DATA SOCIETY®

Advanced classification - day 3

"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	
Summarize the key difference between support vector classifier and support vector machine	
Build a support vector machine model to classify the Costa Rica dataset	
Optimize the support vector machine model using grid search	

Why study SVM?

- 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?
- **Support vector machines** is a machine learning algorithm which was developed to handle such issues in our dataset

Concept behind different classifiers

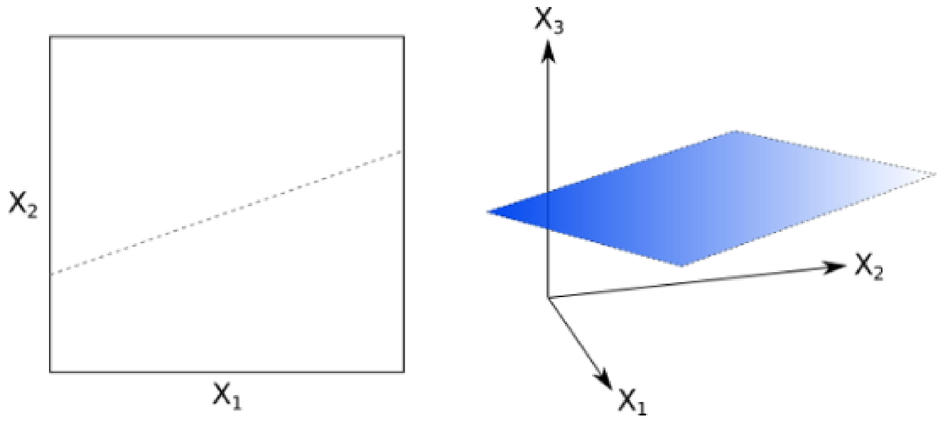
- We studied many classification algorithms so far
  - Logistic regression classifies based on **probability**
  - Trees and ensemble methods classify based on the value of predictors using the idea of **segmentation**
- The model we are going to study today makes use of a **hyperplane** to classify 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 to two categories based on whether the features **lie above or below the hyperplane**

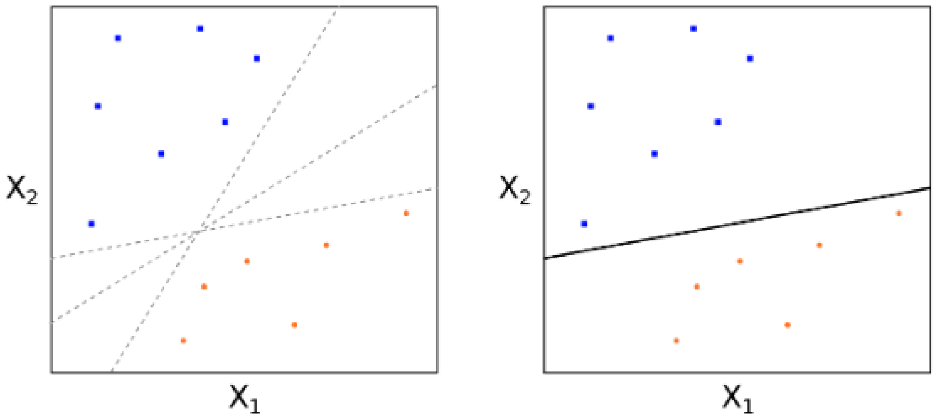
What is a hyperplane?

- 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 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



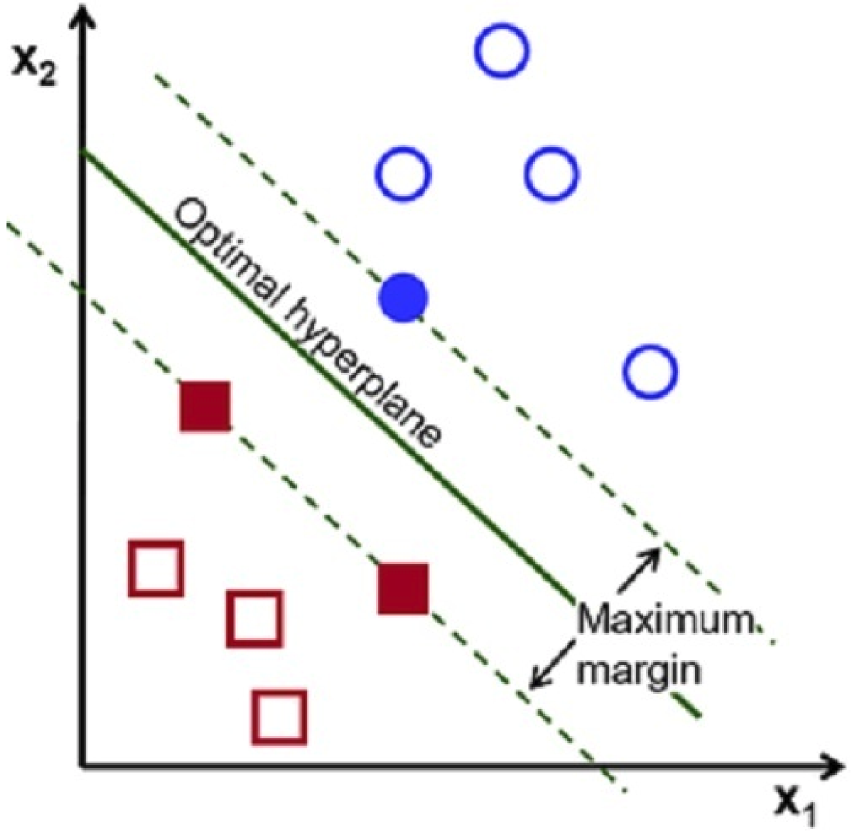
Optimal hyperplane

- 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



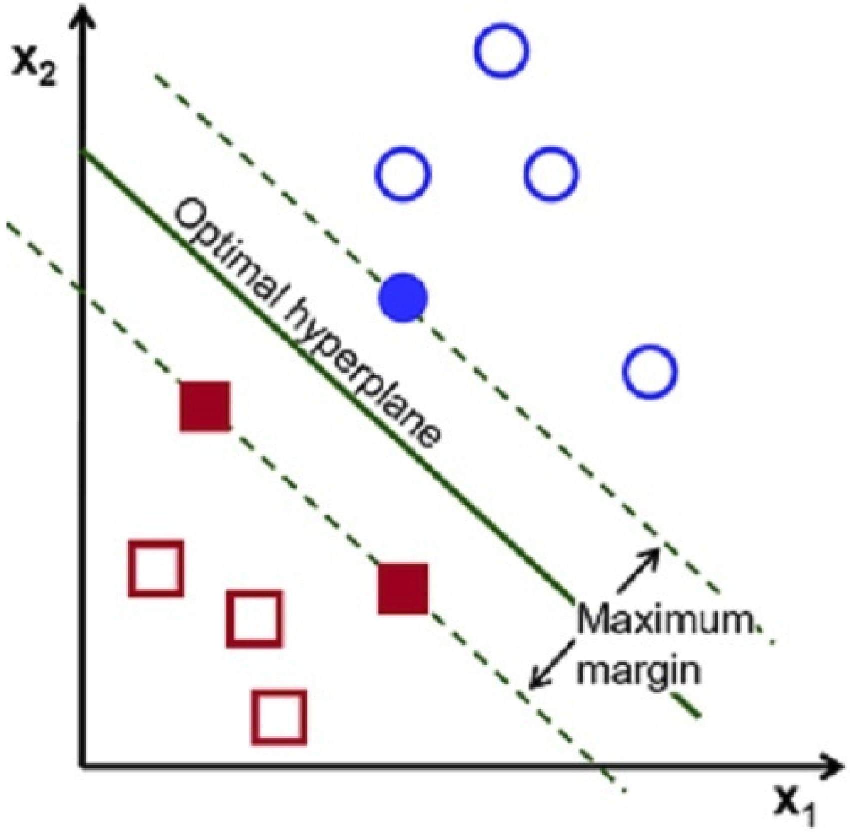
Maximal margin classifier

- To construct an **optimal** hyperplane first we need to develop a **maximal margin hyperplane**, which creates the optimal separating hyperplane
- Then, compute the **perpendicular distance** from each training observation for a given separating hyperplane



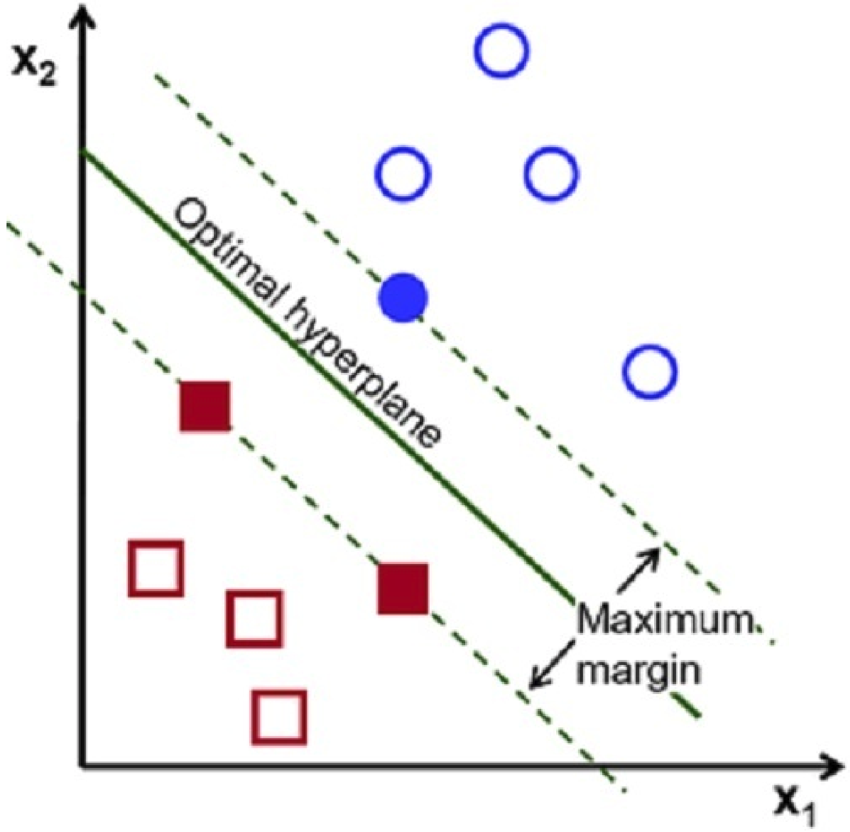
Maximal margin classifier

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



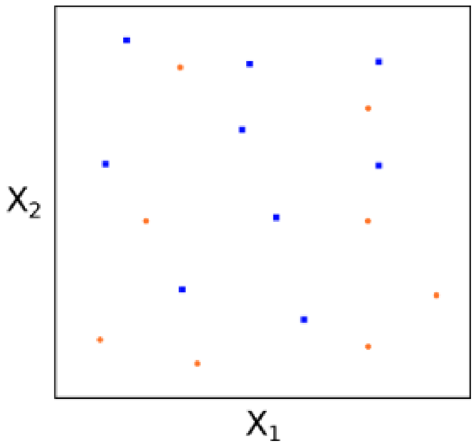
Support vectors

- 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

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	
Summarize the key difference between support vector classifier and support vector machine	
Build a support vector machine model to classify the Costa Rica dataset	
Optimize the support vector machine model using grid search	

Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- Let the `main_dir` be the variable corresponding to your `af-werx` folder

```
# Set `main_dir` to the location of your `af-werx` folder (for Linux).
main_dir = "/home/[username]/Desktop/af-werx"
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Mac).
main_dir = '/Users/[username]/Desktop/af-werx'
```

```
# Set `main_dir` to the location of your `af-werx` folder (for Windows).
main_dir = "C:\\Users\\[username]\\Desktop\\af-werx"
```

```
# Make `data_dir` from the `main_dir` and
# remainder of the path to data directory.
data_dir = main_dir + "/data"
```

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 sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import GridSearchCV
```

Working directory

- Set working directory to `data_dir`

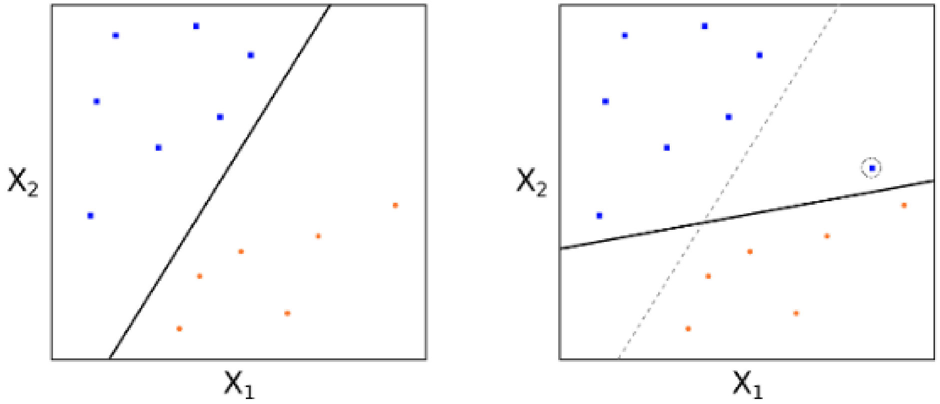
```
# Set working directory.
os.chdir(data_dir)
```

```
# Check working directory.
print(os.getcwd())
```

```
/home/[user-name]/Desktop/af-werx/data
```

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 for **greater robustness** to individual observations and **improved classification** of the observation
- It could be worth to **misclassify a few training observations** in order to do a better job in classifying the remaining observations
- This classifier is called as the support vector classifier or **soft margin classifier**

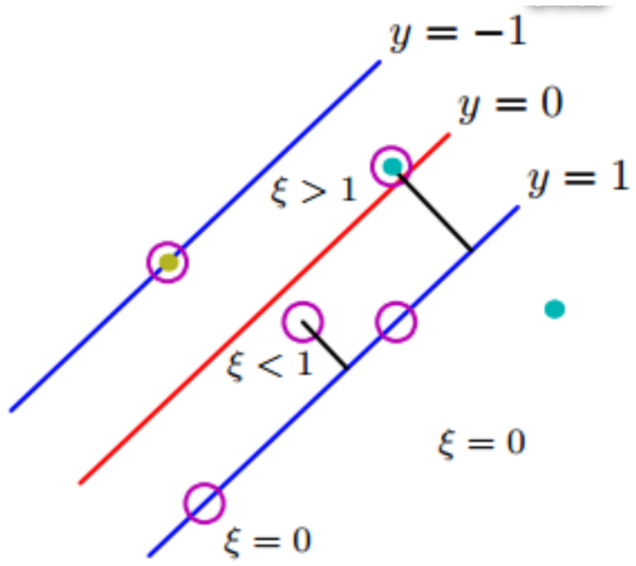


Tuning parameters

- 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`
  - Margin width `M` which we want to make as large as possible
  - Slack variables `ei` which we will talk about next

Slack variables

- Slack variables `ei` are the variables that allow the observations on the wrong side of the hyperplane
- `ei` tells us where the *i*th observation is located relative to the hyperplane and relative to the margin
- If `ei = 0`, then the *i*th observation is on the **correct side of the margin**
- If `ei > 0`, then the *i*th observation is on the **wrong side of the margin**
- If `ei > 1`, then the *i*th observation is on the **wrong side of the hyperplane**



Cost function C

- C bounds the sum of the `eis`
- 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**

Goal for the day

- We already used our `Costa Rican` dataset and classified the poverty levels of the individuals
- We are going to do the same today as well
- We want to find whether a person is **poor or not** based on the information we have about that person's living condition and education qualifications
- We will build 3 models today
  - Support vector classifier
  - Support vector machine
  - Optimized support vector machine using grid search

Review data cleaning steps from last week

- **Today, we will be loading the cleaned dataset we used 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

Load the cleaned dataset

- Let's load the dataset from last week, `costa_no_hc` - no highly correlated variables
- Save it as `costa_clean`

```
os.chdir(data_dir)
```

```
costa_clean = pickle.load(open("costa_no_hc.sav","rb"))
```

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

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

[5 rows x 61 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',
       'females_under_12', 'females_over_12', 'years_of_schooling',
       'wall_block_brick', 'wall_socket', 'wall_prefab_cement', 'wall_wood',
       'floor_mos_cer_terr', 'floor_wood', 'ceiling', 'electric_public',
       'toilet_sewer', 'cookenergy_elec', 'trash_truck', 'wall_bad',
       'wall_reg', 'roof_bad', 'roof_reg', 'floor_bad', 'floor_reg',
       'disabled_ppl', 'male', 'under10', 'free', 'married', 'separated',
       'single', 'hh_head', 'hh_spouse', 'hh_child', 'num_65plus',
       'dependency_rate', 'male_hh_head_educ', 'female_hh_head_educ',
       'meaneduc', 'educ_primary_inc', 'educ_primary', 'educ_secondary_inc',
       'educ_secondary', 'educ_undergrad', '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', 'age', 'Target'],
      dtype='object')
```

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.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
```

Support vector function in sklearn

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.

Find detailed documentation here: ***https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html***

Support vector classifier model

- Let's train our model using `X_train` and `y_train`

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

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

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

- Linear kernel specifies to create a linear hyperplane in the support vector classifier

Predict on the test dataset

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

# Find the accuracy value.
```

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

```
svc_accuracy = metrics.accuracy_score(y_test, svc_y_pred)
print ("Accuracy on test data using svc: ", svc_accuracy)
```

```
Accuracy on test data using svc:  0.7723152022315202
```

Print confusion matrix

```
# Print the confusion matrix.
confusion_matrix(y_test, svc_y_pred)
```

```
array([[1572,  234],
       [ 419,  643]])
```

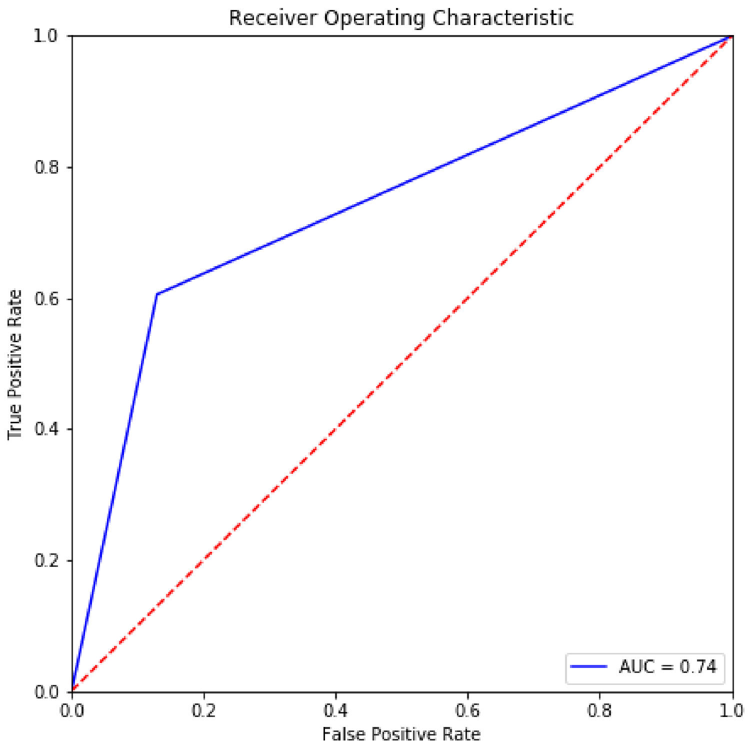
Print classification report

```
# Print an entire classification report.
class_report = metrics.classification_report(y_test,
                                             svc_y_pred)
print(class_report)
```

```
              precision    recall  f1-score   support

       False       0.79      0.87      0.83      1806
        True       0.73      0.61      0.66      1062

    accuracy                           0.77      2868
   macro avg       0.76      0.74      0.75      2868
weighted avg       0.77      0.77      0.77      2868
```

Plot ROC curve

```
# Calculate metrics for ROC (fpr, tpr) and
calculate AUC.
fpr, tpr, threshold = metrics.roc_curve(y_test,
svc_y_pred)
roc_auc = metrics.auc(fpr, tpr)

# Plot ROC.
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' %
roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Save final accuracy of SVC

- **Let's save our svc score in our `model_final` dataset**
- We first have to load our `model_final` dataframe from last class

```
svc_model =
pickle.load(open("model_final_optimized_ensemble.sav","rb"))
```

Save final accuracy of SVC

```
# Add the model to our dataframe.
svc_model = svc_model.append({'metrics' : "accuracy" ,
'values' : round(svc_accuracy,4),
'model':'svc' } ,
ignore_index = True)

print(svc_model)
```

```
     metrics  values                          model
0   accuracy  0.6046                          knn_5
1   accuracy  0.6188               knn_GridSearchCV
2   accuracy  0.6287                         knn_29
3   accuracy  0.6356                       logistic
4   accuracy  0.7845        logistic_whole_dataset
5   accuracy  0.7859                logistic_tuned
6   accuracy  0.6611            tree_simple_subset
7   accuracy  0.9407            tree_all_variables
8   accuracy  0.7183  tree_all_variables_optimized
9   accuracy  0.9338                 random forest
10  accuracy  0.8644                      boosting
11  accuracy  0.8536              optimized forest
12  accuracy  0.8563                 gbm_optimized
13  accuracy  0.7723                           svc
```

- SVC accuracy is lower than ensemble methods 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	✔
Summarize the key difference between support vector classifier and support vector machine	
Build a support vector machine model to classify the Costa Rica dataset	
Optimize the support vector machine model using grid search	

Classification with non-linear boundary

- What if our data cannot be classified using linear boundary?
- SVCs can be useless in a highly non-linear class boundary
- In that case, we can introduce a **classifier with a non-linear decision boundary** or non-linear hyperplane
- Support vector classifiers with a non-linear decision boundary are called **support vector machines (SVM)**

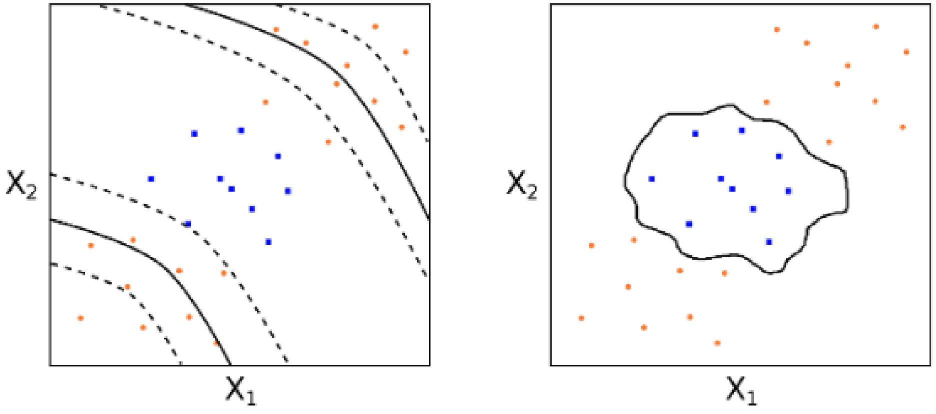
Transforming the features

- The idea is to transform the p features into a higher dimensional space
- For example, if we are transforming in quadratic space, we convert p features to a $2\mathrm{p}$ feature dimension

$$\text{Consider p features}$$
$$x_1, x_2, \ldots\ldots x_n$$
$$\text{They are transformed to 2p features}$$
$$x_1, x_1^2, \ldots x_n, x_n^2$$

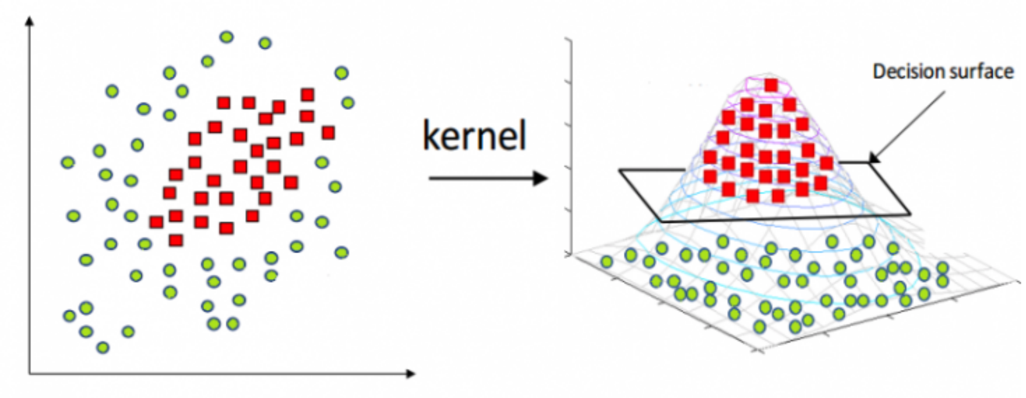
Support vector machine

- So the decision boundary is non-linear in p-dimensional space, but it is linear in the new transformed dimensional space
- This is called the kernel approach and the classifier is called the **support vector machines (SVM)**
- It can use **quadratic, cubic, or even higher order polynomial functions**
- The problem with it is that we start to accumulate more features since they are transformed and computation becomes unmanageable quickly



Decision boundary

- The decision boundary is non-linear in p-dimensional space, but it is linear in the new transformed dimensional space



- We have different kernels (non linear boundary conditions) to do it
  - radial
  - polynomial
  - quadratic
  - cubic

Radial basis function kernel

- Today we will use radial basis function (RBF) kernel for our data
- RBF kernel is the general purpose kernel used when there is no prior knowledge about the data
- Consider there are two observations x and x' - the RBF kernel is defined by the mathematical equation

$$K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$$

- `Gamma` is the value which controls the shape of the kernel

Build a SVM model

```
# Build the SVM model.
# Note here that the kernel rbf means radial kernel.
sv_machine = SVC(kernel = 'rbf', gamma = 0.011)
sv_machine.fit(X_train, y_train)
```

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.011, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

SVM accuracy

```
# Predict on the test data.
y_pred = sv_machine.predict(X_test)
y_pred[0:5]

# Find the accuracy value for SVM model.
```

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

```
svm_accuracy = metrics.accuracy_score(y_test, y_pred)
svm_accuracy
```

```
0.7737099023709902
```

Print confusion matrix

```
# Print the confusion matrix.
confusion_matrix(y_test, y_pred)
```

```
array([[1580,  226],
       [ 423,  639]])
```

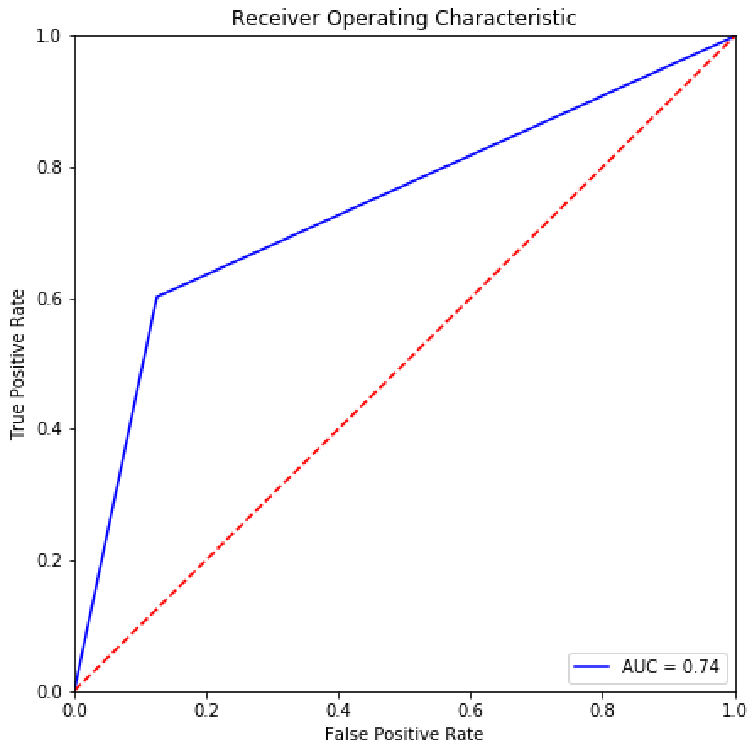
Print classification report

```
# Print an entire classification report.
class_report = metrics.classification_report(y_test,
                                             y_pred)
print(class_report)
```

```
              precision    recall  f1-score   support

       False       0.79      0.87      0.83      1806
        True       0.74      0.60      0.66      1062

    accuracy                           0.77      2868
   macro avg       0.76      0.74      0.75      2868
weighted avg       0.77      0.77      0.77      2868
```

Plot ROC curve

```
# Calculate metrics for ROC (fpr, tpr) and
calculate AUC.
fpr, tpr, threshold = metrics.roc_curve(y_test,
y_pred)
roc_auc = metrics.auc(fpr, tpr)

# Plot ROC.
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' %
roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



Save final accuracy of SVM

```
# Add the model to our dataframe.
svc_model = svc_model.append({'metrics' : "accuracy" ,
'values' : round(svm_accuracy,4),
'model':'svm' } ,
ignore_index = True)

print(svc_model)
```

```
     metrics  values                        model
0   accuracy  0.6046                        knn_5
1   accuracy  0.6188             knn_GridSearchCV
2   accuracy  0.6287                       knn_29
3   accuracy  0.6356                     logistic
4   accuracy  0.7845      logistic_whole_dataset
5   accuracy  0.7859              logistic_tuned
6   accuracy  0.6611          tree_simple_subset
7   accuracy  0.9407          tree_all_variables
8   accuracy  0.7183  tree_all_variables_optimized
9   accuracy  0.9338               random forest
10  accuracy  0.8644                    boosting
11  accuracy  0.8536            optimized forest
12  accuracy  0.8563               gbm_optimized
13  accuracy  0.7723                         svc
14  accuracy  0.7737                         svm
```

- The accuracy of SVM increased when compared to SVC mostly because our decision boundary is non-linear

Knowledge check 3

Exercise 2

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	✔
Summarize the key difference between support vector classifier and support vector machine	✔
Build a support vector machine model to classify the Costa Rica dataset	✔
Optimize the support vector machine model using grid search	

Optimize the SVM model with grid search

- **Grid search** gives a range of n values to a particular parameter in the model and lets the model choose the **best tuning parameter from the best model**
- Here, we give the range of values to both `C` and `gamma` parameters to chose the best tuning parameter and do 5 fold cross-validation

Grid search and cv on SVM model

```
# Set the parameters by cross-validation.
tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],
                     'C': [1, 10, 100, 1000]}]

# Fit the tuned parameters for grid search to SVM.
svm_cv = GridSearchCV(SVC(), tuned_parameters, cv = 5)
svm_cv.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='auto_deprecated', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='warn', n_jobs=None,
             param_grid=[{'C': [1, 10, 100, 1000], 'gamma': [0.001, 0.0001],
                          'kernel': ['rbf']}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

Find best parameters

```
# Find the best tuned parameters.
print(svm_cv.best_params_)
```

```
{'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
```

```
# Extract the best hyperparameters.
optimized_c = svm_cv.best_params_['C']
optimized_gamma = svm_cv.best_params_['gamma']
optimized_kernel = svm_cv.best_params_['kernel']
```

Fit the best parameters to build the optimized model

```
# Run the model with optimized hyperparameters.
sv_cv_optimized = SVC(kernel = optimized_kernel,
gamma = optimized_gamma, C = optimized_c)

sv_cv_optimized.fit(X_train, y_train)
```

```
SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.001, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

SVM accuracy of optimized model

```
# Predict on the test data.
y_pred = sv_cv_optimized.predict(X_test)
y_pred[0:5]
```

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

```
# Find the accuracy value for SVM model.
svm_cv_accuracy = metrics.accuracy_score(y_test, y_pred)
svm_cv_accuracy
```

```
0.8152022315202232
```

Print confusion matrix

```
# Print the confusion matrix.
confusion_matrix(y_test, y_pred)
```

```
array([[1561,  245],
       [ 285,  777]])
```

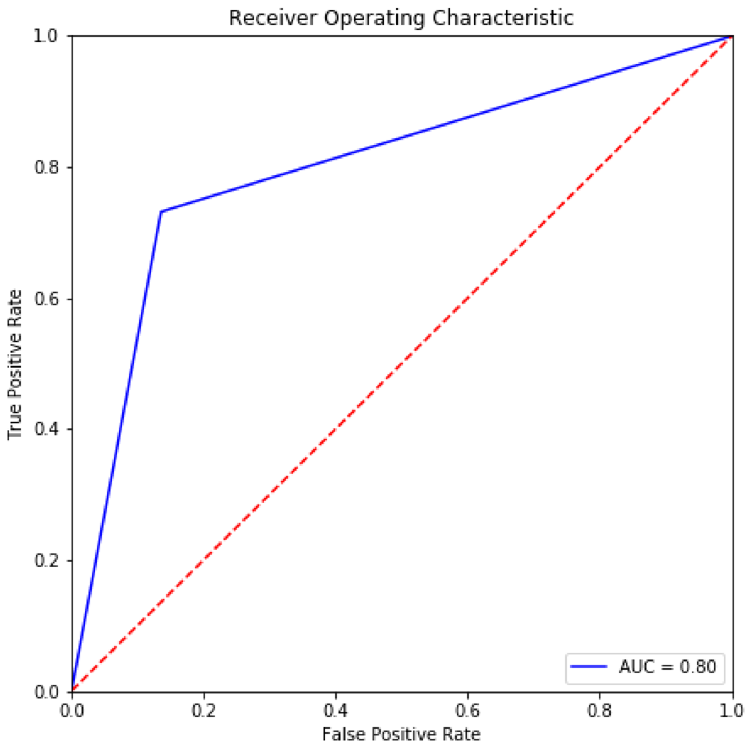
Print classification report

```
# Print an entire classification report.
class_report = metrics.classification_report(y_test,
                                              y_pred)
print(class_report)
```

```
              precision    recall  f1-score   support

       False       0.85      0.86      0.85      1806
        True       0.76      0.73      0.75      1062

    accuracy                           0.82      2868
   macro avg       0.80      0.80      0.80      2868
weighted avg       0.81      0.82      0.81      2868
```

Plot ROC curve

```
# Calculate metrics for ROC (fpr, tpr) and
calculate AUC.
fpr, tpr, threshold = metrics.roc_curve(y_test,
y_pred)
roc_auc = metrics.auc(fpr, tpr)

# Plot ROC.
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' %
roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



SVM performance

- Although ensemble classifiers outperformed SVM, it still performs better than most of the single classifier models like logistic, kNN, and decision trees
- This is because we have a lot of categorical variables and SVM models them better than the other models
- Do you have any data with more categorical variables from your work?
- Try building the classifier on your data and see how it performs compared to other models

Save final accuracy of SVM

```
# Add the model to our dataframe.
svc_model = svc_model.append({'metrics' : "accuracy" , 'values' :round(svm_cv_accuracy,4),
'model':'svm_optimized' } ,  ignore_index = True)
print(svc_model)
```

```
     metrics  values                        model
0   accuracy  0.6046                        knn_5
1   accuracy  0.6188             knn_GridSearchCV
2   accuracy  0.6287                       knn_29
3   accuracy  0.6356                     logistic
4   accuracy  0.7845      logistic_whole_dataset
5   accuracy  0.7859              logistic_tuned
6   accuracy  0.6611          tree_simple_subset
7   accuracy  0.9407          tree_all_variables
8   accuracy  0.7183 tree_all_variables_optimized
9   accuracy  0.9338               random forest
10  accuracy  0.8644                    boosting
11  accuracy  0.8536            optimized forest
12  accuracy  0.8563               gbm_optimized
13  accuracy  0.7723                         svc
14  accuracy  0.7737                         svm
15  accuracy  0.8152               svm_optimized
```

- The accuracy of our optimized SVM has increased here since we found the optimal tuning parameters and ran it with a cross-validation

Advantages and disadvantages of SVM

- **Advantages**
  - It is an effective classifier for **high dimensional data**
  - **Memory efficient** since only the support vectors are needed to be stored in memory for making the classification decision
  - Versatile by allowing **different boundary conditions**
- **Disadvantages**
  - It is **non-probabilistic**, so there is no probabilistic interpretation for group membership
  - When $p > n$, i.e. when number of predictors exceeds the number of observations, SVM performs poorly

Knowledge check 4

Exercise 3

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	✔
Summarize the key difference between support vector classifier and support vector machine	✔
Build a support vector machine model to classify the Costa Rica dataset	✔
Optimize the support vector machine model using grid search	✔

Workshop!

- Workshops are to be completed in the afternoon either with a dataset for a capstone project or with another dataset of your choosing
- Make sure to annotate and comment your code so that it is easy for others to understand what you are doing
- This is an exploratory exercise to get you comfortable with the content we discussed today
- Today you will:
  - Load in your cleaned dataset
  - Use SVC and SVM models to build classifiers
  - Predict on your test set and check the results
  - Use grid search and find the optimal parameters for your best model

This completes our module

Congratulations!