DATA SOCIETY:

Advanced Classification - Part 5

One should look for what is and not what he thinks should be. (Albert Einstein)

Module completion checklist

Objective	Complete
Introduce the concept of a hyperplane and classification using a hyperplane	
Summarize the idea of maximal margin classifier and its pitfalls	
Explain the concept of support vectors and build a support vector classifier model	

Support Vector Machines

- We have studied ensemble methods like Random Forest and Gradient Boosting Models for classification
- Every dataset is different and no one classification algorithm is a best fit for all types of datasets
- What if you have **high dimensional data** at work and you have to classify a target variable?
- What if your dataset contains more **categorical variables?**
- Today, we will learn about **Support vector machines:**
  - a machine learning algorithm which was developed to handle such issues in our dataset

Concept behind different classifiers

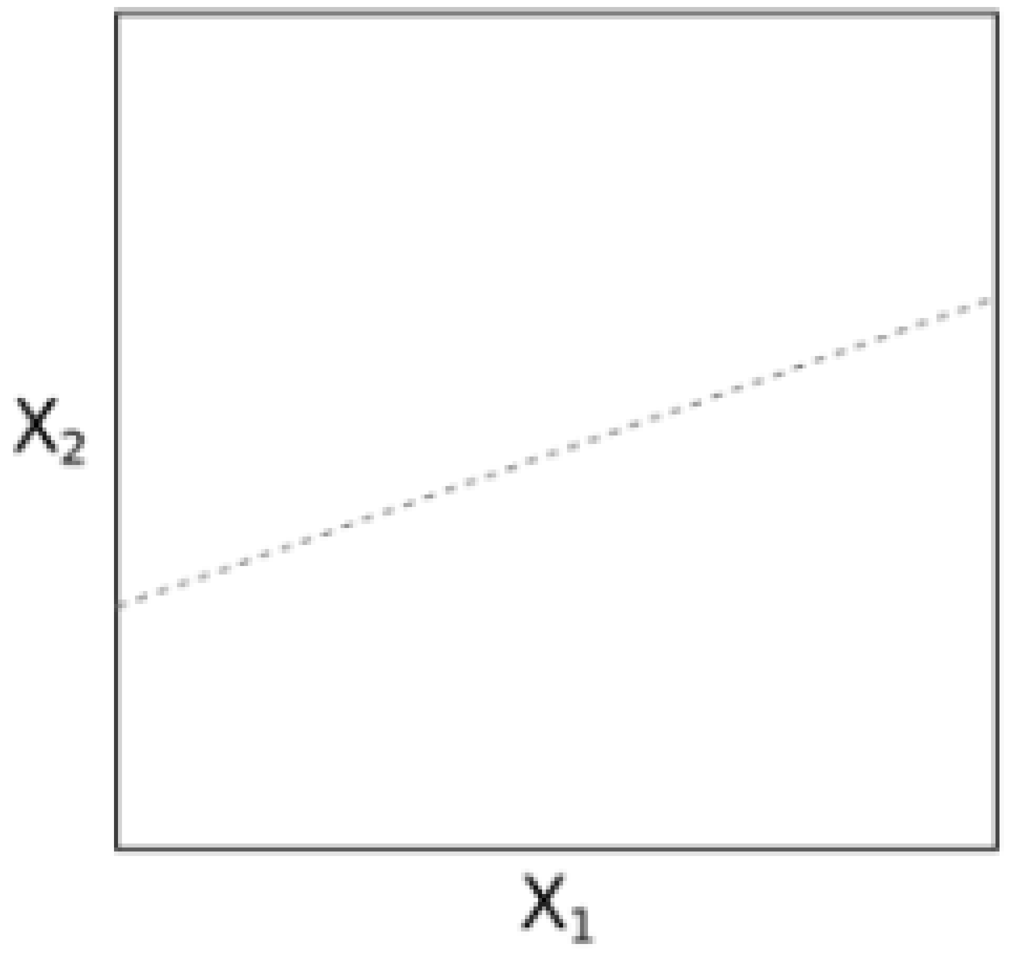
- We studied many classification algorithms so far
  - Trees and ensemble methods classify based on the value of predictors using the idea of **segmentation**
- The model we are going to study today makes use of a **hyperplane** to classify the observations

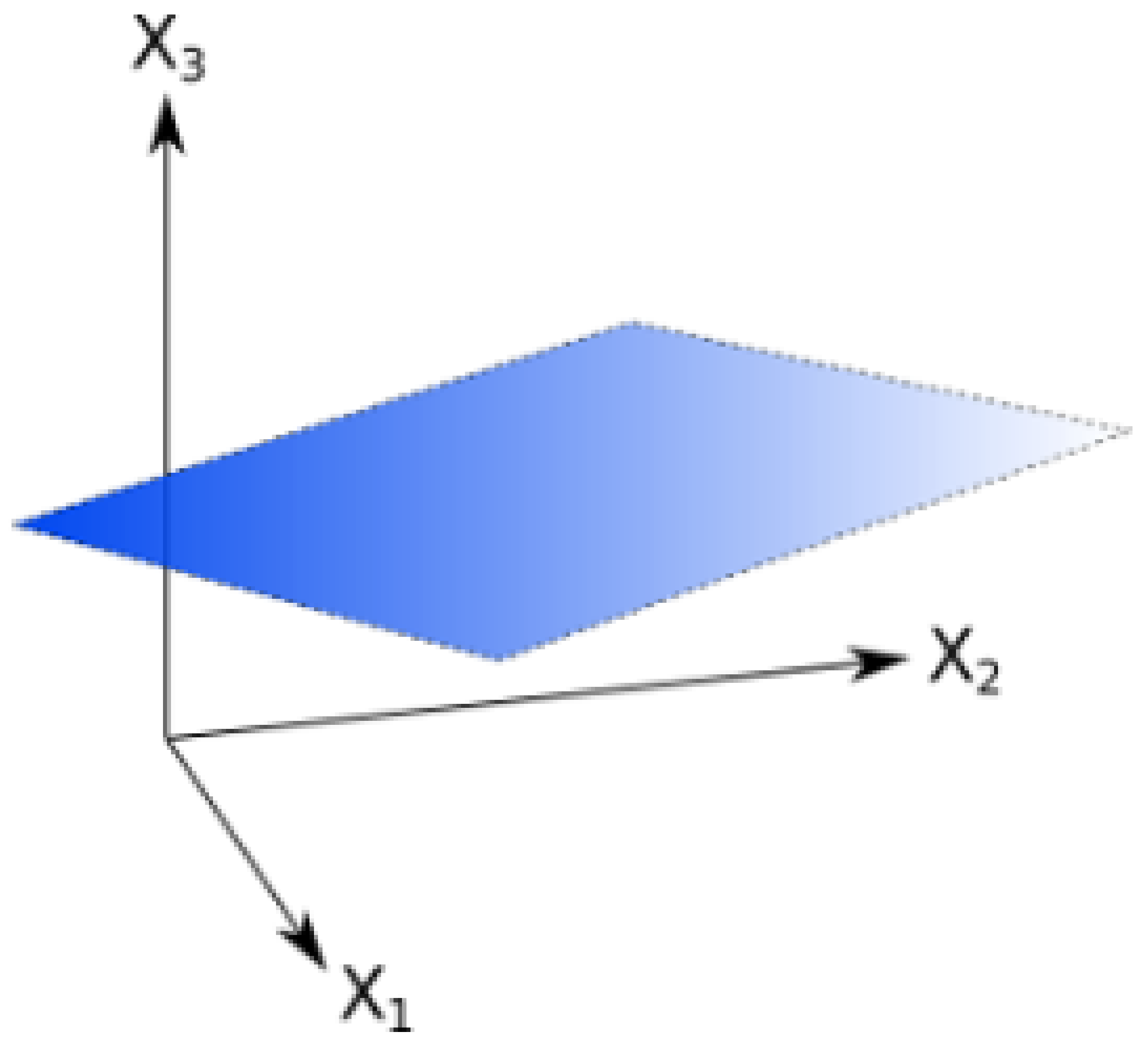
Idea behind SVM as a classifier

- **Support vector machines** build the model by creating a **feature space** which is a finite dimensional vector space
- Each dimension represents the features which we are using to predict the target
- SVM creates a **hyperplane which creates a linear partition of the feature space into two categories**
- The target is classified into two categories based on whether the features **lie above or below the hyperplane**

What is a hyperplane?

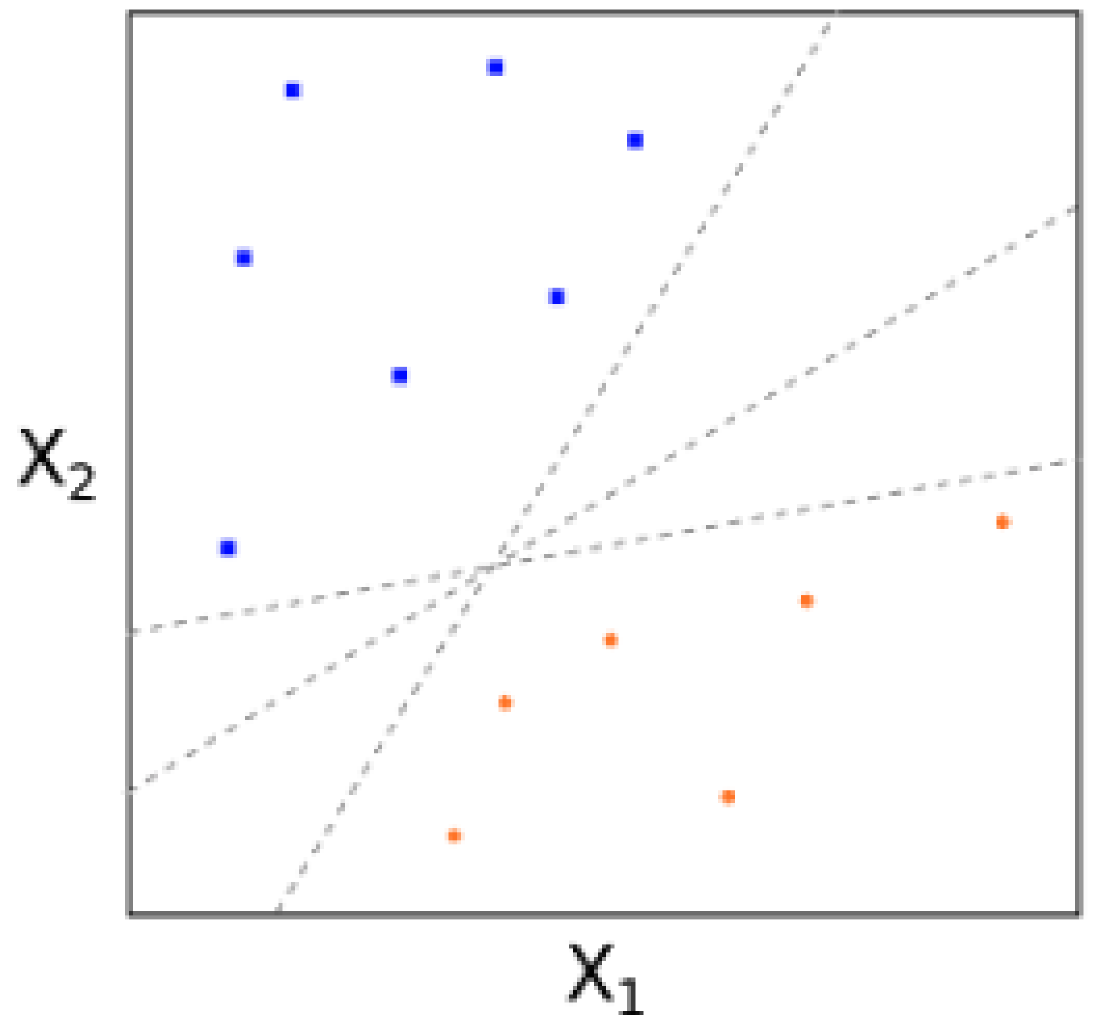
- In a p dimensional space, a **hyperplane is a flat subspace of dimension $p-1$**
- In two dimensions, a hyperplane is a flat one dimensional subspace which is a line
- In three dimensions, a hyperplane is a flat two dimensional subspace which is a **plane**
- For $p > 3$ dimensions, it's hard to visualize, but there are still $p-1$ dimensional flat subspaces
- A hyperplane divides the p dimensional space into two halves

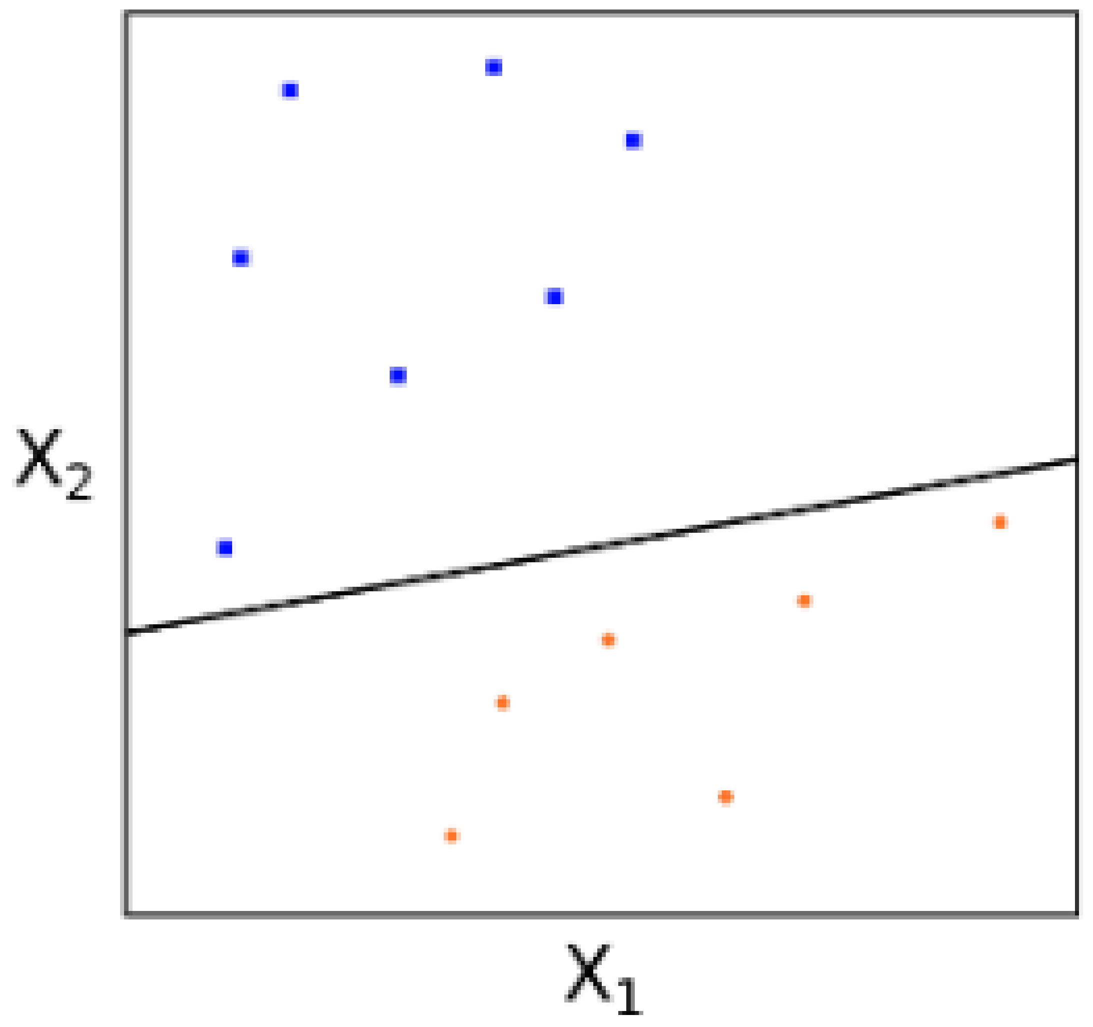




Optimal hyperplane

- Our main goal is to develop a classifier which has a hyperplane that perfectly classifies our data into two classes
- There could be a **infinite number of hyperplanes** that perfectly separate the classes, but we want an **optimal hyperplane** that perfectly separates the classes

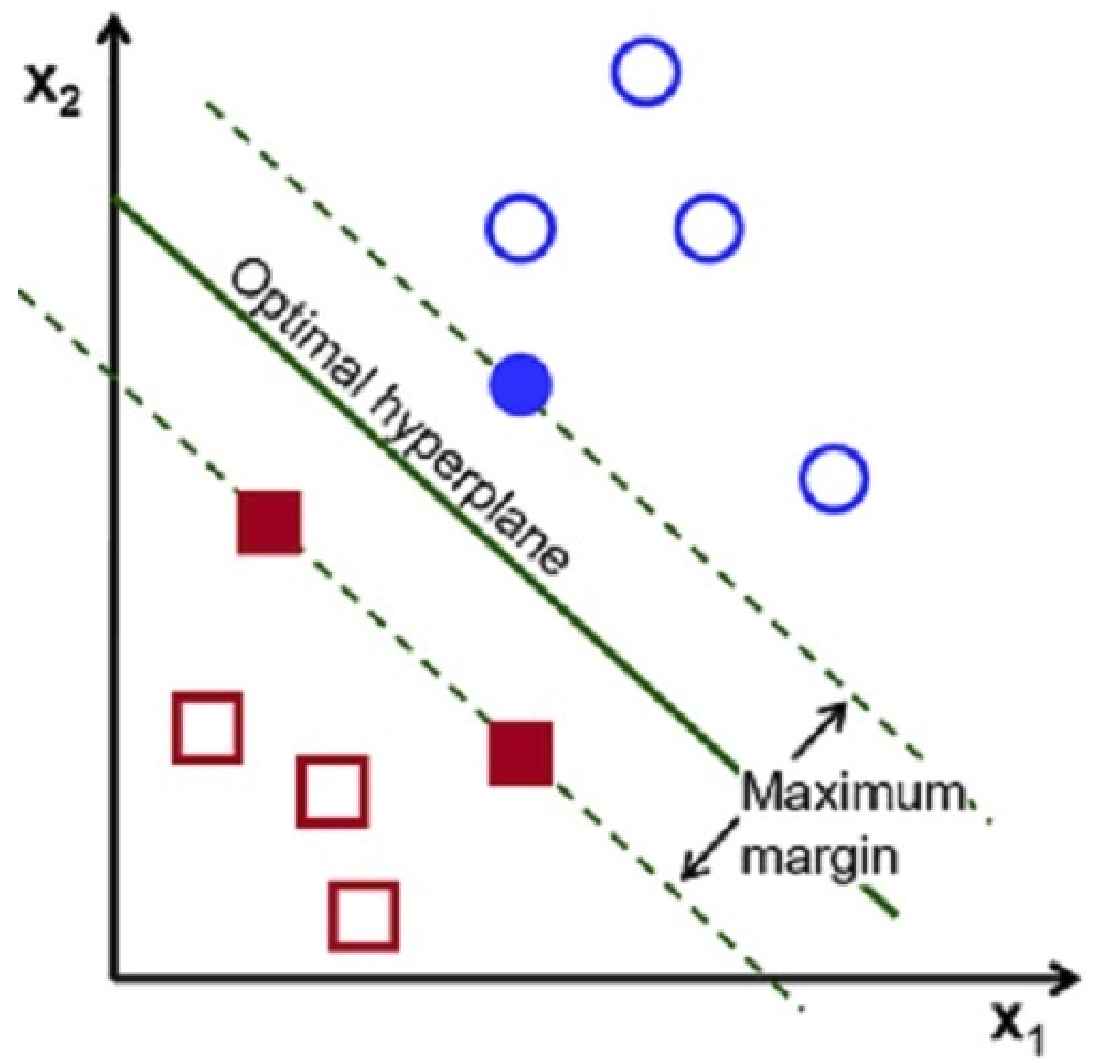




Maximal margin classifier

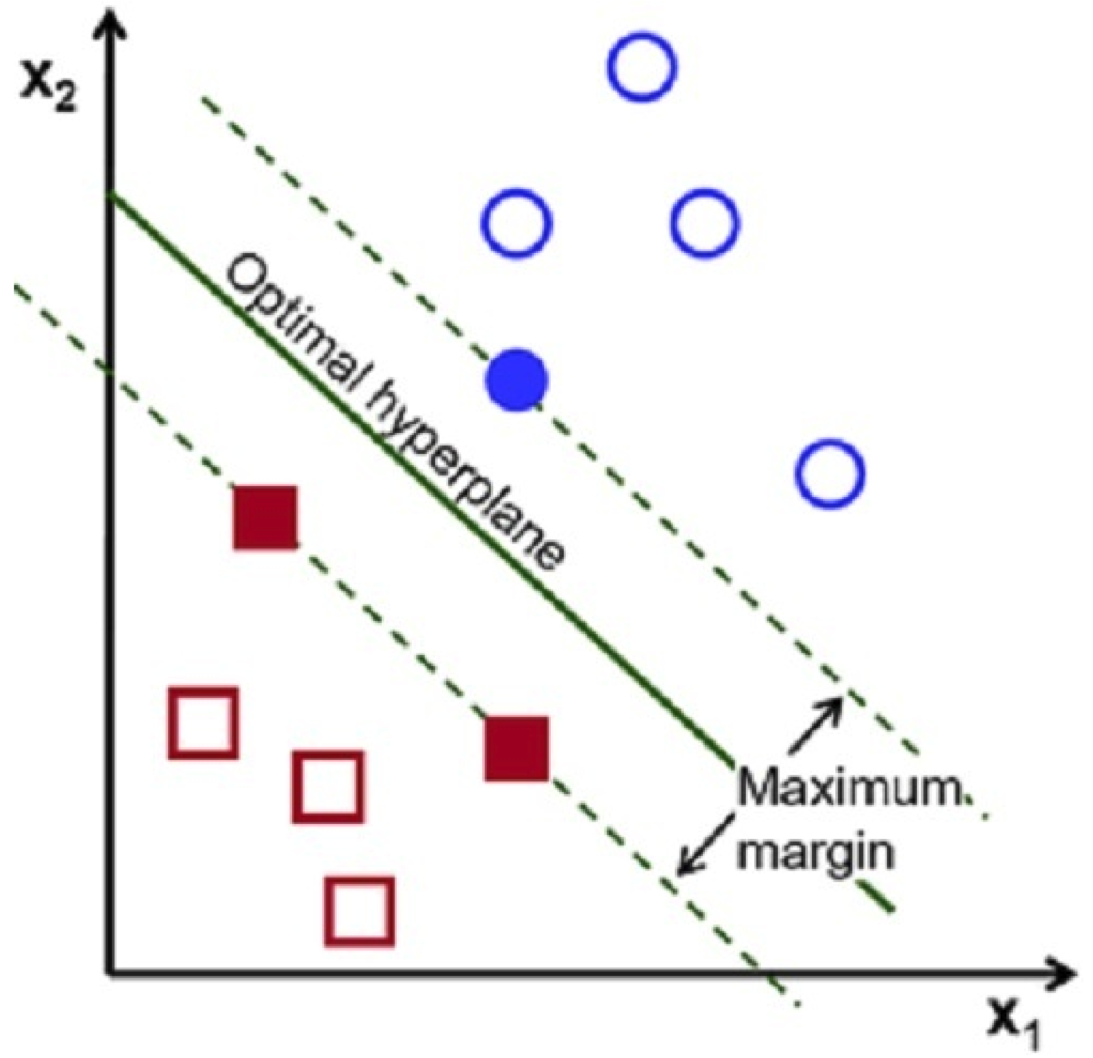
- We need to find an **optimal** hyperplane or a **maximal margin hyperplane** from the set of infinite hyperplanes

- We compute the **perpendicular distance** from each training observation for a given separating hyperplane



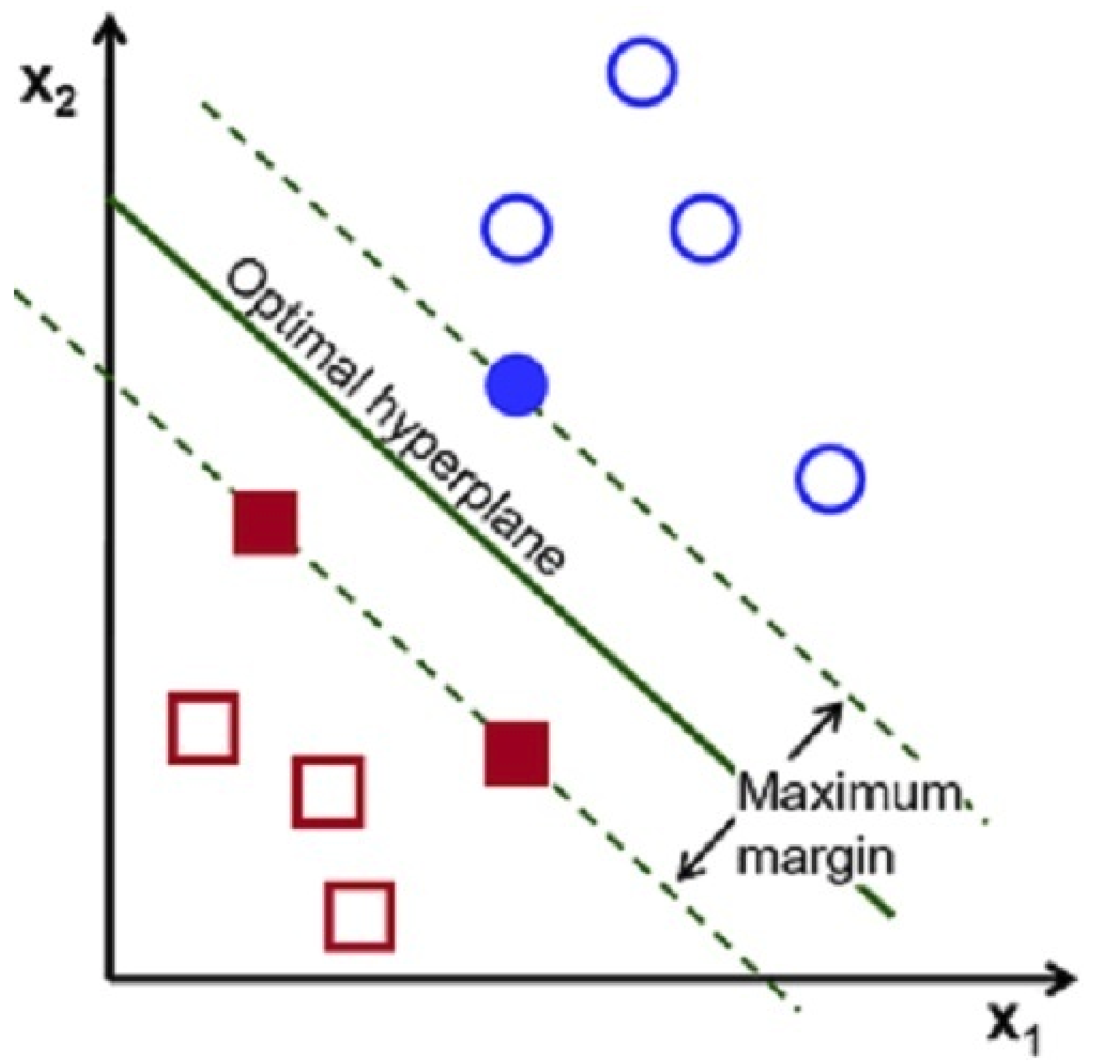
Maximal margin classifier

- The smallest perpendicular distance to a training observation from the hyperplane is known as the **margin**
- We use optimization methods to maximize this margin
- The maximal margin hyperplane is the hyperplane where the margin is the largest
- Such a classifier is known as **maximal margin classifier (MMC)**



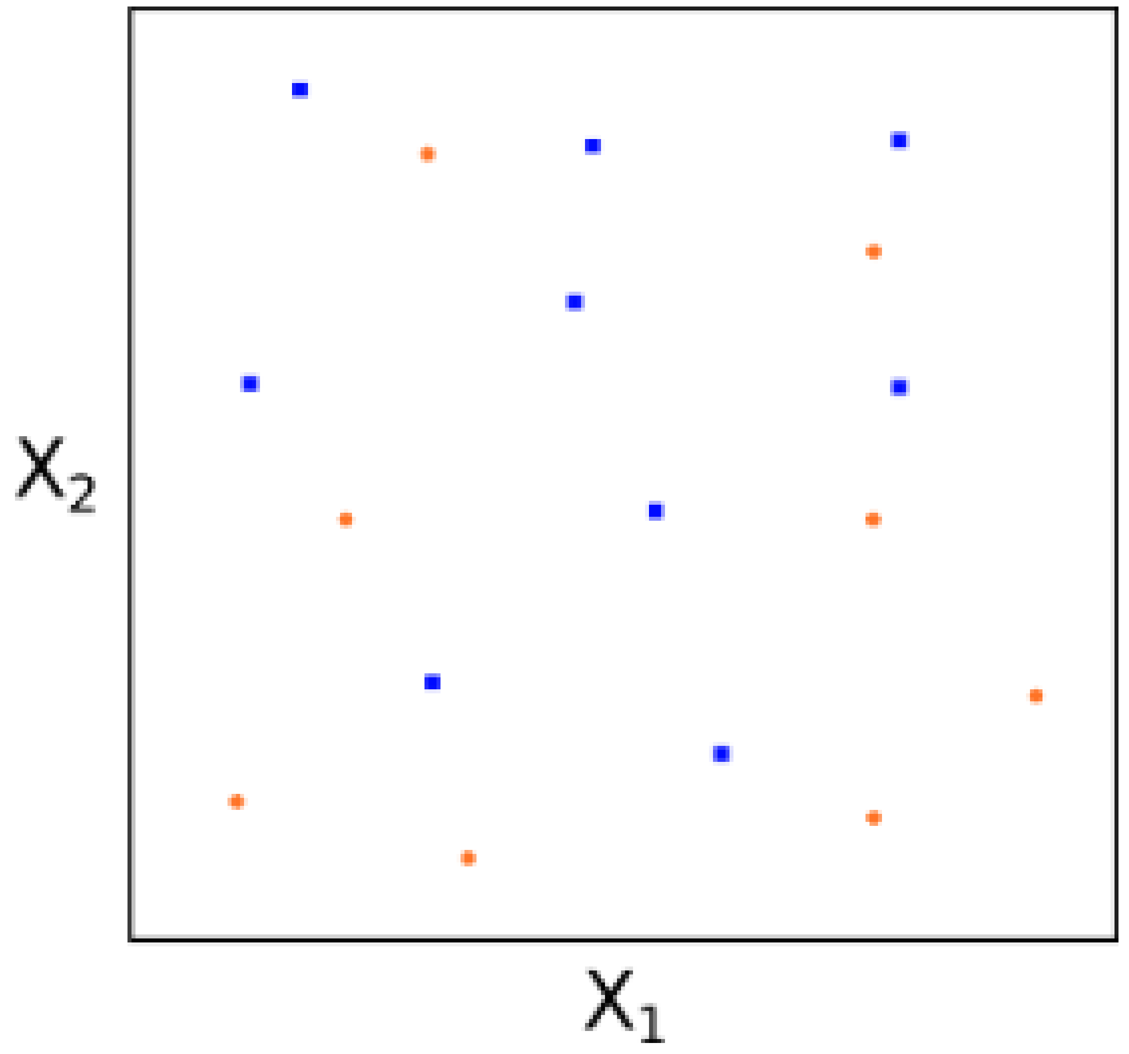
Support vectors

- The points that are used to decide this boundary are the set of closest equidistant perpendicular points, and are called the **support vectors**
- The maximum margin hyperplane depends directly on these support vectors, and not on the other observations
- In the image here, the points on the **margin are the support vectors**
- These support vectors define the decision boundary for classification



Pitfalls in MMC - no perfectly separating hyperplane

- MMC is a natural way to perform classification only if a natural hyperplane which perfectly separates the two classes exists
- But do you think that all the real life datasets are perfectly separable?
- In most real life cases, a **perfectly separating hyperplane will not exist**
- In such cases, we cannot use a maximal margin classifier



Pitfalls in MMC - overfitting

- Even when a perfectly separating hyperplane does exist, there are instances where it may not be desirable
- This method has a **high potential to overfit,** because it perfectly fits the data and is highly sensitive to only few training observations

Knowledge check 1

Warm up

- Before we start, check out this video about AI in the medical field: ***link***

Welcome back!

- In the previous session we have covered ensemble methods like Random Forest, Gradient Boosting Models for classification and the concepts of hyperplanes and Support Vector Machines
- Today we will start looking into:
  - differences between support vector classifier and support vector machine
  - building a support vector machine
  - using grid search to optimize SVM

Module completion checklist

Objective	Complete
Introduce the concept of a hyperplane and classification using a hyperplane	✔
Summarize the idea of maximal margin classifier and its pitfalls	✔
Explain the concept of support vectors and build a support vector classifier model	

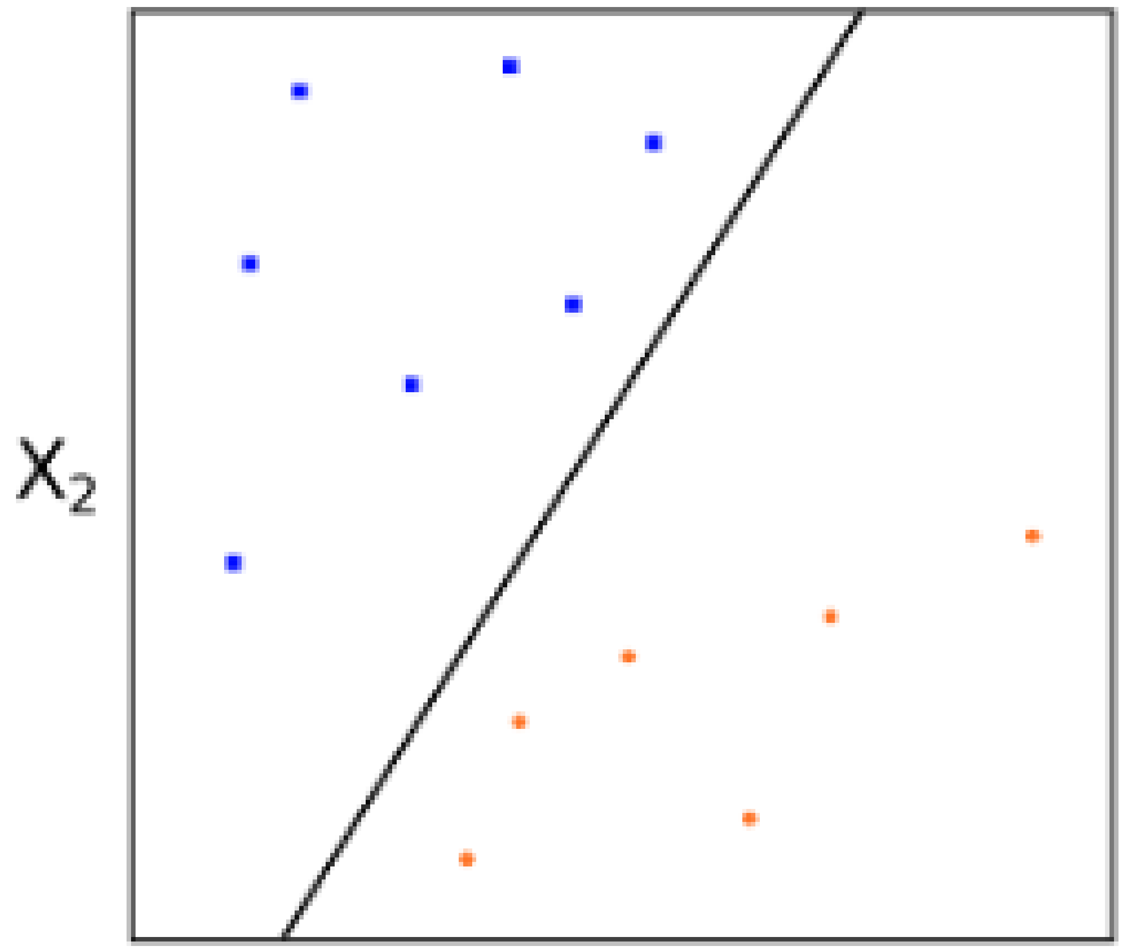
Loading packages

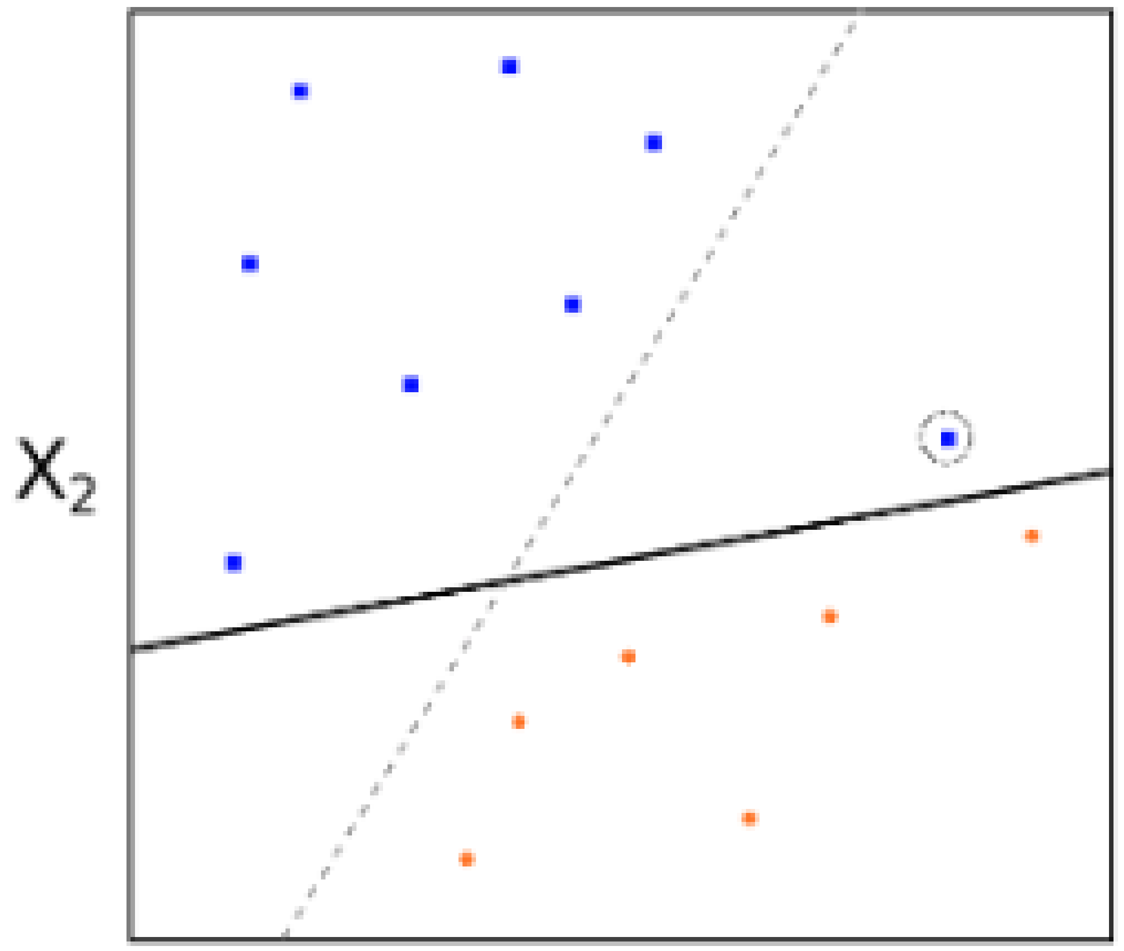
- Let’s load the packages we will be using
- These packages are used for classification in SVM

```
import os
import pickle
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pathlib import Path
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.model_selection import GridSearchCV
```

Support vector classifier

- To overcome the pitfalls of maximal margin classifier, we might be willing to consider a classifier based on a hyperplane that does not perfectly separate the two classes
- This will help add **greater robustness** to individual observations and **improve classification** of the observation
- It could be worth it to **misclassify a few training observations** in order to do a better job in classifying the remaining observations
- This classifier is called a support vector classifier or **soft margin classifier**



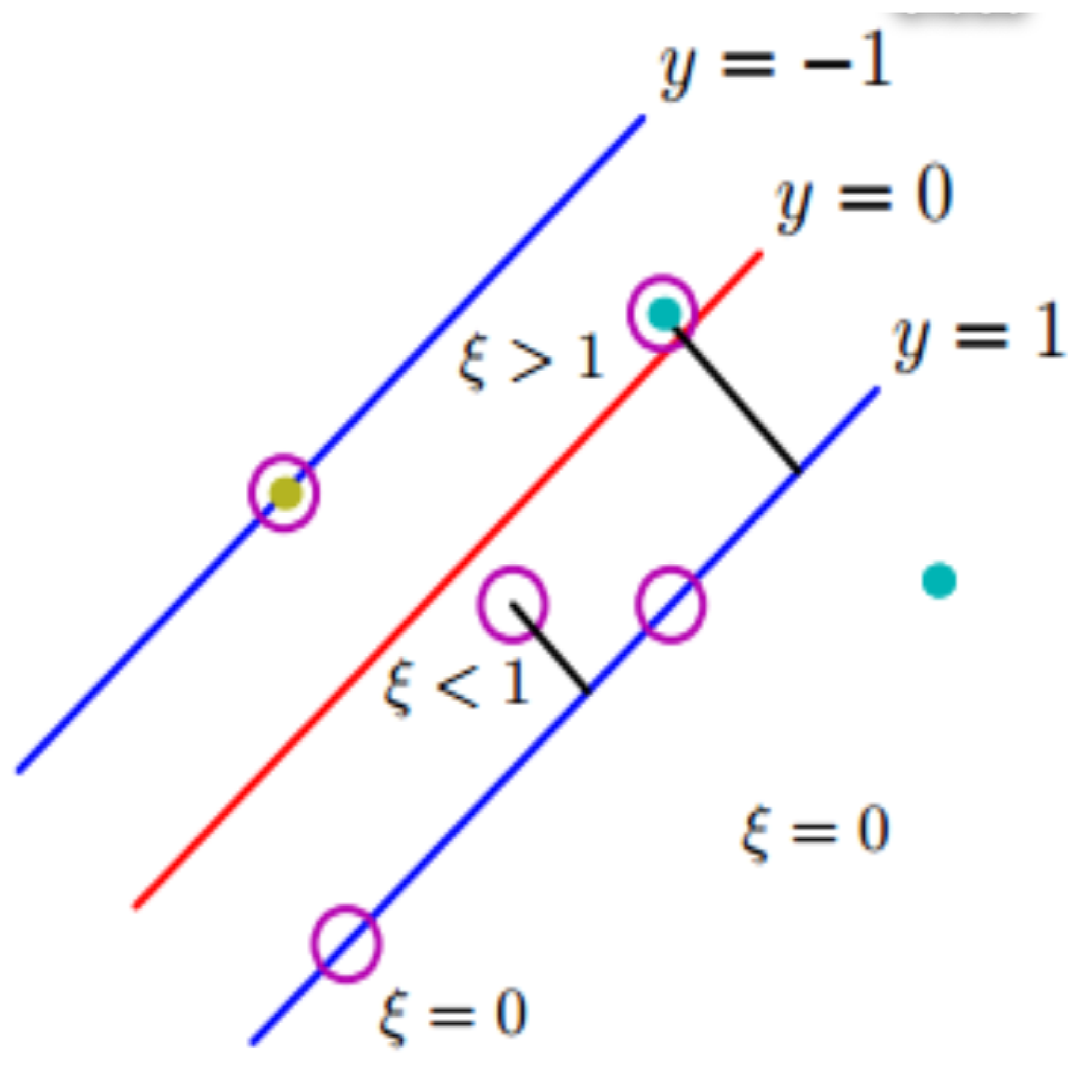


Tuning parameters

- There are three tuning parameters to get a optimal hyperplane for classification using a soft margin classifier
  - **Non-negative tuning parameter (aka cost function)** C which determines the severity of misclassification
  - **Margin width** M which we want to make as large as possible
  - **Slack variables** ϵ_i which we will talk about next

Slack variables

- Slack variables ϵ_i are the variables that allow the observations on the wrong side of the hyperplane
- ϵ_i tells us where the ith observation is located relative to the hyperplane and relative to the margin
- If $\epsilon_i = 0$, then the ith observation is on the **correct side of the margin**
- If $1 > \epsilon_i > 0$, then the ith observation is on the **wrong side of the margin**
- If $\epsilon_i >= 1$, then the ith observation is on the **wrong side of the hyperplane**



Cost function C

- C bounds the sum of the ϵ_i values
- It determines the number and severity of the violations to the margin and hyperplane that we will tolerate
- If $C = 0$, then we have **no limits for margin violations** which makes our soft margin classifier behave like a maximal margin classifier
- For $C > 0$, as C **increases**, we become **more tolerant of violations** to the margin and hence the margin widens
- C is generally chosen with cross-validation and controls the bias-variance trade-off
- When C is small, we have a **narrow margin that rarely violates**, which makes the classifier have low bias and high variance

Support vectors in SVC

- The observations that either lie on the margin or that violate the margin will affect the hyperplane
- The observations that lie clearly on the correct side of the margin do not affect the classifier
- The observations that lie on the wrong side of the margin drive the classifier and affect the hyperplane and those points are called **support vectors**

Building a Support vector classifier model

- We already used our `Costa Rican` dataset and classified the poverty levels of the individuals
- Today, we will determine whether a person is classified as **poor or not** based on the information we have about that person's living conditions and educational qualifications
- We will build a Support vector classifier model and discuss Support vector machine

Review data cleaning steps

- **Today, we will be loading the cleaned dataset we used in last class**
- To recap, the steps to get to this cleaned dataset were:
    - Remove household ID and individual ID
    - Remove variables with over 50% NAs
    - Transformed target variable to binary
    - Remove highly correlated variables

Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- We will use the `pathlib` library
- Let the `main_dir` be the variable corresponding to your `course` folder
- Let `data_dir` be the variable corresponding to your `data` folder

```
# Set 'main_dir' to location of the project folder
home_dir = Path(".").resolve()
main_dir = home_dir.parent.parent
print(main_dir)
```

```
data_dir = str(main_dir) + "/data"
print(data_dir)
```

Load the cleaned dataset

- Let’s load the pickled dataset, `costa_clean`
- Save it as `costa_clean`

```
costa_clean = pickle.load(open((data_dir + "/costa_clean.sav"),"rb"))
```

```
# Print the head.
print(costa_clean.head())
```

```
   rooms  tablet  males_under_12  ...  rural_zone  age  Target
0      3       0               0  ...           0   43    True
1      4       1               0  ...           0   67    True
2      8       0               0  ...           0   92    True
3      5       1               0  ...           0   17    True
4      5       1               0  ...           0   37    True

[5 rows x 81 columns]
```

Print info on data

- Let's view the column names

```
# Print the columns.
costa_clean.columns
```

```
Index(['rooms', 'tablet', 'males_under_12', 'males_over_12', 'males_tot',
       'females_under_12', 'females_over_12', 'females_tot', 'ppl_under_12',
       'ppl_over_12', 'ppl_total', 'years_of_schooling', 'wall_block_brick',
       'wall_socket', 'wall_prefab_cement', 'wall_wood', 'floor_mos_cer_terr',
       'floor_cement', 'floor_wood', 'ceiling', 'electric_public',
       'electric_coop', 'toilet_sewer', 'toilet_septic', 'cookenergy_elec',
       'cookenergy_gas', 'trash_truck', 'trash_burn', 'wall_bad', 'wall_reg',
       'wall_good', 'roof_bad', 'roof_reg', 'roof_good', 'floor_bad',
       'floor_reg', 'floor_good', 'disabled_ppl', 'male', 'female', 'under10',
       'free', 'married', 'separated', 'single', 'hh_head', 'hh_spouse',
       'hh_child', 'num_child', 'num_adults', 'num_65plus', 'num_hh_total',
       'dependency_rate', 'male_hh_head_educ', 'female_hh_head_educ',
       'meaneduc', 'educ_none', 'educ_primary_inc', 'educ_primary',
       'educ_secondary_inc', 'educ_secondary', 'educ_undergrad', 'bedrooms',
       'ppl_per_room', 'house_owned_full', 'house_owned_paying',
       'house_rented', 'house_other', 'computer', 'television',
       'num_mobilephones', 'region_central', 'region_Chorotega',
       'region_pacifico', 'region_brunca', 'region_antlantica',
       'region_huetar', 'urban_zone', 'rural_zone', 'age', 'Target']
```

Split into training and test sets

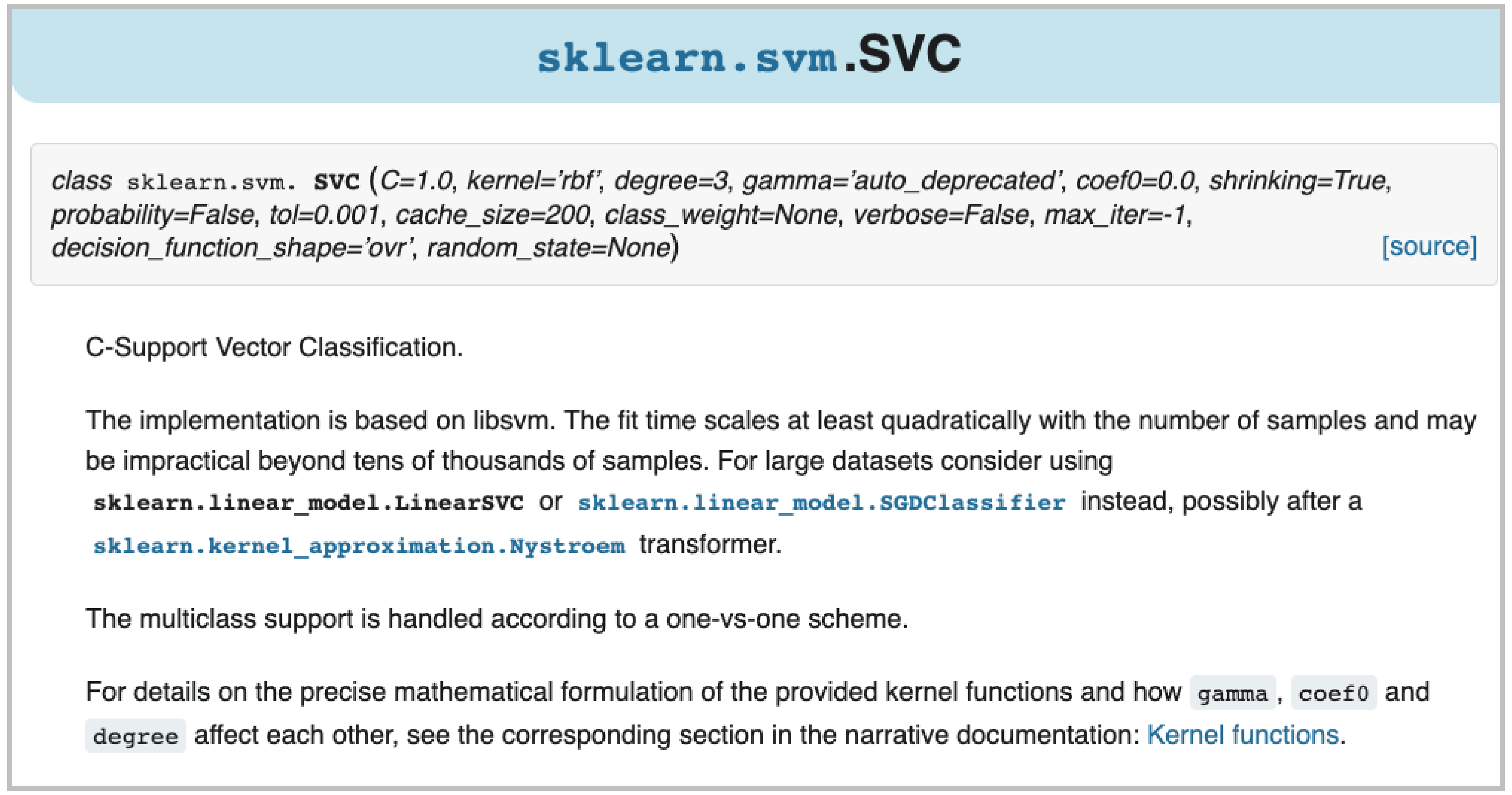
```
# Select the predictors and target.
X = costa_clean.drop(['Target'], axis = 1)
y = np.array(costa_clean['Target'])

# Set the seed to 1.
np.random.seed(1)

# Split into training and test sets with 70% train and 30% test.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
```

Support vector function in sklearn

- We will use the `svm.svc` function from the `sklearn` package to build our support vector classifier
- Find detailed documentation ***here***



sklearn.svm.SVC

class `sklearn.svm.` **SVC** (*C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None*) [source]

C-Support Vector Classification.

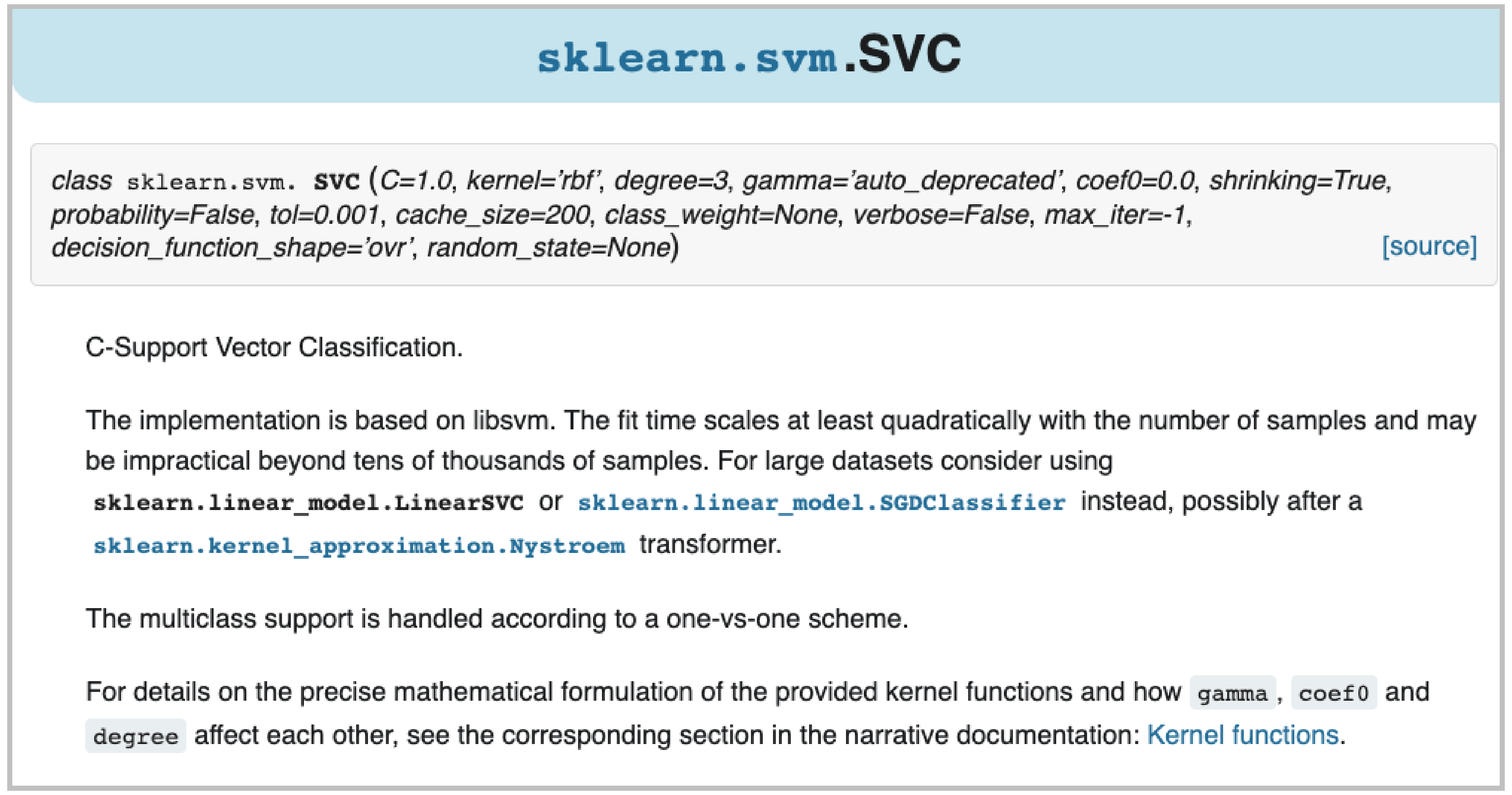
The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using `sklearn.linear_model.LinearSVC` or `sklearn.linear_model.SGDClassifier` instead, possibly after a `sklearn.kernel_approximation.Nystroem` transformer.

The multiclass support is handled according to a one-vs-one scheme.

For details on the precise mathematical formulation of the provided kernel functions and how `gamma`, `coef0` and `degree` affect each other, see the corresponding section in the narrative documentation: Kernel functions.

Support vector function in sklearn

- The main arguments of the function are :
  - `C` : Non-negative tuning parameter (aka cost function)
  - `kernel` : Specifies the kernel type to be used in the algorithm (We will use `linear` for SVC and one of the other types for SVM)
  - `gamma` : the value which controls the shape of the kernel (We will talk more about this when we build an SVM)
  - `probability` : whether to enable probability estimates



sklearn.svm.SVC

class `sklearn.svm.` **SVC** (*C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', random_state=None*) [source]

C-Support Vector Classification.

The implementation is based on libsvm. The fit time scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using `sklearn.linear_model.LinearSVC` or `sklearn.linear_model.SGDClassifier` instead, possibly after a `sklearn.kernel_approximation.Nystroem` transformer.

The multiclass support is handled according to a one-vs-one scheme.

For details on the precise mathematical formulation of the provided kernel functions and how `gamma`, `coef0` and `degree` affect each other, see the corresponding section in the narrative documentation: Kernel functions.

Support vector classifier model

- Let's train our model using `X_train` and `y_train`
- We will use the default value of `C`, which is 1
- Linear kernel specifies to create a linear hyperplane in the support vector classifier

```
# Create an SVC classifier.
svclassifier = SVC(kernel = 'linear',
                   probability = True)

# Fit the model.
svclassifier.fit(X_train, y_train)
```

```
SVC(kernel='linear', probability=True)
```

Recap: score evaluation function

- In the last module, we used a function to wrap all performance score calculations into one block of code which can be called repeatedly
- We will use this function to evaluate our SVM models today

```
def get_performance_scores(y_test, y_predict, y_predict_prob, eps=1e-15, beta=0.5):
    from sklearn import metrics
    # Scores keys.
    metric_keys = ["accuracy", "precision", "recall", "f1", "fbeta", "log_loss", "AUC"]
    # Score values.
    metric_values = [None]*len(metric_keys)
    metric_values[0] = metrics.accuracy_score(y_test, y_predict)
    metric_values[1] = metrics.precision_score(y_test, y_predict)
    metric_values[2] = metrics.recall_score(y_test, y_predict)
    metric_values[3] = metrics.f1_score(y_test, y_predict)
    metric_values[4] = metrics.fbeta_score(y_test, y_predict, beta=beta)
    metric_values[5] = metrics.log_loss(y_test, y_predict_prob[:, 1], eps=eps)
    metric_values[6] = metrics.roc_auc_score(y_test, y_predict_prob[:, 1])
    perf_metrics = dict(zip(metric_keys, metric_values))
    return(perf_metrics)
```

Predict on the test dataset

```
# Predict on the test dataset.
svc_y_predict = svclassifier.predict(X_test)
svc_y_predict[0:5]

#Predict on test, but instead of labels
# we will get probabilities for class 0 and 1.
```

```
array([ True, False,  True,  True,  True])
```

```
svc_y_predict_prob = svclassifier.predict_proba(X_test)
print(svc_y_predict_prob[5:])
```

```
[[0.56316354 0.43683646]
 [0.02540825 0.97459175]
 [0.00935204 0.99064796]
 ...
 [0.20868202 0.79131798]
 [0.84090023 0.15909977]
 [0.75329657 0.24670343]]
```

Predict on the test dataset

```
svc_scores = get_performance_scores(y_test, svc_y_predict, svc_y_predict_prob)
print(svc_scores)
```

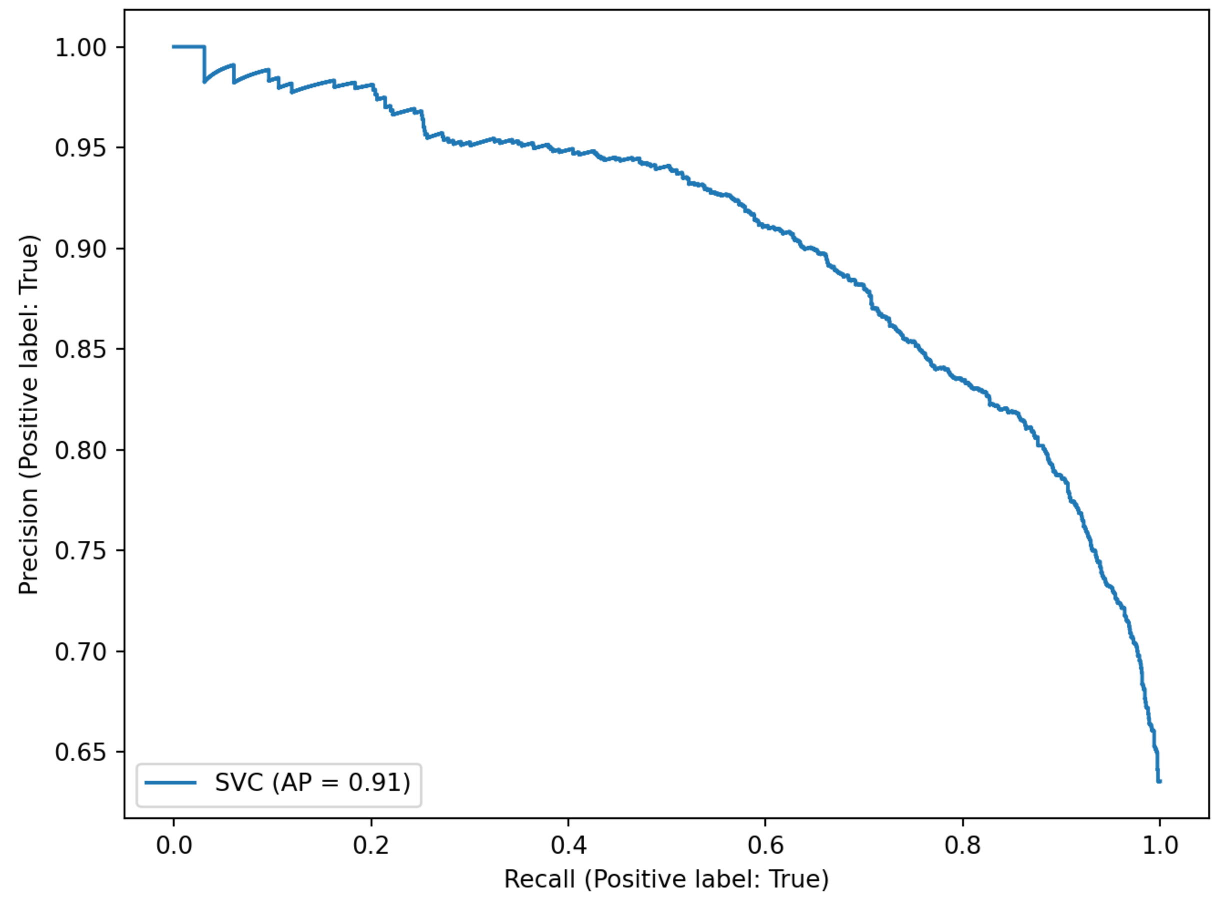
```
{'accuracy': 0.7894002789400278, 'precision': 0.8085215605749486, 'recall':
0.872093023255814, 'f1': 0.8391049547149706, 'fbeta': 0.82048343404876, 'log_loss':
0.45829399974895174, 'AUC': 0.8533430102212127}
```

- As we can see, our SVC model performed moderately on all fronts

Precision vs recall curve: the tradeoff

- Plot precision vs recall curve for the SVC model

```
svc_prec_recall =
metrics.plot_precision_recall_curve(svclassifier,
                                    X_test, y_test,
                                    name = "SVC")
plt.show()
```

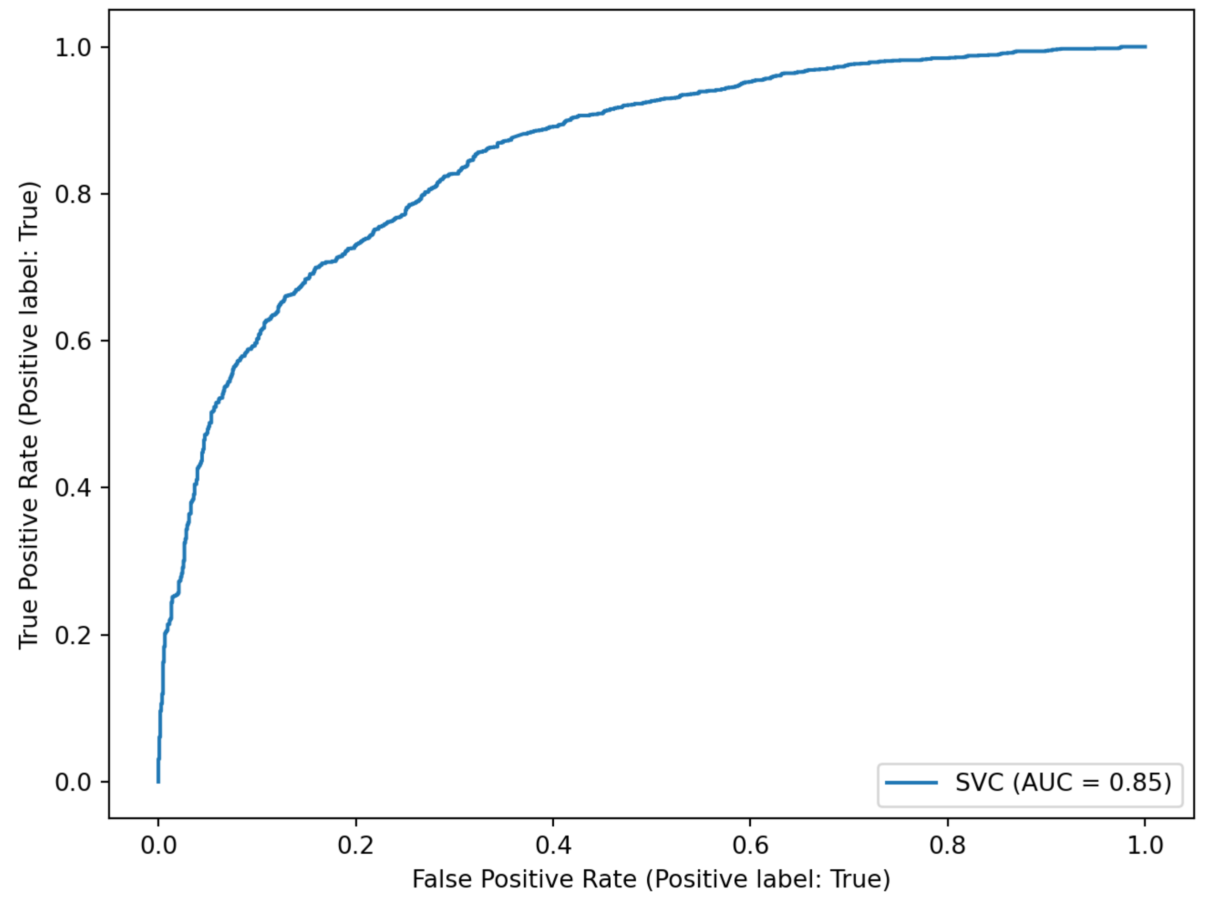


ROC curve: the tradeoff

- Plot ROC curve for the SVC model

```
svc_roc = metrics.plot_roc_curve(svclassifier,
                                X_test,
                                y_test,
                                name = "SVC")
plt.show()
```

- The AUC for this curve is about 85%, which is not bad, but we can do better



Save final metrics of SVC

- **Let's save our svc score in our `metrics_svm` dataset**
- We first have to load our `metrics_gbm` dataframe which we created previously

```
metrics_svm = pickle.load(open((data_dir +
"/metrics_gbm.sav"),"rb"))
```

Save final metrics of SVC

```
# Add the model to our dataframe.
metrics_svm.update({"SVC": svc_scores})
print(metrics_svm)
```

```
{'RF': {'accuracy': 0.9483960948396095, 'precision': 0.9447424892703863, 'recall':
0.9750830564784053, 'f1': 0.9596730245231607, 'fbeta': 0.9506586050529044, 'log_loss':
0.21947942349408847, 'AUC': 0.986855647527701}, 'Optimized RF': {'accuracy':
0.9483960948396095, 'precision': 0.9400212314225053, 'recall': 0.9806201550387597, 'f1':
0.9598915989159891, 'fbeta': 0.9478698351530723, 'log_loss': 0.21667604220201855, 'AUC':
0.9876901226920936}, 'GBM': {'accuracy': 0.8291492329149233, 'precision':
0.8374358974358974, 'recall': 0.9042081949058693, 'f1': 0.869542066027689, 'fbeta':
0.8499895898396836, 'log_loss': 0.39102395649143923, 'AUC': 0.902052011186816}, 'Optimized
GBM': {'accuracy': 0.9592050209205021, 'precision': 0.9678670360110804, 'recall':
0.9673311184939092, 'f1': 0.9675990030462477, 'fbeta': 0.9677598050077555, 'log_loss':
0.5621060823499561, 'AUC': 0.9918591095177615}, 'SVC': {'accuracy': 0.7894002789400278,
'precision': 0.8085215605749486, 'recall': 0.872093023255814, 'f1': 0.8391049547149706,
'fbeta': 0.82048343404876, 'log_loss': 0.4582939974895174, 'AUC': 0.8533430102212127}}
```

- SVC metrics are lower than ensemble methods on almost all fronts because, in most cases, ensemble of classifiers tend to outperform a single classifier

Knowledge check 2

Exercise 1

Module completion checklist

Objective	Complete
Introduce the concept of a hyperplane and classification using a hyperplane	✔
Summarize the idea of maximal margin classifier and its pitfalls	✔
Explain the concept of support vectors and build a support vector classifier model	✔

Congratulations on completing this module!