
Support Vector Machines Report

Darryl Vas Prabhu

Department of Computer Science and Engineering
University at Buffalo, Buffalo, NY 14260
dvasprab@buffalo.edu

1 Support Vector Machines

¹Classification of data is one of the goals of machine learning. Image classification, handwriting analysis, cancer detection, are some of the applications of machine learning when using a classifier model. A model which classifies data into classes based on learnings of the training data-set is called classification model. One might argue that we could use logistic regression, which is another classification model used to categorize data. This is not an issue as long as the model is a fit for the data. Logistic regression uses a tight decision boundary to classify data into categories. However, there might be cases where most of the incoming or new data-points would fall close to the decision boundary, in which case they would be labelled incorrectly. In order to minimize the error of incorrect classification a linear regression as well as classifier model called support vector machines was developed. In support vector machines, the classification is based on a margin, in fact the best fit margin between 2 close data points, belonging to different classes.

In support vector machines, we attempt to classify if a new data-point to a particular class using the maximum margin principle. There can be 2 types of data : 1) linearly separable data which can be separated easily using decision boundary and 2) non-linearly separable in which the data points appear clustered together and decision making requires some extra step to separate these points. Consider a pair of data vectors **green** on left - belonging to the class of diabetes and the other one **blue** on the right - not belonging to the diabetes from the training data Figure 1[?]] A margin is found here which can classify the 2 data points. In this representation where the feature set is 2-dimensional, the margin is a line. If data-points appear with n features ,then the margin would be in $(n-1)$ th dimension and is called a hyperplane.

Now if a new data point comes in, and it's value is on the left side of the margin , it would be labelled as diabetes. There can be many hyper-planes and this depends on which two vectors from two different classes are selected for deciding the margin. Selecting the best hyperplane such that the marginal distance is large enough is precisely the goal of SVM. Computing a large margin is usually preferred even though a few incoming data points may be classified incorrectly. This is usually allowed, considering that a huge portion new data points are classified correctly. If a small margin is selected to fit the training data, then new incoming data might have a high probability of getting classified incorrectly.

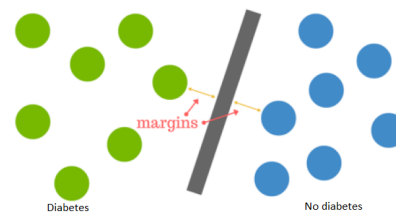


Figure 1: Support Vector Machines

The maximum margin principle can be explained in geometric meaning as shown in Figure 2[?]] and Figure 3[?]]:

¹Support Vector Machines.

Maximum Margin Principle

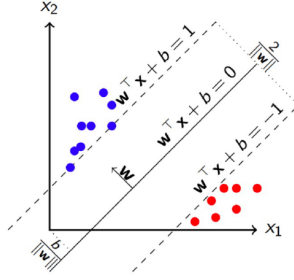


Figure 2: Maximum margin principle

Definitions

Define the hyperplanes H such that:
 $w^T x + b \geq +1$ when $y_i = +1$
 $w^T x + b \leq -1$ when $y_i = -1$

H_1 and H_2 are the planes:
 $H_1: w^T x + b = +1$
 $H_2: w^T x + b = -1$
 The points on the planes H_1 and H_2 are the tips of the **Support Vectors**
 The plane H_0 is the median in between, where $w^T x + b = 0$

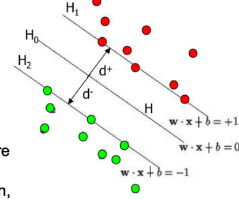


Figure 3: Marginal hyperplane computation

We use the margin

$$y = w_0 + w^T x$$

and set it to 0. The points above the line are tagged positive and the points below the line are tagged negative. The closest points which determine the best fit maximum margin are called support vectors. The distance between the two points is given as $\frac{2}{\|w\|}$ where $\|w\|$ is the norm of the weight vector. The optimum margin will have the size of the margin $\frac{2}{\|w\|}$ set to maximum or in other words minimize $\frac{\|w\|}{2}$ such that

$$y_i = \{ 1, w^T x + b \geq 1 \}$$

$$y_i = \{ -1, w^T x + b \leq -1 \}$$

$y_i * (w^T * x) \geq 1$ is the condition for a correctly classified data-point.

If the above condition does not hold true then the point is a mis-classified point. There can arise a situations, where a multiple data points can be valued between the support-vectors which were used to compute the margin from training data, and be classified incorrectly. Then the distance of these new points to the hyperplane is calculated and summed. This regularization method is used to minimize the error and find the best hyperplane. The hyperplane which gives the least error is then selected among all the hyper-planes.

In case of non linearly separable data, the data is not easily separable in a sense that the group of data-points belonging to different classes are intertwined. A hyperplane cannot divide the point using normal techniques. In this case we move data-points to a higher dimension to find the hyperplane. This is the transformation step. This is done with the help of certain functions called kernel functions and is called the kernel trick. The hyperplane or support vector classifier which separates the data into 2 separate groups in the higher dimension is found and used as the SVC. Note that the kernel function does not actually convert the data-points but only calculates the relationship between the data-points as if they were in a higher dimension as shown in Figure 4[?]

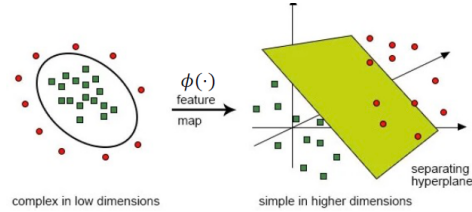


Figure 4: Non linear separable kernel trick

2 Experiments

To illustrate the application of SVM a dataset containing various features of two types of raisins was fetched from [8] Figure 5 shows a snippet of the various features of the raisins which are self explanatory. In this experiment the class of Benshi resins was predicted based on two features i.e. Area and Perimeter.

2.1 Decription of the dataset

The column headers in the below are the various features of the dataset and the last one "class" is class of the raisin which is either Besni or Kecimen.

- Area - Area of the raisin.
- Perimeter - The Perimeter of the raisin.

Since the class is a categorical data, OneHotEn-coding was used to convert the data into numerical data as a preprocessing step as shown in Figure 6 The features were Area and Perimeter were selected based on the correlation matrix which illustrated that area and perimeter had a high correlation with respect to Besni class. This is indicated with the blue circle in Figure 7

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent	Perimeter	Class
712	77985	473.340705	214.092031	0.891866	82967	0.674377	1214.981	Besni
11	43725	301.322218	186.950629	0.784258	45021	0.697068	818.873	Kecimen
356	58460	290.753556	258.384354	0.458545	60254	0.774756	908.357	Kecimen
500	143386	469.276508	397.310190	0.532160	146328	0.750260	1422.014	Besni
530	90559	473.575846	246.919968	0.853316	95477	0.673231	1328.744	Besni

Figure 5: Dataset for Raisins

	Area	MajorAxisLength	MinorAxisLength	Eccentricity	ConvexArea	Extent	Perimeter	Class
712	77985	473.340705	214.092031	0.891866	82967	0.674377	1214.981	Besni
11	43725	301.322218	186.950629	0.784258	45021	0.697068	818.873	Kecimen
356	58460	290.753556	258.384354	0.458545	60254	0.774756	908.357	Kecimen
500	143386	469.276508	397.310190	0.532160	146328	0.750260	1422.014	Besni
530	90559	473.575846	246.919968	0.853316	95477	0.673231	1328.744	Besni


```

from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import make_column_transformer

transformer = make_column_transformer((OneHotEncoder(), ['Class']),remainder='passthrough')
transformed = transformer.fit_transform(df)
transformed_df = pd.DataFrame(transformed,columns=transformer.get_feature_names_out())

df = transformed_df
df

```

	onehotencoder__Class_Besni	onehotencoder__Class_Kecimen	remainder__Area	remainder__MajorAxisLength	remainder__MinorAxisLength	remainder__Eccentricity	remainder__ConvexArea	remainder__Extent	remainder__Perimeter
0	1.0	0.0	77985.0	473.340705	214.092031	0.891866	82967.0	0.674377	1214.981
1	0.0	1.0	43725.0	301.322218	186.950629	0.784258	45021.0	0.697068	818.873
2	0.0	1.0	58460.0	290.753556	258.384354	0.458545	60254.0	0.774756	908.357

Figure 6: Classes changed to numerical values

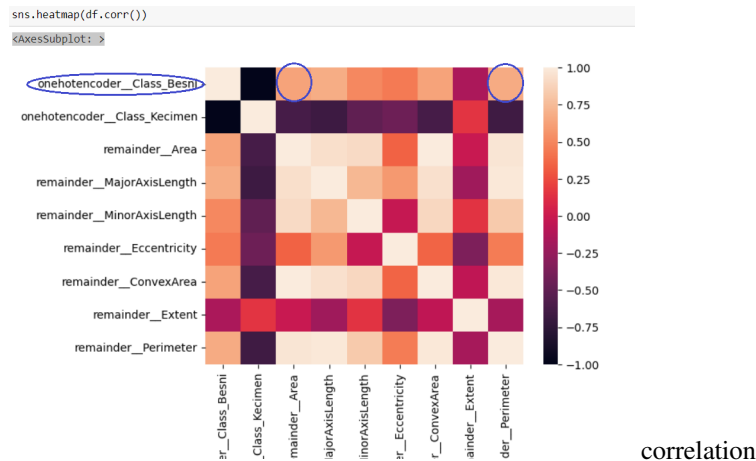


Figure 7: Correlation between Besni,Area and Perimeter

Training and test data was then split accordingly with 75% of data as training data and remaining 25% of the data as test data to test if the Besni raisins will be predicted correctly.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.25, random_state = 0)

X_train, X_test, z_train, z_test = train_test_split(X, Z, test_size = 0.25, random_state = 0)
```

Figure 8: Split data using into test and training

In order to have standardization of the features so that overestimation due to one of the features does not occur, sklearn's StandardScaler() was used on the input test and training data. The SVC classifier which is the support vector classifier was then trained on the training data-set and after training it was tested against the X_{test} which are the unseen test input features. The prediction y_{pred} shown below indicates the classes of X_{test} based on the SVC classifier. The total length of the test data was 225 records and the confusion matrix indicated that the sum of the true negative and true positive classified points is 194 giving an accuracy of 86%. The confusion matrix is given below

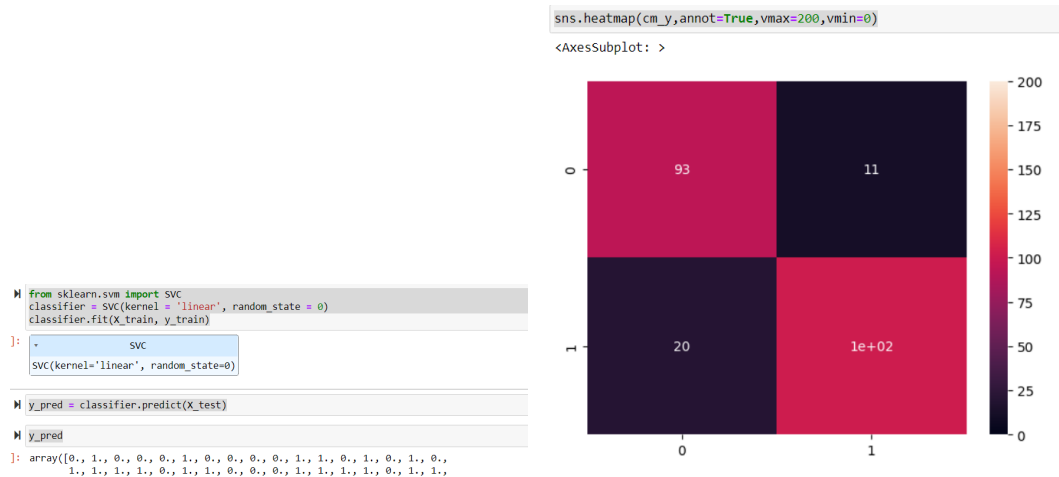


Figure 9: Class prediction denoted by y_{pred}

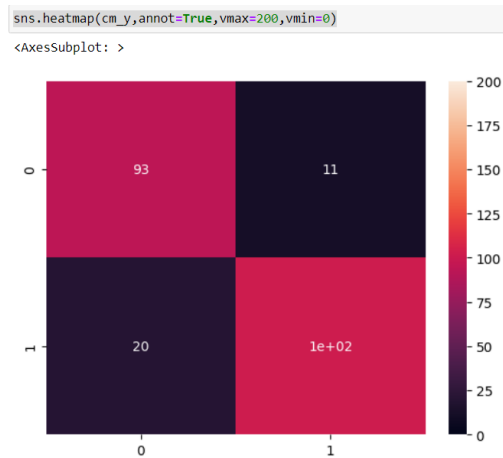


Figure 10: Confusion Matrix

The correlation between the y_{pred} and y_{test} verifies the confusion matrix % correct by informing .86 of the data was accurately classified. The heatmap below indicates the same for y_{pred} vs y_{test} . Finally we have the Support Vector classifier training data plotted with respect to the features Area and Perimeter for both training data and test data. From the same we observe that although certain points are incorrectly classified near the margin, the same on the test data indicates that the most of the Besni raisins which are green dots are classified correctly in the green background and blue dots which are not Besni raisins are in the blue background. A few outliers lie which is natural in any machine learning model.

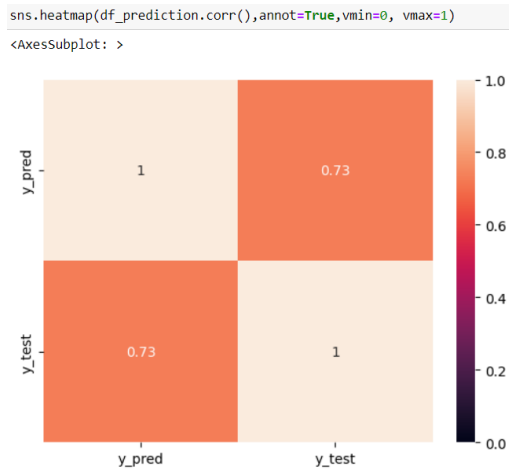


Figure 11: Y_pred vs Y_test

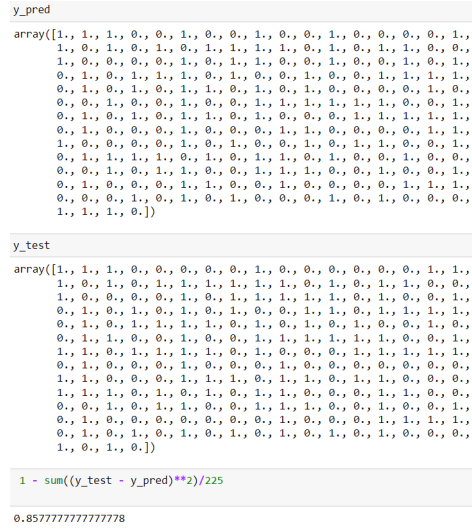


Figure 12: Percentage correct

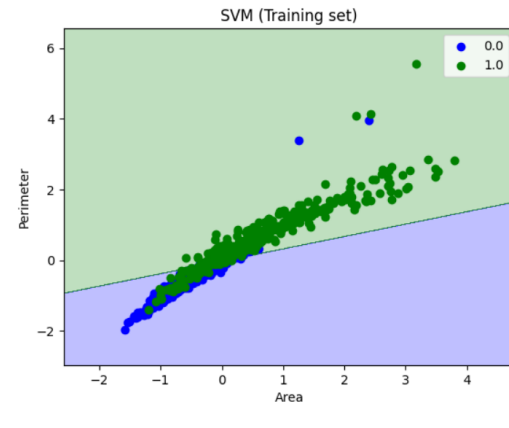


Figure 13: SVM training set

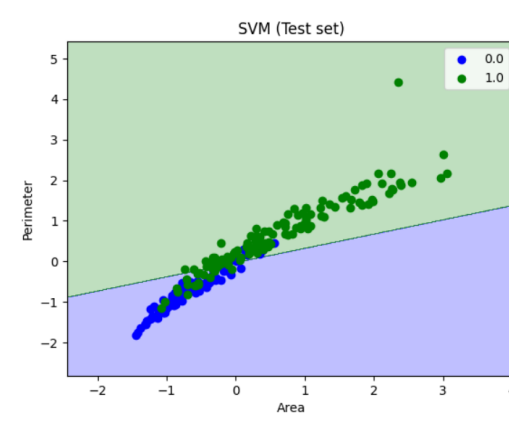


Figure 14: SVM testing set

References

- [1] Numpy documentation. <https://numpy.org/doc/stable/reference/>.
- [2] Pandas documentation. <https://pandas.pydata.org/docs/>.
- [3] Scikit learn. <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>.
- [4] *Probabilistic Machine Learning An Introduction*. The MIT Press, 2022.
- [5] Changyou Chen. 100 days of ml code. <https://github.com/cchangyou/100-Days-Of-ML-Code/blob/master/Code/Day%2013%20SVM.md>.
- [6] Changyou Chen. Support vector machines.
- [7] Nik datagy. One hot encoding. <https://datagy.io/sklearn-one-hot-encode/>.
- [8] Kaggle. Kaggle dataset. <https://www.kaggle.com/code/rafarsilva/raisin-svm-classification/data>.
- [9] Krish Naik. Svm basic intuition. <https://youtu.be/H9yACitf-KM>.

Backpropagation Report

Darryl Vas Prabhu

Department of Computer Science and Engineering
University at Buffalo, Buffalo, NY 14260
dvasprab@buffalo.edu

1 Backpropagation

¹ In today's age of big data, there is always scope for development of machine learning models. There is influx of data and hence efficient usage of this data becomes vital for learning or predicting new outcomes. It may seem as though machines cannot understand feedback, however provided sufficient training data, machines can be modelled to learn patterns and develop functions which can then be used on new incoming data. One such model is called the Backpropagation model and the works on the basis of neural networks. An example of a neural network is shown in Figure 1

When we hear the word "neural networks", our mind is prone to think of the nervous system and neurons which send signals to the brain. The machine learning version of the model is not too dissimilar in this regard. Consider the case of classification of digits. Each image contains pixels which can be transformed into vectors holding a value. These values form the first layer of the network or are called the inputs. The pixel value inside these neurons is called activation. In a neural network the input layer is connected to further layers which are hidden and then to the output layer which is a class representing the output which are numbers from 0 to 9 at-least in this example. In simple terms the neurons in the first layer have certain values which fire up the second layer and further on the hidden layers till the output layer is reached. At the output layer, the result is obtained which is corresponding to the input layer. But what activates the hidden layers and how the data is perceived ?

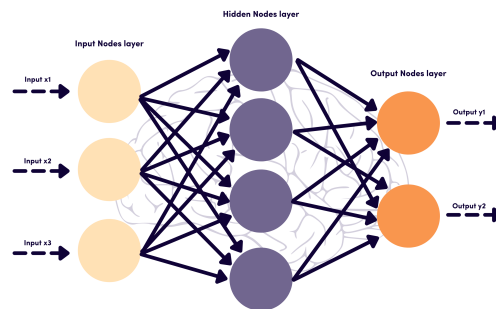


Figure 1: One hidden-layer Neural Network

Initially each of the neurons in the first layer are weighted randomly, in the sense that the weights for each of the inputs are assigned at random and summed. You can view it as a scaled linear combination of weights and the neurons activation. A bias is included such that it decides whether the neuron is active or not. Since in machine learning we are required to standardize the value between 0 and 1, a sigmoid function is used on the summed value, although in modern neural network models, ReLU functions has been known to have give better outcomes. Note that this combination of weighted activation function and the bias can be viewed in matrix form as in Figure 2.

In this way a recursively moving forward with new weights assigned to the hidden layers activation neurons, a feed forward neural network transmits the weighted and biased input. At the final layer, the value for each output neuron is compared with the final value obtained from the previous layer. A cost is determined by summing the squared difference between each output value and it's corresponding activation. The cost now denotes the amount of loss or error between the actual data and the calculated data based on the weights which were initialized. In order to improve this the

¹Backpropagation

Sigmoid

$$a_0^{(1)} = \sigma(w_{0,0} a_0^{(0)} + w_{0,1} a_1^{(0)} + \dots + w_{0,n} a_n^{(0)} + b_0)$$

Bias

$$\begin{bmatrix} w_{0,0} & w_{0,1} & \dots & w_{0,n} \\ w_{1,0} & w_{1,1} & \dots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \dots & w_{k,n} \end{bmatrix} \begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} = \begin{bmatrix} ? \\ ? \\ \vdots \\ ? \end{bmatrix}$$

Figure 2: Matrix form of weighted inputs and bias

Visually, here's what we're doing:

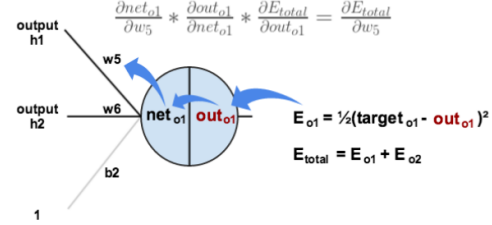


Figure 3: Minimizing the Cost with respect to weights

machine feeds back this information to the hidden network to improve the model. This is done in mathematical sense by reducing the cost function and is done through a process called gradient descent. We improve the weights for those neurons which seem to have a great impact on the next layer output and this happens at every layer and at every neuron. The weights are adjusted according to minimize the cost function, i.e. back-propagation occurs recursively to make the model learn the correct weights required until a good accuracy is reached and the output layer is approximately trained to fit the training data. The goal of back-propagation is thus to reduce the cost by adjusting the weights at every layer of the model. The mathematics that goes behind the reasoning of back-propagation is summarized in the below Figure 3:

2 Experiments

To illustrate back-propagation, some sample inputs were taken and OR operation was fed in random order for 10000 iterations to train the model. As the training samples are fed, we can observe that the error decreases.

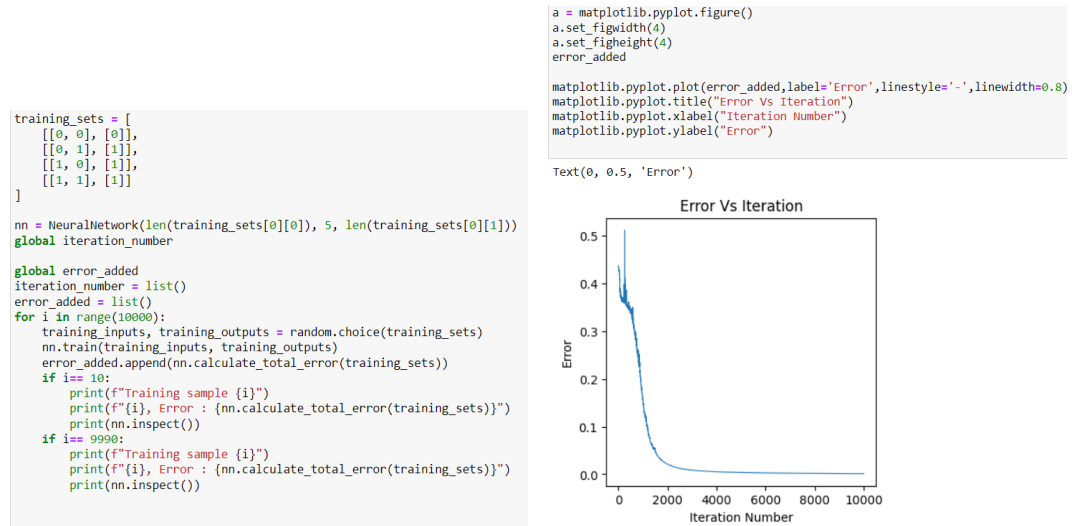


Figure 4: OR training dataset

Figure 5: Error vs Iteration

The weights and biases during the 10th training data and 9990th training data is shown here Figure 6 , Figure 7. Notice how the weights are changed . It indicates that the error fell rapidly with more and more training.

```
Training sample 10
10, Error : 0.3896905328825308
```

```
-----
* Inputs: 2
-----
```

```
Hidden Layer
```

```
Neurons: 5
```

```
Neuron 0
```

```
Weight: 0.4891525765041752
```

```
Weight: 0.2734092020906974
```

```
Bias: 0.41653984419151113
```

```
Neuron 1
```

```
Weight: 0.2973239726650529
```

```
Weight: 0.23395385592296175
```

```
Bias: 0.41653984419151113
```

```
Neuron 2
```

```
Weight: 0.2133950045077842
```

```
Weight: 0.6535484615301254
```

```
Bias: 0.41653984419151113
```

```
Neuron 3
```

```
Weight: 0.7073007207627064
```

```
Weight: 0.473072022217133
```

```
Bias: 0.41653984419151113
```

```
Neuron 4
```

```
Weight: 0.1162932866171462
```

```
Weight: 0.253491071102056
```

```
Bias: 0.41653984419151113
```

```
-----
* Output Layer
```

```
Neurons: 1
```

```
Neuron 0
```

```
Weight: 0.2837811770103914
```

```
Weight: 0.7946370014269102
```

```
Weight: 0.17303745638906762
```

```
Weight: 0.14226123876644636
```

```
Weight: 0.1829947044534695
```

```
Bias: 0.820742329903317
```

```
Training sample 9990
```

```
9990, Error : 0.001729230823105676
```

```
-----
* Inputs: 2
-----
```

```
Hidden Layer
```

```
Neurons: 5
```

```
Neuron 0
```

```
Weight: 0.24122407528757786
```

```
Weight: -0.010453832448332647
```

```
Bias: 0.41653984419151113
```

```
Neuron 1
```

```
Weight: 2.0950071812666016
```

```
Weight: 2.158344053191994
```

```
Bias: 0.41653984419151113
```

```
Neuron 2
```

```
Weight: -0.8808991350047904
```

```
Weight: -0.4701310277011403
```

```
Bias: 0.41653984419151113
```

```
Neuron 3
```

```
Weight: 1.5290215799042088
```

```
Weight: 1.4436446267547658
```

```
Bias: 0.41653984419151113
```

```
Neuron 4
```

```
Weight: -3.5567636242368232
```

```
Weight: -3.644574510733319
```

```
Bias: 0.41653984419151113
```

```
-----
* Output Layer
```

```
Neurons: 1
```

```
Neuron 0
```

```
Weight: -0.5112021657383754
```

```
Weight: 2.9727544893127495
```

```
Weight: -1.5850839575780384
```

```
Weight: 1.580813911542616
```

```
Weight: -8.958964245213592
```

```
Bias: 0.820742329903317
```

Figure 6: Error,Weights and Bias on Sample 10

Figure 7: Error,Weights and Bias on Sample 9990

The captured output graph Figure 5 indicates the amount of error minimized after training the data using backpropagation method. Notice the steep fall in the error as the number of iterations / training samples increases. The goal of backpropagation thus illustrates that a greater extent of correct data can train the model to achieve higher accuracy and minimize error.

References

- [1] *Probabilistic Machine Learning An Introduction*. The MIT Press, 2022.
- [2] 3Blue1Brown. But what is a neural network? <https://youtu.be/aircAruvnKk>.
- [3] Changyou Chen. 100 days of ml code. <https://github.com/cchangyou/simple-neural-network/blob/master/neural-network.py>.
- [4] Matt Mazur. A step by step backpropagation example. <https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>.
- [5] Michael Nielsen. Chapter 1 using neural nets to recognize handwritten digits. <http://neuralnetworksanddeeplearning.com/chap1.html>.
- [6] Wikipedia. Backpropagation. <https://en.wikipedia.org/wiki/Backpropagation>.

Naive Bayes Classifier Report

Darryl Vas Prabhu

Department of Computer Science and Engineering
University at Buffalo, Buffalo, NY 14260
dvasprab@buffalo.edu

1 Naive Bayes Classifier

¹ Classification models are seen quite often to model data which can be labelled. For instance classifying whether a song belongs to a particular genre, forgery analysis, tumour detection and so on. However it becomes essential that certain approximations are made in order to classify data. One of the classifier models which makes certain assumptions in order to classify the data is the Naive Bayes classifier.

As the name suggests the Naive Bayes classifier is based on Bayes theorem which goes by the notion that given some condition, we can use it to calculate the posterior probability of an event. The formula is given in the image Figure 1

Naive Bayes classifier as the name suggests makes use of 2 assumptions: The first is that the data or features in the dataset are independent of each other. The second one is that each datapoint will equally contribute to the outcome of the label. This means that if there are 2 features are present in the dataset, it is assumed that the features are in-dependant to ease the calculation. However they may not actually be independent. The classifier regardless of this aspect classifies the data into the right class label, despite the "naive" assumption and is hence called Naive Bayes classifier.

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

Figure 1: Bayes Theorem

Since each input feature is important here, Bayes theorem is used to calculate the posterior we find the probability of given set of inputs for all possible values of the class variable y and pick output with maximum probability. Once this is done $P(y)$ and $P(x|y)$ needs to be computed. This is usually computed by the observation in the training data and counting the values of the inputs which give the desired output class and changing the counts to likelihood values. Once they are obtained the Naive Bayes equation is used to calculate the posterior probability of each output class for the given set of input records. The maximum probability yielding class is selected as the classified values.

There can be a case where a feature would not influence the output class label in which case its probability for the output class would be 0. In this case, using the Naive Bayes equation would yield result 0 when used in the calculation. In order to avoid this zero frequency problem a 1 is added to all the input variables such that the probability does not approach 0 for any feature. Adding this one does not influence the result much since this is added to all the input features.

A sample example of the Bayes theorem is given below in figure 2

¹Naive Bayes Classifier.

2 Experiments

To illustrate the application of Naive Bayes Theorem a dataset for mushroom was selected from [9] . The dataset has too many attributes and for ease of understanding only certain attributes were picked from here.

2.1 Description of the dataset

The input variables and their values are the first 3 attributes or features of the dataset and the last one which is edible or poisonous is the output or class variable.

- cap-surface: brous, grooves, scaly, smooth
- bruises: bruises, no
- population: abundant, clustered, numerous, scattered, several, solitary
- Output : classes: edible, poisonous

Bayes Theorem: An Example

$$p(\theta|D) = \frac{p(D, \theta)}{p(D)} = \frac{p(\theta)p(D|\theta)}{\int p(\theta)p(D|\theta)d\theta} = \frac{p(\theta)p(D|\theta)}{p(D)}$$

- Solution: use Bayes theorem

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

- We know

$$P(X|A) = \left(\frac{1}{3}\right)^4 \left(\frac{2}{3}\right)^6, \quad P(X|B) = \left(\frac{1}{3}\right)^6 \left(\frac{2}{3}\right)^4$$

$$\rightarrow P(A|X) = \frac{4}{5}, \quad P(B|X) = \frac{1}{5}$$

Sampling from urn A is more likely

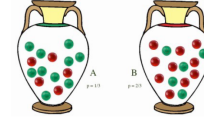


Figure 2: Naive Bayes Classifier example

```
def preprocess():
    # Using ordinal encoder to convert the categories in the range of 0 to n-1
    # cap_shape_enc = OrdinalEncoder()
    # cap_shape_ = cap_shape_enc.fit_transform(cap_shape)

    cap_surface_enc = OrdinalEncoder()
    cap_surface_ = cap_surface_enc.fit_transform(cap_surface)

    # cap_color_enc = OrdinalEncoder()
    # cap_color_ = cap_color_enc.fit_transform(cap_color)

    bruises_enc = OrdinalEncoder()
    bruises_ = bruises_enc.fit_transform(bruises)

    # odor_enc = OrdinalEncoder()
    # odor_ = odor_enc.fit_transform(odor)

    population_enc = OrdinalEncoder()
    population_ = population_enc.fit_transform(population)

    # habitat_enc = OrdinalEncoder()
    # habitat_ = habitat_enc.fit_transform(habitat)

    # Stacking all the features
    X = np.column_stack((cap_surface_, bruises_, population_))
```

Figure 3: Categorical data changed to numerical values

Various preprocessing steps are involved with Naive Bayes categorical data classification. We make use of the ordinal-encoder from sklearn's preprocessing package to change the categorical data into data in the range 0 to n-1 categories, which

here is 0 to 2 since we have 3 categories of input. The data is also checked for missing values in order to avoid any classification using .isna().

For each feature, all the values pertaining to that feature are fetched and all the data-points for each class are counted. The count of the different categories belonging to the feature is recorded and stored in the count_matrix. The count_matrix represents the output for each feature as a 2 dimensional array. One for the class edible and the other for poisonous.

The likelihood probabilities are then calculated and transformed to log probabilities and stored in log_probs.

Next the posterior probability is predicted by adding all the log probabilities for a feature as it is fetched from the log_probs table and the maximum posterior probability is returned.

An illustration of the same shows that using manual python processing and the inbuilt sklearn CategoricalNB classifier which is the Naive Bayes classifier for categorical features, the same value is retrieved as shown below.

```
log_probs = calculate_likelihood_probs(count_matrix,1,n_features)

log_probs

[array([[ -0.99261095, -8.34569287, -1.0291447 , -1.30253296],
        [-1.63921357, -6.66440902, -0.81163199, -1.02037655]]),
       array([[ -0.42477142, -1.06108312],
        [-1.83558495, -0.17378232]]),
       array([[ -2.39292426, -2.67974091, -2.35220617, -1.56510997, -1.261
94117,
        -1.37543752],
        [-8.27435701, -4.30406509, -8.27435701, -2.36356036, -0.319
63367,
        -1.79892429]])]
```

Figure 4: Log probabilities

```

from sklearn.naive_bayes import CategoricalNB
clf = CategoricalNB()
clf.fit(X, df['class'])
print('Sklearn feature log-probabilities\n',clf.feature_log_prob_)
print('Manually implemented likelihood probabilities\n',log_probs)

print('Sklearn feature prior-probabilities\n',clf.class_log_prior_)
print('Manually implemented prior probabilities\n',prior_probs)

print()
print('Sklearn predict',clf.predict(X[4:5]))
print('Manual predict',predict(X[3],log_probs,prior_probs))

```

Sklearn feature log-probabilities
 [array([[-0.99261095, -8.34569287, -1.0291447 , -1.30253296],
 [-1.63921357, -6.66440902, -0.81163199, -1.02037655]]), array([[-0.4
 2477142, -1.06108312],
 [-1.83558495, -0.17378232]]), array([[-2.39292426, -2.67974091, -2.3
 5220617, -1.56510997, -1.26194117,
 -1.37543752],
 [-8.27435701, -4.30406509, -8.27435701, -2.36356036, -0.31963367,
 -1.79892429]])]

Manually implemented likelihood probabilities
 [array([[-0.99261095, -8.34569287, -1.0291447 , -1.30253296],
 [-1.63921357, -6.66440902, -0.81163199, -1.02037655]]), array([[-0.4
 2477142, -1.06108312],
 [-1.83558495, -0.17378232]]), array([[-2.39292426, -2.67974091, -2.3
 5220617, -1.56510997, -1.26194117,
 -1.37543752],
 [-8.27435701, -4.30406509, -8.27435701, -2.36356036, -0.31963367,
 -1.79892429]])]

Sklearn feature prior-probabilities
 [-0.65783517 -0.72975192]

Manually implemented prior probabilities
 [-0.65783517 -0.72975192]

Sklearn predict ['edible']
 Manual predict edible

Figure 5: CategoricalNB and manual Naive Bayes Classifier

References

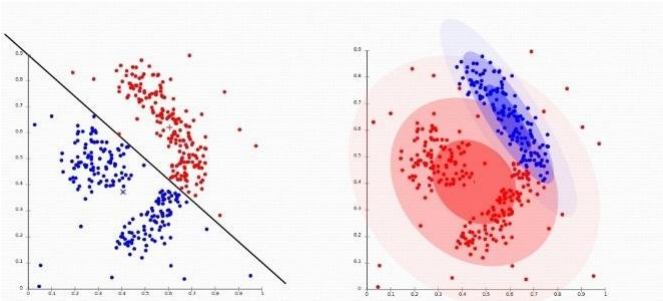
- [1] Numpy documentation. <https://numpy.org/doc/stable/reference/>,.
- [2] Pandas documentation. <https://pandas.pydata.org/docs/>,.
- [3] Scikit learn. https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.CategoricalNB.html,.
- [4] *Probabilistic Machine Learning An Introduction*. The MIT Press, 2022.
- [5] Changyou Chen. Bayesian methods.
- [6] Changyou Chen. Categorical naive bayes from scratch in python. <https://www.kaggle.com/code/gautigadu091/categorical-naive-bayes-from-scratch-in-python/notebook>.
- [7] Nik datagy. One hot encoding. <https://datagy.io/sklearn-one-hot-encode/>.
- [8] GeekforGeeks. [Geekforgeeks. https://www.geeksforgeeks.org/naive-bayes-classifiers/](https://www.geeksforgeeks.org/naive-bayes-classifiers/).
- [9] Kaggle. [Kaggle dataset. https://www.kaggle.com/code/upadorprofzs/bayesian-learning-basics-tutorial/data](https://www.kaggle.com/code/upadorprofzs/bayesian-learning-basics-tutorial/data).
- [10] KDnuggets Nagesh Singh Chauhan. Naïve bayes algorithm: Everything you need to know. <https://www.kdnuggets.com/2020/06/naive-bayes-algorithm-everything.html>.

Gaussian Mixture Model Report

Darryl Vas Prabhu

Department of Computer Science and Engineering
University at Buffalo, Buffalo, NY 14260
dvasprab@buffalo.edu

1 Gaussian Mixture Model

¹ Clustering of data is a unsupervised method used in machine learning in order to "cluster" data-points based on the similarities between the them. Groups of similar data are clustered together based on their similarities. There are various clustering techniques such as k-means clustering which defines which class a particular data-point belongs to. However in k-means we make use of the concept of hard cluster, where the data either belongs to a cluster or not. Now what if the data-set given was random with multiple points and features such that the a data-points had similarities and belonged to multiple clusters? That is to say the clusters defined here overlap. In such a case ,using a hard cluster which merely classifies points as 'belonging' or 'not belonging' becomes inefficient and is prone to error. When the clusters overlap, clustering the data into it's right cluster cannot be done with traditional method. A stronger method known as soft clustering needs to be employed which is the basis for Gaussian Mixture models. An image 1 of clustering is shown where the left sub-image corresponds to data classified based on hard clustering and the right sub-image one denotes the cluster overlapping data point.

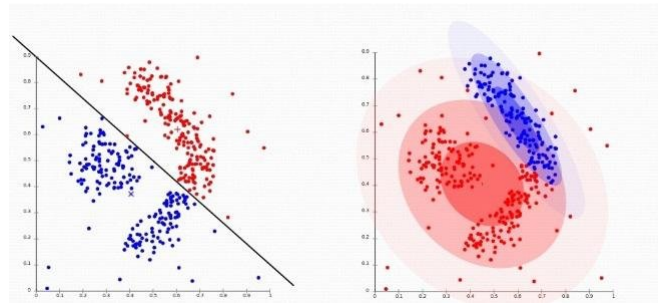


Figure 1: Hard Clustering Vs Soft Clustering

So how do we cluster the data in soft clustering. Each cluster is defined to be a probability density function in a d-dimensional space and the points are the samples which make of the distribution. Turns out that most real world data can be represented as Gaussian distribution function. The Gaussian distribution is a bell shaped curve where most data points are centered around the mean. Each cluster is defined to be a Gaussian distribution of it's data-points, i.e the data points are samples of the probability distribution function. Say for example we have a set of data-points, and we want to know the source i.e which Gaussian it belongs to. Then we need the mean and standard deviation. From this we can use Bayes rule to find out the posterior probability of the point belonging to a particular class. However the mean and variance are unknown at this point as well. The problem of having both mean and variance unknown as well as the undetermined Gaussian curve the data point belongs to results in a chicken and egg problem.

¹Gaussian Mixture Model

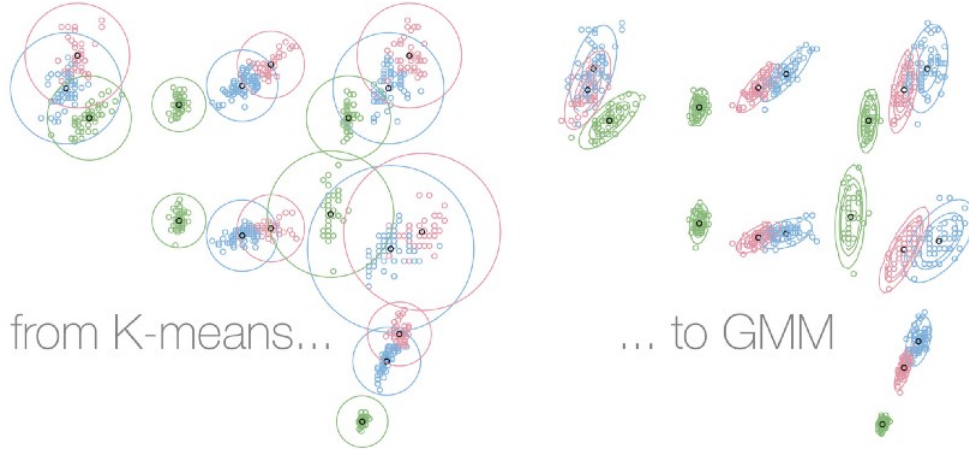


Figure 2: K-means to GMM

In order to solve this problem, Expectation Maximization algorithm is used where we randomly select the mean and variance for the Gaussian curve of a particular class the data-point belongs to. We then compute for each point the probability that it belongs to the Gaussian curve. The mathematical formula includes finding out how different Gaussian component attributes correlate using the Covariance matrix. After this step Bayes rule is applied to convert the probability to posterior probability. Certain standard schemes such as assigning weight equal to the posterior divide by the summation of all the posterior is used to recompute the new mean and Covariance matrix. The prior probabilities of each class is re-calculated if needed to improve efficiency. The process is repeated with the new mean and covariance matrix, until it converges at local optimum value. The summary is on the right side in image 3. The EM algorithm described mathematically in the image 4 below.

Gaussian Mixture Model

- Data with D attributes, from Gaussian sources $c_1 \dots c_k$
 - how typical is \mathbf{x}_i under source c : $P(\bar{\mathbf{x}}_i | c) = \frac{1}{\sqrt{2\pi|\Sigma_c|}} \exp\left[-\frac{1}{2}(\bar{\mathbf{x}}_i - \bar{\mu}_c)^T \Sigma_c^{-1} (\bar{\mathbf{x}}_i - \bar{\mu}_c)\right]$
 - how likely that \mathbf{x}_i came from c : $P(c | \bar{\mathbf{x}}_i) = \frac{P(\bar{\mathbf{x}}_i | c)P(c)}{\sum_{c'} P(\bar{\mathbf{x}}_i | c')P(c')}$
 - how important is \mathbf{x}_i for source c : $w_{ic} = P(c | \bar{\mathbf{x}}_i) / (P(c | \bar{\mathbf{x}}_i) + \dots + P(c | \bar{\mathbf{x}}_i))$
 - mean of attribute \mathbf{a} in items assigned to c : $\mu_{ic} = w_{ic}x_{ia} + \dots + w_{ic}x_{im}$
 - covariance of \mathbf{a} and \mathbf{b} in items from c : $\Sigma_{ic} = \sum_{i=1}^N w_{ic} (x_{ia} - \mu_{ic})(x_{ib} - \mu_{ic})$
 - prior: how many items assigned to c : $P(c) = \frac{1}{N} (P(c | \bar{\mathbf{x}}_1) + \dots + P(c | \bar{\mathbf{x}}_N))$

Figure 3: GMM analysis

EM Algorithm for GMM

- **Initialize** the means μ_k , covariances Σ_k and mixing coefficients π_k
- **Iterate until convergence:**
 - ▶ **E-step:** Evaluate the responsibilities given current parameters

$$\gamma_k^{(n)} = p(z^{(n)} | \mathbf{x}) = \frac{\pi_k \mathcal{N}(\mathbf{x}^{(n)} | \mu_k, \Sigma_k)}{\sum_{j=1}^K \pi_j \mathcal{N}(\mathbf{x}^{(n)} | \mu_j, \Sigma_j)}$$

- ▶ **M-step:** Re-estimate the parameters given current responsibilities

$$\mu_k = \frac{1}{N_k} \sum_{n=1}^N \gamma_k^{(n)} \mathbf{x}^{(n)}$$

$$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^N \gamma_k^{(n)} (\mathbf{x}^{(n)} - \mu_k)(\mathbf{x}^{(n)} - \mu_k)^T$$

$$\pi_k = \frac{N_k}{N} \quad \text{with} \quad N_k = \sum_{n=1}^N \gamma_k^{(n)}$$

- ▶ Evaluate log likelihood and check for convergence

$$\ln p(\mathbf{X} | \pi, \mu, \Sigma) = \sum_{n=1}^N \ln \left(\sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}^{(n)} | \mu_k, \Sigma_k) \right)$$

Figure 4: Expectation Maximization algorithm

2 Experiments

To order to understand the GMM model, the code from was used to determine the Gaussian curve for the various components of the Iris data-set and data-points in each class. This shows the variation of the Sepal-length of the Iris flowers with the classes they are part of.

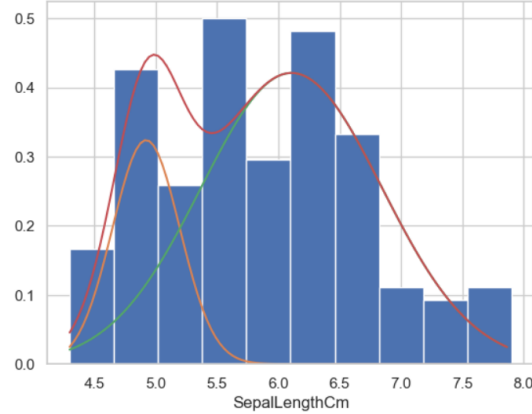


Figure 5: GMM output for Iris Sepal-Length perspective

References

- [1] Different types of convergence for sequences of random variables. https://www.probabilitycourse.com/chapter7/7_2_3_different_types_of_convergence_for_sequences_of_random_variables.
- [2] *Probabilistic Machine Learning An Introduction*. The MIT Press, 2022.
- [3] Changyou Chen. Gaussian mixture model. <https://github.com/cchangyou/Gaussian-Mixture-Model/blob/master/main.py>.
- [4] Youtube:Victor Lavrenko. Expectation maximization: how it works. <https://youtu.be/iQoXFmbXRJA>.
- [5] Arnaud M. Mixture modelling from scratch:towards data science. <https://towardsdatascience.com/mixture-modelling-from-scratch-in-r-5ab7bfc83eef>.
- [6] Farhad Malik : Medium. Machine learning hard vs soft clustering. <https://medium.com/fintechexplained/machine-learning-hard-vs-soft-clustering-dc92710936af>.
- [7] Ahmed Elyamany : Researchgate. Hard clustering fuzzy clustering. https://www.researchgate.net/figure/Figure-3-Hard-clustering-Fuzzy-clustering_fig2_337306741.