytics Vidhya, 26 Aug.2021, https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/.

-Srivastava, Tavish. “Support Vector Machine: SVM Classification Algorithm.” *Analytics Vidhya*, 25 June 2020, https://www.analyticsvidhya.com/blog/2014/10/support-vector-machine-simplified/?utm_source=blog&utm_medium=understandingsupportvectormachinearticle.

-“Learn.” *Scikit*, https://scikit-learn.org/stable/index.html.

-Al-hadidi, Moh’d Rasoul, et al. “Breast Cancer Detection Using K-Nearest Neighbor Machine …” *Breast Cancer Detection Using K-Nearest Neighbor Machine Learning Algorithm* , 9 Aug. 2016,https://www.researchgate.net/publication/317071879_Breast_Cancer_Detection_Using_K-Nearest_Neighbor_Machine_Learning_Algorithm.

-Harrison, Onel. “Machine Learning Basics with the K-Nearest Neighbors Algorithm.” *Medium*, Towards Data Science, 14 July 2019,https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761.

-Ray, Sunil. “SVM | Support Vector Machine Algorithm in Machine Learning.” *Analytics Vidhya*, 13 September 2017, https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/. Accessed 28 February 2022.

-Zett, Julian. “Breast Cancer Classification Using Support Vector Machine (SVM).” *Towards Data Science*, 22 November 2018, https://towardsdatascience.com/breast-cancer-classification-using-support-vector-machine-svm-a510907d4878. Accessed 28 February 2022.

-Danilo Bzdok, Martin Krzywinski, Naomi Altman. Machine learning: Supervised methods, SVM and kNN. Nature Methods, Nature Publishing Group, 2018, pp.1-6. Ffhal-01657491f

-Zett, Julian. “Breast Cancer Classification Using Support Vector Machine (SVM).” *Towards Data Science*, 22 November 2018, https://towardsdatascience.com/breast-cancer-classification-using-support-vector-machine-svm-a510907d4878. Accessed 28 February 2022.

-Christmann, Andreas, and Ingo Steinwart. *Support Vector Machines*. Springer New York, 2008. Accessed 28 February 2022.

-Prabhakar, Sunil Kumar, and Harikumar Rajaguru. *KNN Classifier and K-Means Clustering for Robust Classification of Epilepsy from EEG Signals. A Detailed Analysis*. Anchor Academic Publishing, 2017. Accessed 28 February 2022.

-Ruscica, Tim. “SVM Implementation.” *Techwithtim.net*, 25 Jan. 2019, https://www.techwithtim.net/tutorials/machine-learning-python/svm-p-3-implementation/.