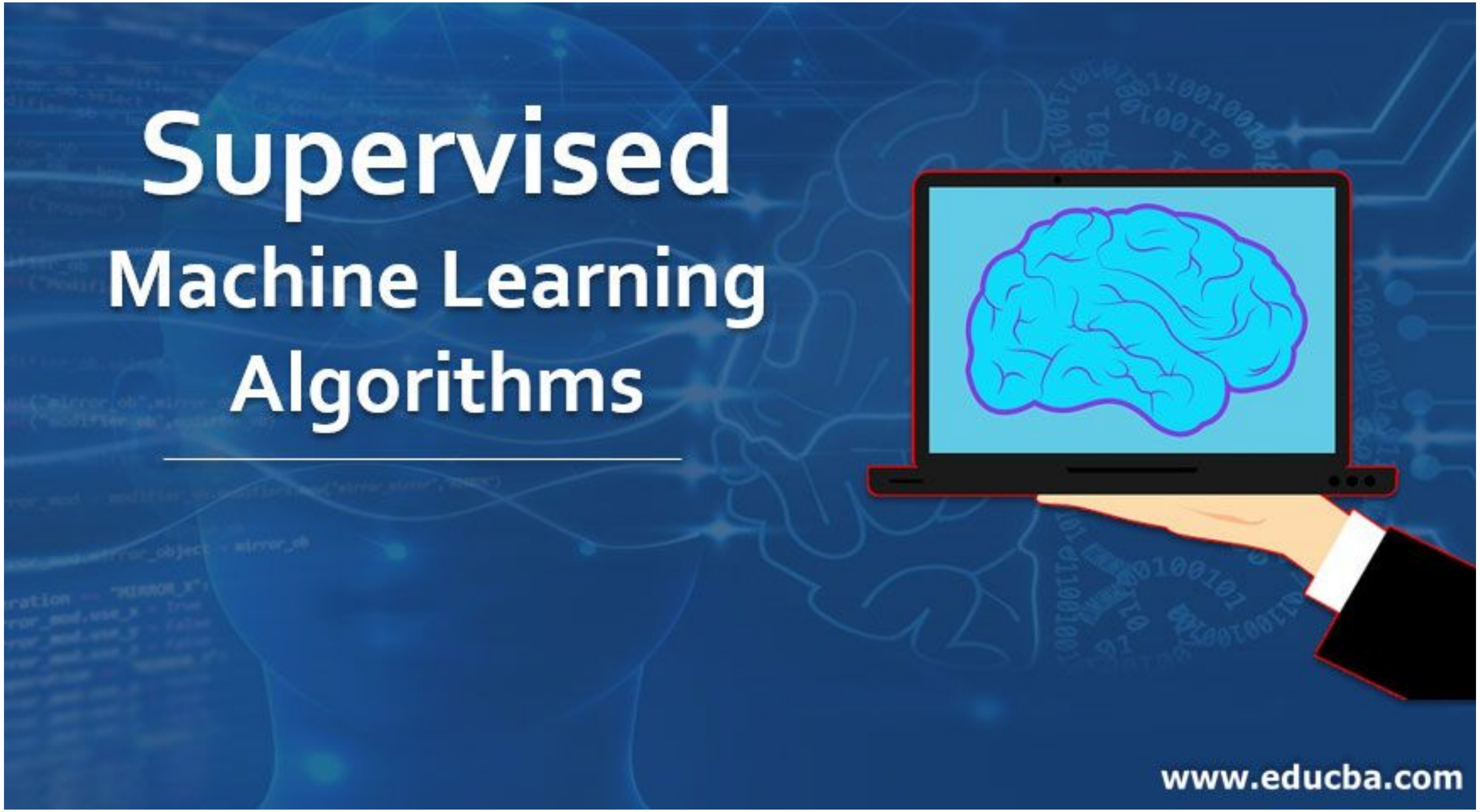

Supervised
Machine Learning
Algorithms
www.educba.com

Last lecture reminder

We learned about:

- Logistic Regression - Multiple categories
- Logistic Regression with multiple categories - Python example
- KNN (K-Nearest Neighbors) algorithm - Introduction
- KNN - Dealing with tie
- KNN - Simple Python example (K = 1)

KNN - Choosing K Value

So in the previous lecture we saw a simple example for using the KNN algorithm when K = 1.
We already discussed that the number of K determine how many nearest neighbors are going to be involved in the prediction process. So choosing the right value of K is very important in KNN.

How can we choose the right value for K?

- **Elbow method** → We train our model on different values of K and chose the one that minimize the model error.
  Our model error formula will be: **Error = 1 - Accuracy**
  (accuracy is the percentage of observations that our model predict correctly).
  Once we have all error values for each K value we can plot it and chose the K value that minimize this error.

KNN - Choosing K Value

How can we choose the right value for K?

- **Cross Validation** → Similar with the Elbow method, we will run the KNN algorithm for different values of K and select the one that produces the best prediction accuracy.

What is the difference between Elbow Method and Cross Validation?

The two methods, Cross-validation and the Elbow method, aim to optimize the hyperparameter K in KNN. However, they differ in how they evaluate the performance for these different K values:

- **Elbow method** is a more graphical and less computationally intensive strategy that tests the KNN model's performance for various K values.
- **Cross Validation** will also divide our dataset into k-fold (multiple data splits) and evaluate for each of them the best K value. So Cross Validation find the optimize K value in addition to fully utilize the dataset.

KNN Elbow Method - Python Example

Let’s see how to execute the Elbow Method on KNN model to select the optimal K value.

For the example we will use the **‘gene_expression.csv’** file from the previous lecture.

First, let’s prepare our data for the model training:

```
In [26]: from sklearn.model_selection import train_test_split
         from sklearn.preprocessing import StandardScaler
         from sklearn.neighbors import KNeighborsClassifier

         df = pd.read_csv('/Users/ben.meir/Downloads/gene_expression.csv')
         X = df.drop('Cancer Present', axis=1)
         y = df['Cancer Present']

         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

         scaler = StandardScaler()
         scaled_X_train = scaler.fit_transform(X_train)
         scaled_X_test = scaler.transform(X_test)
```

KNN Elbow Method - Python Example

Now let's create a for loop that will train our model on every K value between 1 - 30 and calculate the error value for each K:

```
In [27]: from sklearn.metrics import accuracy_score

         test_error_rates = []

         for k in range(1,30):
             knn_model = KNeighborsClassifier(n_neighbors=k)
             knn_model.fit(scaled_X_train, y_train)

             y_pred_test = knn_model.predict(scaled_X_test)
             test_error = 1 - accuracy_score(y_test, y_pred_test)
             test_error_rates.append(test_error)
```

For each K value we train our model accordingly and save the error rate (1 - accuracy_score) in a dedicated array

We can take a look at the array we created and see that higher K value is producing more accurate model predictions.

```
In [28]: test_error_rates

Out[28]: [0.10555555555555551,
          0.09999999999999998,
          0.07444444444444442,
          0.07777777777777772,
          0.07222222222222219,
          0.06666666666666665,
          0.06333333333333335,
          0.05888888888888888,
          0.05777777777777782,
          0.06222222222222218,
```

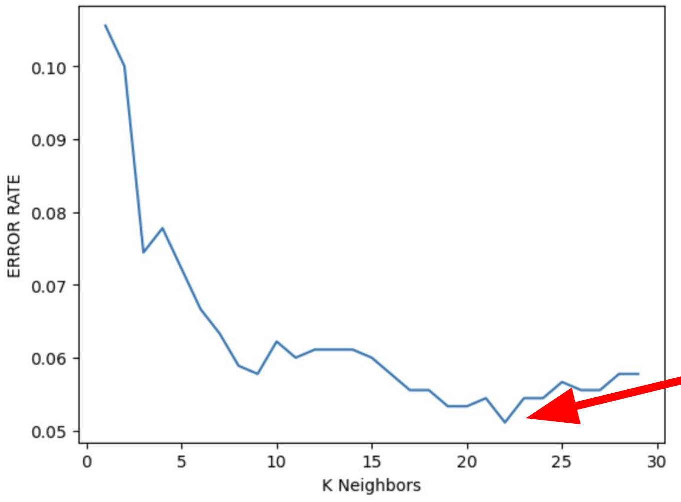
ECOM SCHOOL

המכללה למקצועות הדיגיטל וההייטק

KNN Elbow Method - Python Example

We can also plot our results so it will be visually more clear where is the optimal K value.

```
In [30]: plt.plot(range(1,30), test_error_rates)
         plt.ylabel('ERROR RATE')
         plt.xlabel('K Neighbors')
         plt.show()
```



We can see that for K = 23 we got the lowest error rate.

Note: It is very common to execute the Elbow Method and chose the K with the lowest error rate but remember that increasing the K value meaning increasing your model complexity, because you are considering more point for each prediction. <u>When increasing the K is not significantly increasing our model accuracy we should consider not to choose the optimal K automatically</u>.

KNN Cross Validation - Python Example

Let’s now execute Cross Validation on our KNN model in order to choose the optimal K value.

For this we will use the Scikit-Learn **cross_val_score()** method that we learned before.

```
In [36]: from sklearn.model_selection import cross_val_score

         cv_scores = []

         for k in range(1, 31):
             knn = KNeighborsClassifier(n_neighbors=k)
             scores = cross_val_score(knn, scaled_X_train, y_train, cv=10, scoring='accuracy')
             cv_scores.append(scores.mean())

         optimal_k = cv_scores.index(max(cv_scores)) + 1
         optimal_k

Out[36]: 29
```

We are performing Cross Validation with 10 K-folds on each K value and save the mean accuracy score of each 10 folds

The K value with the max accuracy score will be the optimal K value for our KNN model

Now let’s train our model on the optimal K value our Cross Validation has found:

```
In [37]: knn_optimal = KNeighborsClassifier(n_neighbors=optimal_k)
         knn_optimal.fit(scaled_X_train, y_train)
         y_pred = knn_optimal.predict(scaled_X_test)
```

We now train our model according to the optimal K value we found in CV

KNN Cross Validation - Python Example

Finally, let's print the model error metrics and evaluate our model performance.

```
In [38]: print(confusion_matrix(y_test, pred))
         print()
         print('Accuracy score: ',accuracy_score(y_test, pred))
         print()
         print(classification_report(y_test, pred))

[[447  23]
 [ 29 401]]

Accuracy score:  0.9422222222222222

              precision    recall  f1-score   support

           0       0.94      0.95      0.95       470
           1       0.95      0.93      0.94       430

    accuracy                           0.94       900
   macro avg       0.94      0.94      0.94       900
weighted avg       0.94      0.94      0.94       900
```

We can see that our model accuracy increased from 0.895 in K = 1 to 0.942 in K = 29

Remember, In CV we evaluation the optimal K value from multiple K-fold datasets and selecting the K that had the best accuracy score from all posibles K values.

ECOM SCHOOL
המכללה למקצועות הדיגיטל וההייטק

Class Exercise - KNN

Instructions:

For this exercise use the **‘iris.csv’** dataset. Your mission to predict the species of iris based on selected features. Use the KNN model in order to predict the iris species.

- Perform simple KNN with K = 1 and print your model accuracy, confusion matrix and classification report
- Perform Elbow Method to find the best K that optimize your model error value
  Plot your results for better visualization
  Perform KNN model with the optimal K you found with the Elbow Method, print the model error metrics
- Perform Cross Validation to find the best K value with k-fold value of 5
  Print the optimal K value you found with Cross Validation
  Perform KNN model with the optimal K you found with the Elbow Method, print the model error metrics

Class Exercise Solution - KNN

Support Vector Machines - Introduction

Support Vector Machine (SVM) is a supervised machine learning algorithm widely used for classification and regression analysis. SVM is one of the more complex algorithm in supervised learning family.
At its core, SVM is a maximum-margin classifier, it aims to find the hyperplane (על-מישור) in an N-dimensional space (where N is the number of features) that distinctly classifies the data points.
The hyperplane is selected in a way to minimize error while maximizing the geometric margin between classes.

The hyperplane acts as a decision boundary. Once the SVM has been trained and the hyperplane is determined, the model makes predictions based on which side of the hyperplane the new data point falls.
For a simple two-class SVM:

1. If a data point falls on one side of the hyperplane, it is predicted to belong to the first class.
2. If it falls on the other side of the hyperplane, it is predicted to belong to the second class.

Support Vector Machines - Theory

In order to understand the logic behind SVM we first need to understand what is a hyperplane

Hyperplane (על מישור) → A hyperplane is a subspace in an n-dimensional space that is one dimension less than the space itself. In simple terms, it's a flat affine subspace which divides an n-dimensional space into two half-spaces.

For example →

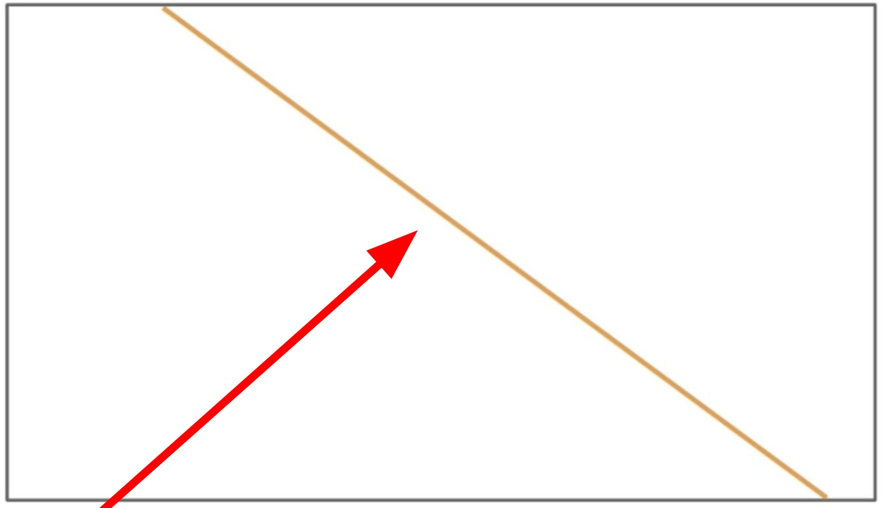
- In a 1D space (a line) a hyperplane is a 0D point that divides the line into two parts.
- In a 2D space (a plane) a hyperplane is a 1D straight line that divides the plane into two parts.
- In a 3D space, a hyperplane is a 2D plane that divides the space into two halves.
- In a 4D space, a hyperplane is a 3D space that divides the 4D space into two halves.

Support Vector Machines - Theory

Let’s take a look at the different hyperplane dimensions visualizations:

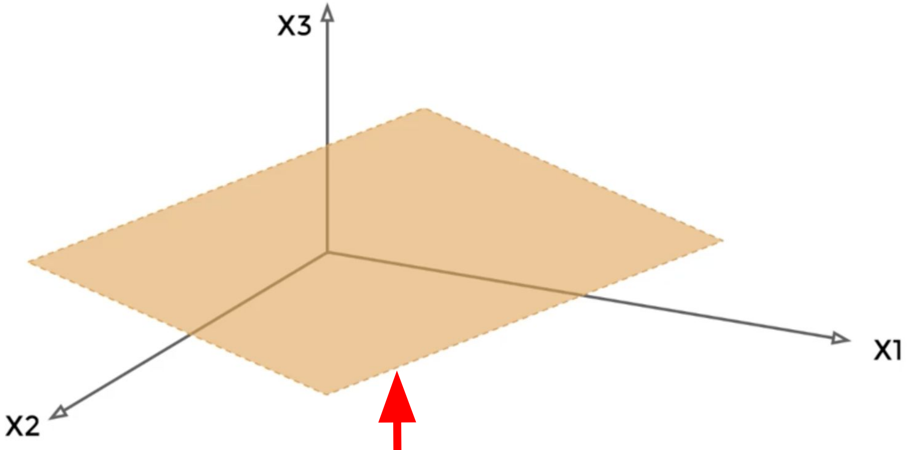
A point (0D) hyperplane for a 1-dimensional space

X2



X1

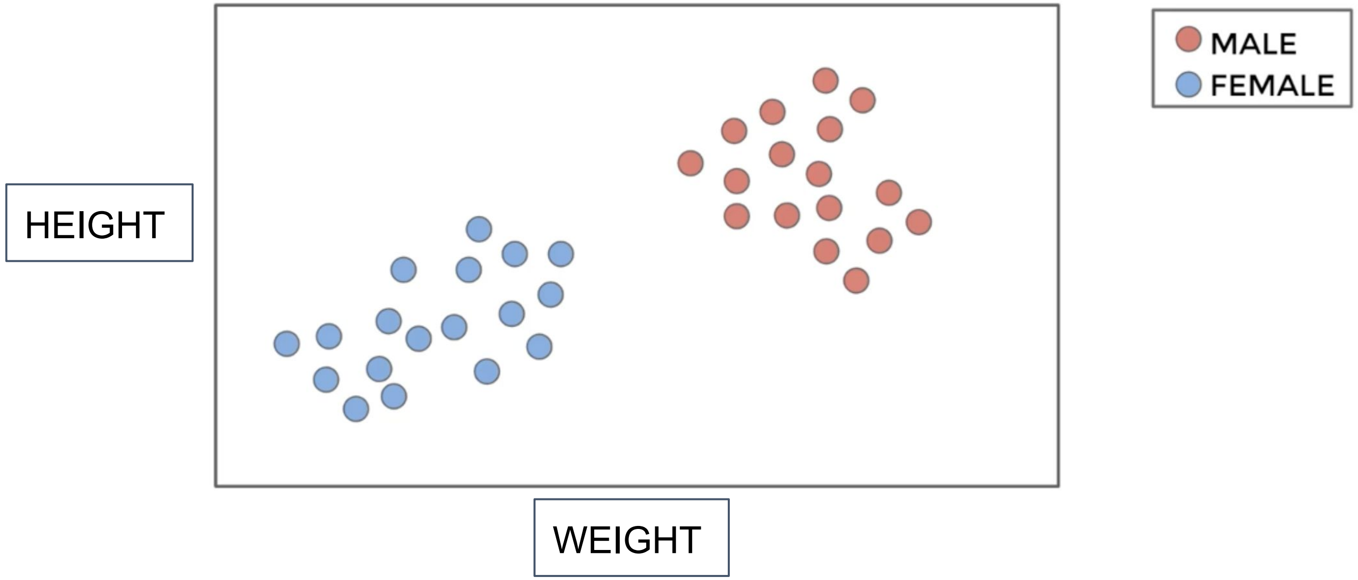
A line (1D) hyperplane for a 2-dimensional space



A plane (2D) hyperplane for a 3-dimensional space

Support Vector Machines - Theory

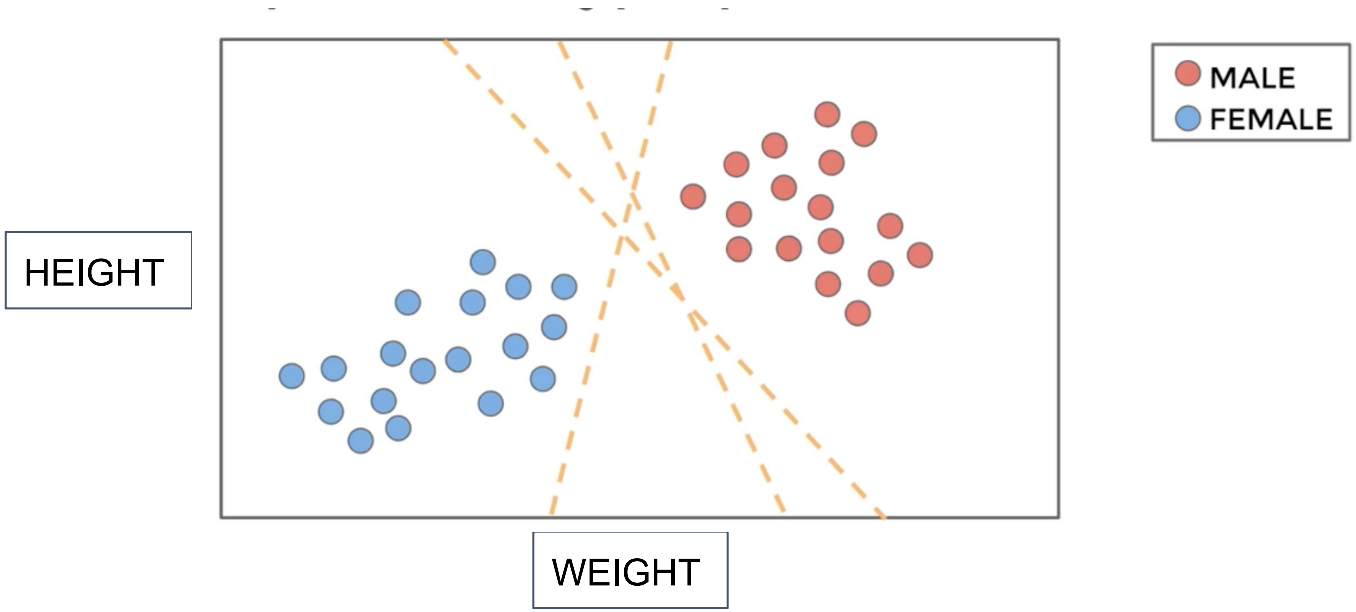
So if we will take, for example, our previous classification problem when we want to predict the animal sex type according to the animal height and weight, and let’s say our data scatter plot will look like this:



We can generate a <u>line hyperplane to separate between the male and female groups</u>, our model will use this hyperplane line in order to predict future observation accordingly.

Support Vector Machines - Theory

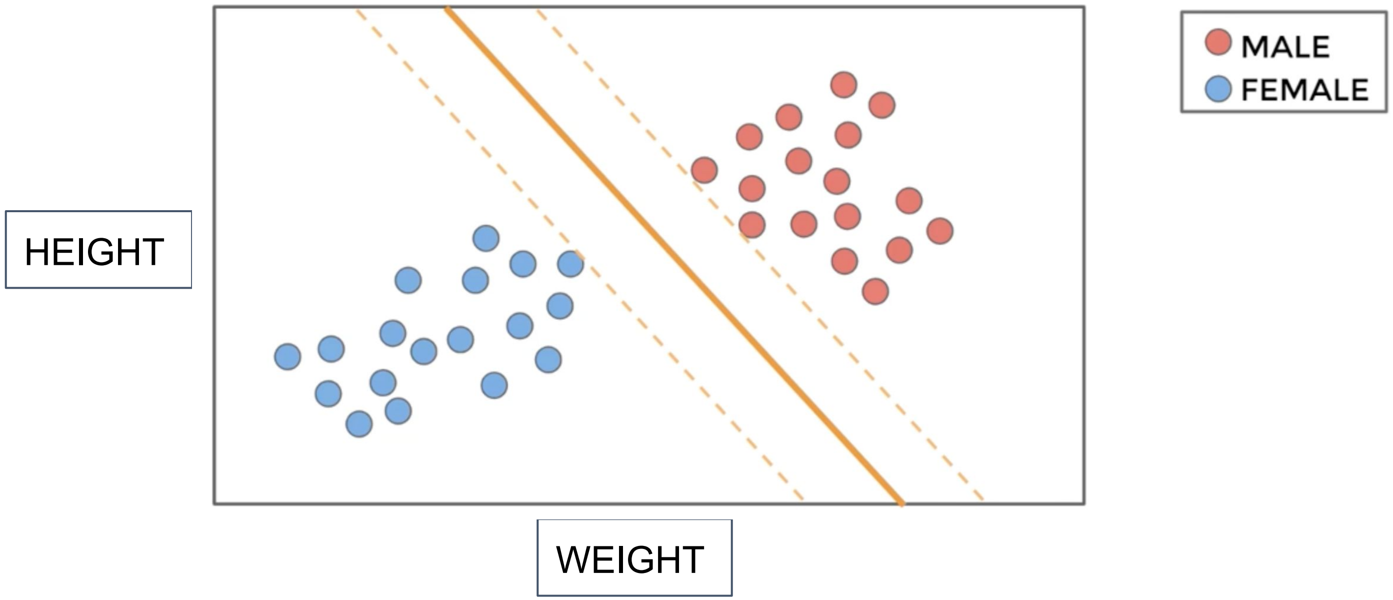
As shown, we can have multiple line options that will perfectly separated the two groups:



The way the SVM algorithm is choosing the hyperplane line, is by <u>calculating the hyperplane that maximizes the margin between the different classes that need to be separated</u>.

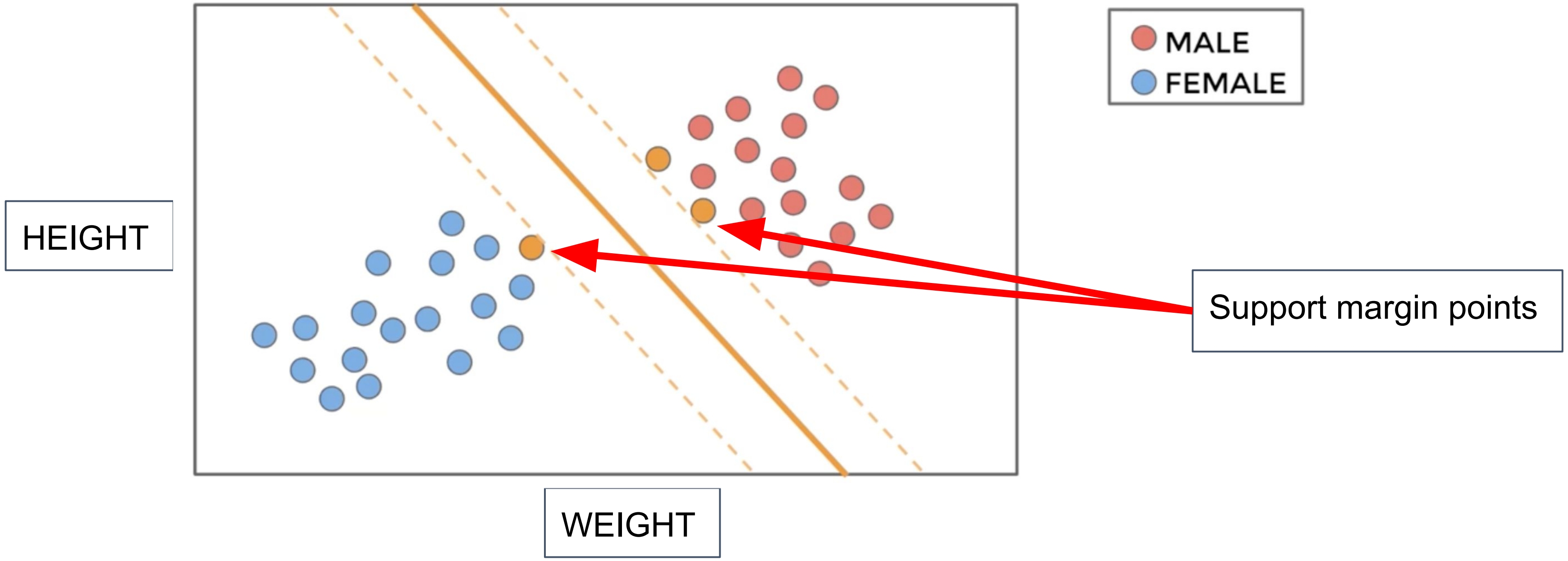
Support Vector Machines - Theory

So in our example, if we take the margin between the ‘Male’ and ‘Female’ groups, the selected hyperplane line should be the one that has the maximized distance between those two group points:



Support Vector Machines - Theory

Support vectors points → The support vectors points are the nearest training data points of each class. Those points create the margin between the data groups and the hyperplane that been selected inside this margin and been calculated as the hyperplane with the maximum distance to the nearest training data points of any class.



ECOM SCHOOL
המכללה למקצועות הדיגיטל וההייטק

Support Vector Machines - Theory

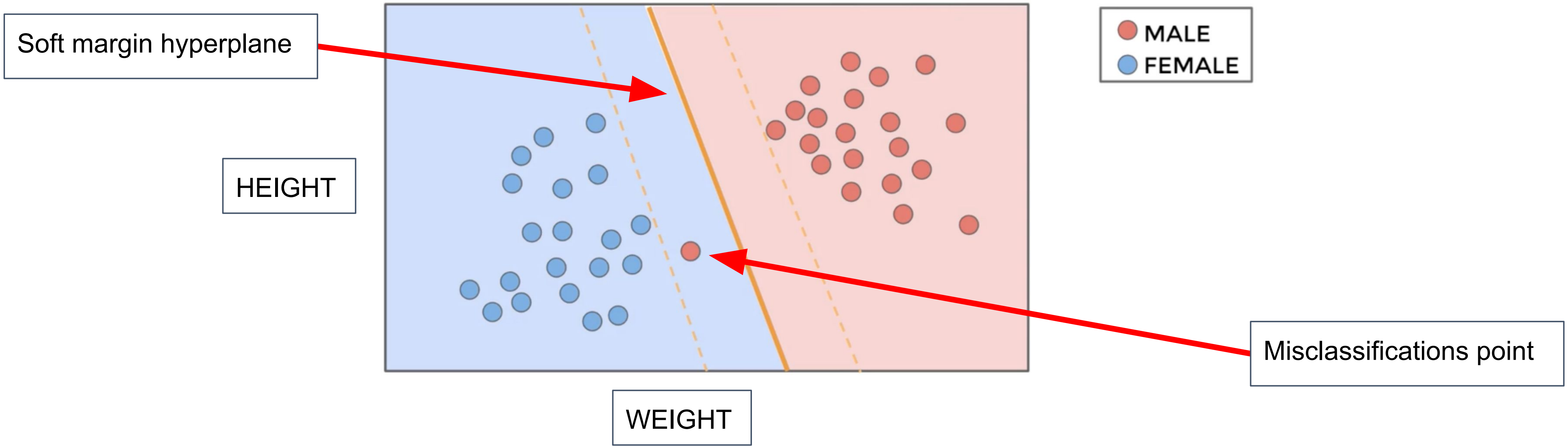
Unlike in our example, in most cases It is not always possible to find a hyperplane that perfectly separating the data groups. In order to solve this we need to allow soft margins.

Soft margins → Soft margins refer to allowing a certain degree of misclassification in order to achieve a better, more generalized model.
The concept of a soft margin allows for some data points to be on the "wrong" side of the hyperplane. As previously mentioned, enforcing a perfect separation when the data is not linearly separable could lead to overfitting. However, allowing a soft margin means the model can still generalize well to unseen data.

C parameter → The trade-off between the margin maximization and amount of misclassifications is regulated by a hyperparameter, typically denoted 'C'. A larger 'C' results in a smaller margin and fewer classification errors (a harder margin). A smaller 'C' results in a larger margin but more classification errors (a softer margin).

Support Vector Machines - Theory

In this chart we can see soft margin been applied:



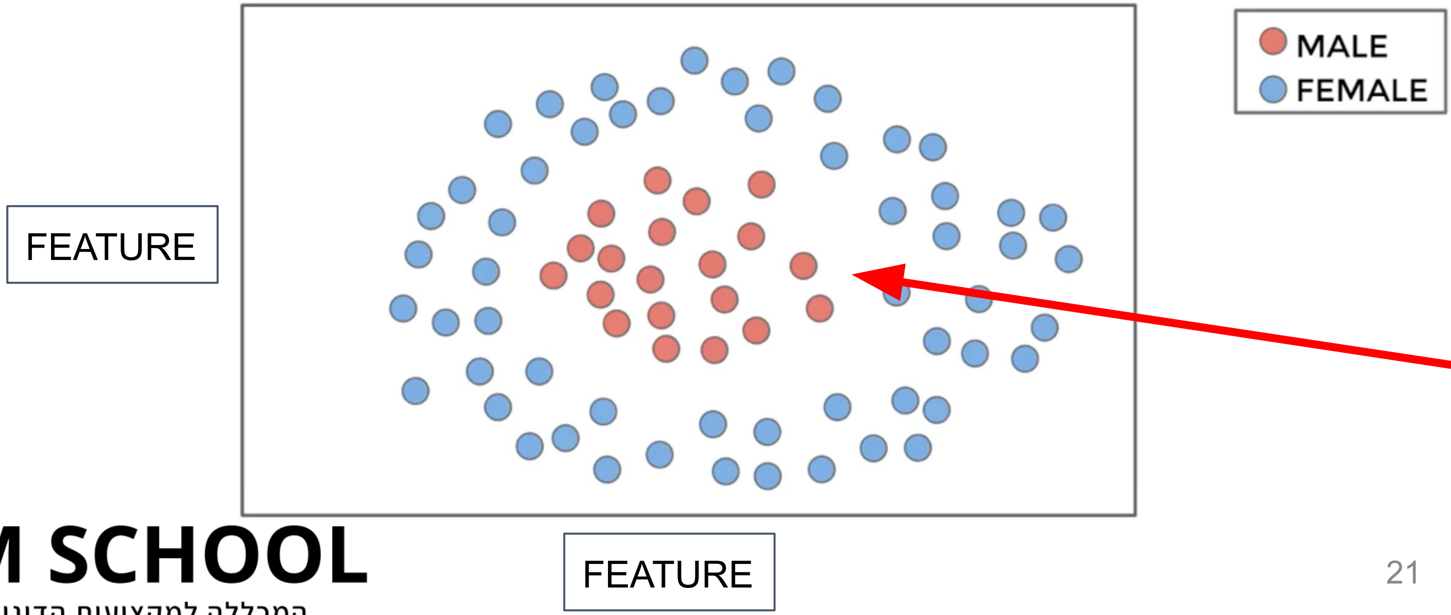
Support Vector Machines - Kernel Trick

Until now we saw examples for hard margin and soft margin using hyperplan separator.

In all the previous examples we easily managed to found a hyperplane that can separate most of the data correctly between the classes.

However, <u>we can also have data points that are not that easy to separate using a hyperplane</u>.

For example → Let’s take a look at those data points:



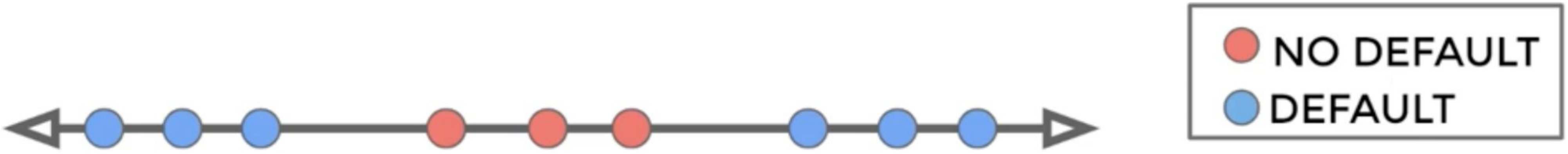
We can see that there is no straight line that will separate the two classes

Support Vector Machines - Kernel Trick

Kernel → A kernel in machine learning is a function that takes two inputs and outputs the similarity between the two. It is used in a variety of machine learning algorithms to handle high-dimensional data.

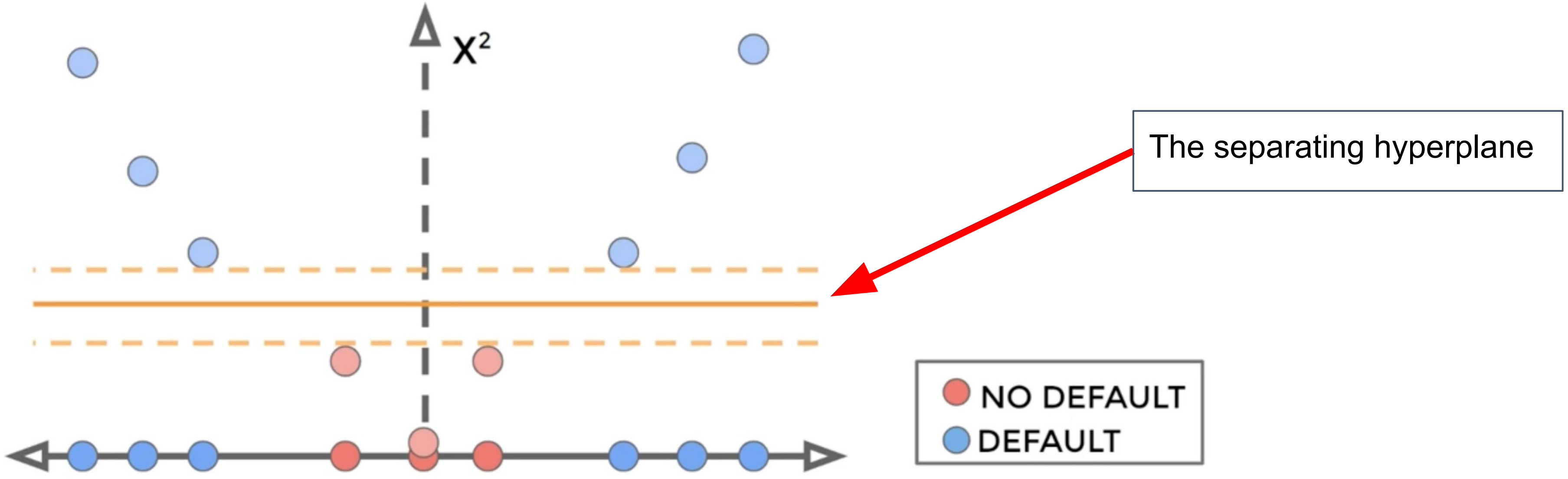
Kernel trick → The kernel trick allowing us to move our data points to higher dimension in order to find a hyperplane that can separate that data into the different classes.
When applying the kernel trick we project each data point in the higher dimensional space and calculate its position in that space using **kernel function**.

For example → Let’s say we have the following data points in the 1D space.
Clearly we can’t find any hyperplane (in 1D space - dot) that will separate the data points into 2 groups.



Support Vector Machines - Kernel Trick

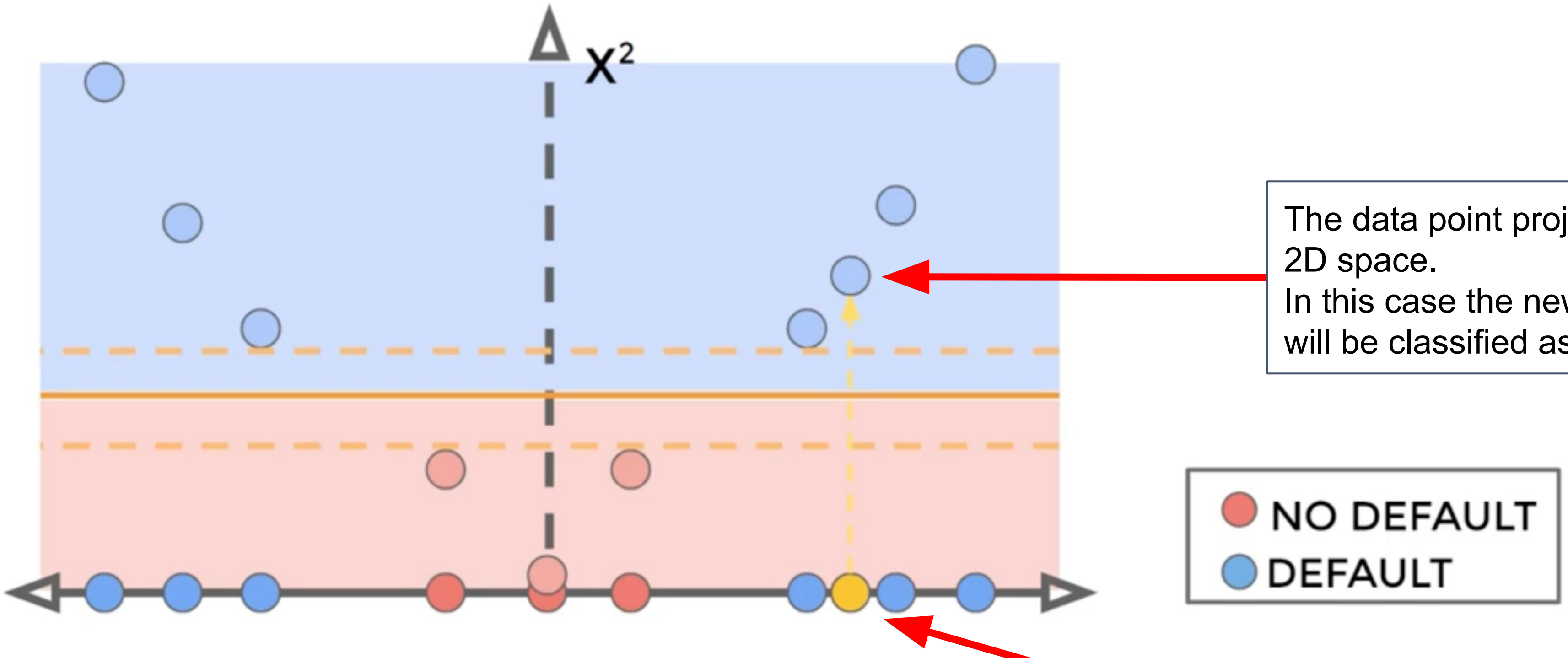
What we can do is project those data point into a 2D space and try to find a hyperplane (in 2D space - line) that will separate them correctly. For this we will use the polynomial kernel meaning for each X data point we will calculate it's x^2 value as the y and place them in 2D axis system.



When moving to higher space (2D) we can now find a line that will separate the projections of the data points between the classes.

Support Vector Machines - Kernel Trick

When we want to predict a class to a new data point, we will need to first project it to our new dimensional space and classify the data point according to the separating hyperplane.



The data point projection in the 2D space.
In this case the new data point will be classified as 'DEFAULT'

New data point to classify

Support Vector Machines - Kernel Trick

In the same way we can project data points in the 2D space to the 3D space and find the separator hyperplane (in 3D space - plane):