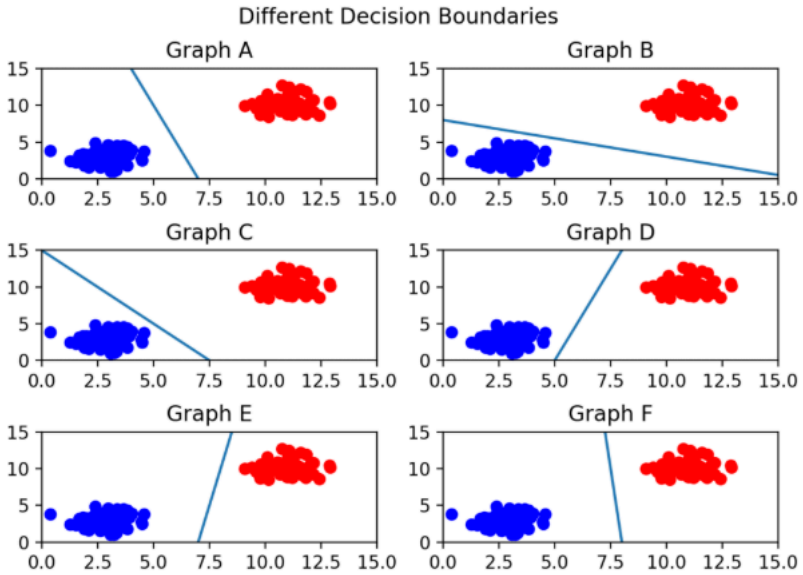
SVM

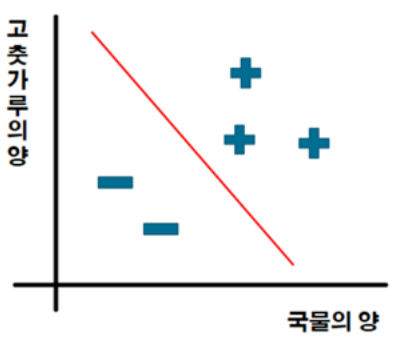
- 분류와 회귀 문제에 사용할 수 있는 강력한 머신러닝 지도학습 모델
  - 결정 경계(Decision Boundary), 즉 분류를 위한 기준 선을 정의하는 모델
    - 분류되지 않은 새로운 점이 나타나면 어느 쪽에 속하는지 확인해서 분류 과제를 수행할 수 있게 됨
  - 결정 경계와 가까이 있는 데이터 포인트들을 의미
- 결국 이 결정 경계라는 걸 어떻게 정의하고 계산하는지 이해하는게 더 중요하다는 뜻임!!!!:::

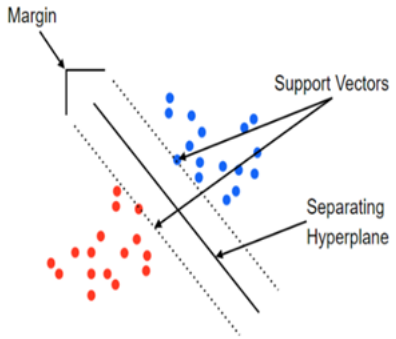


생각 문제

- 빨간색, 파란색 label을 분류하는 선형(Linear)을 찾는다고 가정해보자!!!
  - 기본적으로 SVM은 두 개의 다른 클래스를 분류할 수 있는 여러가지의 선형식이 있을테지만 이 중에서 가장 잘 분류할 수 있는 하나의(Unique) 선형식을 찾으려고 함

    - 그렇다면 가장 잘 분류할 수 있는 기준은 무엇일까?
- 그림에서 Separating Hyperplane(일종의 Decision boundary)가 두 개의 클래스를 가장 잘 분류할 수 있는 선형식을 찾는것
    - 이 Decision boundary를 찾기 위한 조건이 있음?
        - 여러가지의 선형식 후보들중에서 Support Vector들과의 Margin이 가장 Maximium이 되게해야 함
        - 이때 Support Vector는 반드시 최소 2개 이상의 데이터가 존재해야 함



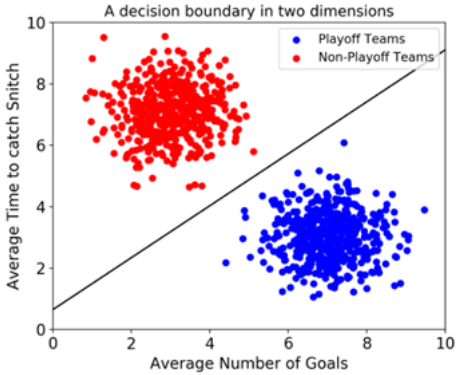


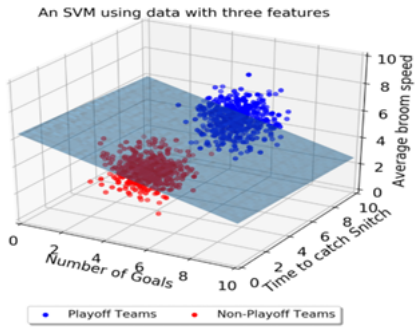
Support Vector

- 두 가지 카테고리에 각각 해당되는 data set들('-'샘플이 모인곳/'+'샘플이 모인곳)의 최외각에 있는 샘플들을 의미
    - 이 가장 최외각에 있는 벡터들을 토대로 margin을 구할 수 있기 때문에 중요한 벡터들임

하이퍼플레인(hyperplane)

- N차원 공간을 두 부분으로 나누는 N-1차원의 부분공간
    - 2차원 공간(평면)에서 하이퍼플레인은 1차원의 선
    - 3차원 공간에서는 2차원의 평면
- 머신러닝과 특히 서포트 벡터 머신(SVM)에서 중요한 개념으로, 하이퍼플레인은 데이터 클래스를 구분하는 결정 경계로 사용됨





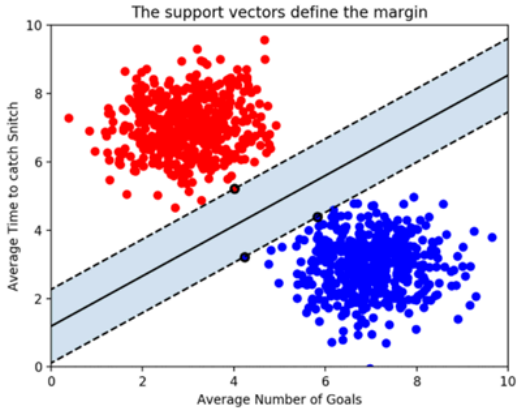
- 하이퍼플레인의 정의는 주로 선형 방정식을 통해 이루어짐
  - 다음과 같은 방정식으로 표현될 수 있음
    - w는 가중치 벡터, x는 입력 벡터, b는 편향

$$w \cdot x + b = 0$$

- 하이퍼플레인은 고차원 데이터를 분류할 때 중요한 역할을 함
  - 특히, 서포트 벡터 머신에서는 가장 가까운 데이터 포인트(서포트 벡터) 사이의 마진을 최대화하는 하이퍼플레인을 찾음으로써, 두 클래스를 가장 잘 구분할 수 있는 결정 경계를 찾음
  - 이러한 성질 때문에 하이퍼플레인은 패턴 인식, 이미지 분석, 바이오인포매틱스 등 다양한 머신러닝 응용 분야에서 활용됨

마진(Margin)

- 결정 경계와 서포트 벡터 사이의 거리를 의미



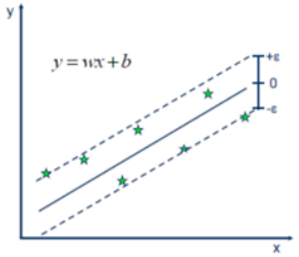
✓ x축과 y축 2개의 속성을 가진 데이터로 결정 경계를 그었는데, 총 3개의 데이터 포인트(서포트 벡터)가 필요함

✓ 즉, **n개의 속성을 가진 데이터에는 최소 n+1개의 서포트 벡터가 존재한다**는 걸 알 수 있음

✓ 대부분 머신러닝 지도 학습 알고리즘은 학습 데이터 모두를 사용하여 모델을 학습하는데 **SVM에서는 결정 경계를 정의하는 게 결국 서포트 벡터이기 때문에 데이터 포인트 중에서 서포트 벡터만 잘 골라내면 나머지 쓸 데 없는 수많은 데이터 포인트들을 무시할 수 있기 때문에 매우 빠름**

- 하드 마진 SVR 회귀모델

- 분류와는 다르게 예측에서 오차가 0이 되는 직선은 이론상으로도 존재하기 어려움
- 따라서 서포트 벡터 회귀에서는 마진을 최대화하면서 동시에 오차 범위 내($\pm\varepsilon$)로 샘플이 들어오도록 예측 직선을 구축함



- Solution:

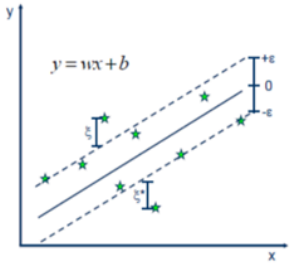
$$\min \frac{1}{2}\|w\|^2$$

- Constraints:

$$y_i - wx_i - b \le \varepsilon$$
$$wx_i + b - y_i \le \varepsilon$$

- 소프트 마진 SVR 회귀모델

- 그러나 마진을 최대화하면서 동시에 오차 범위 내($\pm\varepsilon$)로 샘플이 들어오도록 예측 직선도 거의 존재하지 않아, 추가적인 예측 오차(양의 오차 ξ_i와 음의 오차 ξ_i^*)를 허용함



- Minimize:

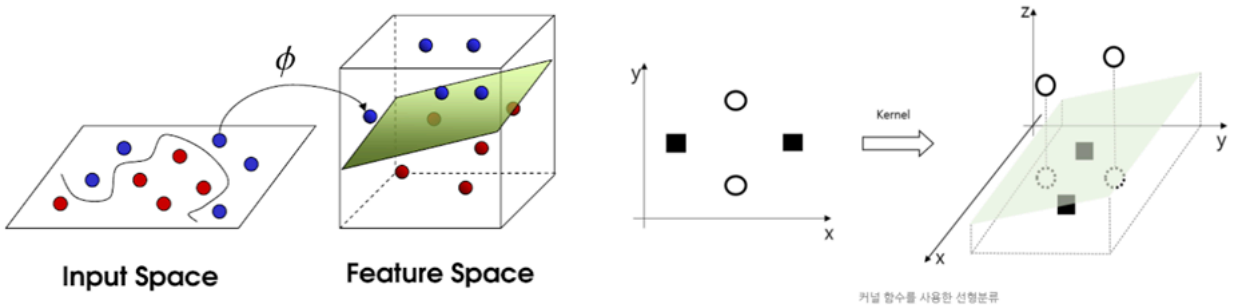
$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*)$$

- Constraints:

$$y_i - wx_i - b \le \varepsilon + \xi_i$$
$$wx_i + b - y_i \le \varepsilon + \xi_i^*$$
$$\xi_i, \xi_i^* \ge 0$$

커널 함수(kernel function)

- Support Vector Machine(SVM)과 같은 머신러닝 모델에서 사용되는 함수
- SVM(Support Vector Machine)은 두 개의 클래스를 가진 학습 데이터들을 구분하기 위해 두 범주 사이의 거리를 최대화 시키는 최적의 초평면을 찾는 이진분류기법이라 할 수 있음
- 일반적으로 분류하고 싶은 데이터는 비선형일 경우가 많기 때문에 SVM은 기본적으로 초평면을 사용하는 선형분류기법이지만 선형분류가 되지 않는 경우, 이를 해결하기 위해 그림과 같이 커널 함수를 통해 저차원의 데이터를 고차원으로 매핑하여 선형분류 가능하게 함



scikit-learn 라이브러리를 이용한 SVM 구현 - SVC(1)

```
#SVC는 SVM에서 M을 C(Classification)으로 바꾼 것, 회귀는 SVR(Regression)임
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear')
training_points=[[1,2],[1,6 ],[2, 2],[7,5],[9,4],[8,2]]
labels=[1,1,1,0,0,0] #레이블은 빨간 1, 파란 0
classifier.fit(training_points, labels)
```

```
SVC(kernel='linear')
```

```
print(classifier.predict([[3, 2]])) #[3,2]는 빨간 점 1로 분류
```

```
[1]
```

```
#서포트 벡터(결정 경계)를 정의하는 서포트 벡터를 확인하려면
#classifier.support_vectors_를 point해보면 됨
classifier.support_vectors_
```

```
array([[7., 5.],
       [1., 6.],
       [2., 2.]])
```

scikit-learn 라이브러리를 이용한 SVM 구현 - SVC(2)

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn import svm
from sklearn.datasets import make_blobs #from sklearn.datasets.samples_generator import make_blobs를 제공하지 않음
X, y = make_blobs(n_samples=40, centers=2, random_state=20)
```

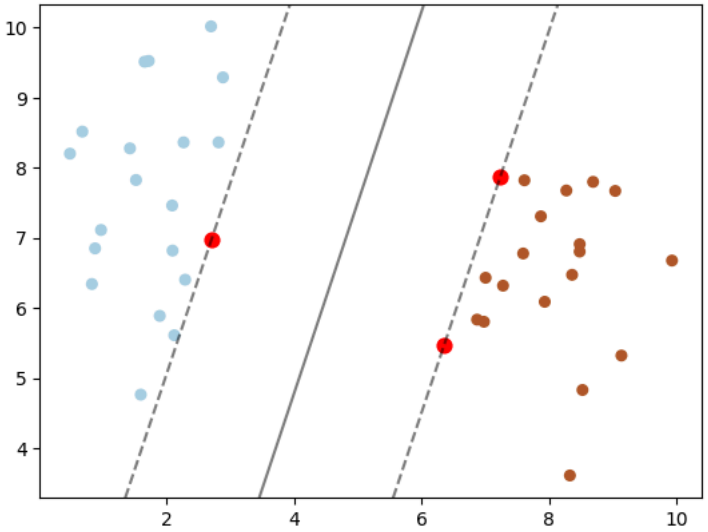
```
clf = svm.SVC(kernel='linear')
clf.fit(X, y)
```

```
SVC(kernel='linear')
```

```
newData = [[3,4]]
print(clf.predict(newData))
```

```
[0]
```

```
# 샘플 데이터 표현
plt.scatter(X[:,0], X[:,1], c=y, s=30, cmap=plt.cm.Paired)
# 초평면(Hyper-Plane) 표현
ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy).reshape(XX.shape)
ax.contour(XX, YY, Z, colors='k', levels=[-1,0,1], alpha=0.5, linestyles=['--', '-', '--'])
# 지지벡터(Support Vector) 표현
ax.scatter(clf.support_vectors_[:,0], clf.support_vectors_[:,1], s=60, facecolors='r')
plt.show()
```



scikit-learn 라이브러리를 이용한 SVM 구현 - SVR

```
#https://medium.com/@niousha.rf/support-vector-regressor-theory-and-coding-exercise-in-python-ca6a7dfda927
#Dataset
import pandas as pd

df = pd.read_csv('/content/drive/MyDrive/머신러닝&딥러닝/머신러닝-7-SVM과 결정트리/Student_Marks.csv')
df.head()
```

	number_courses	time_study	Marks
0	3	4.508	19.202
1	4	0.096	7.734
2	4	3.133	13.811
3	6	7.909	53.018
4	8	7.811	55.299

```
from sklearn.model_selection import train_test_split

train, test = train_test_split(df, test_size=0.2, random_state=42)

# train and test datasets are sorted for plotting purpose
train = train.sort_values('time_study')
test = test.sort_values('time_study')

X_train, X_test = train[['time_study']], test[['time_study']]
y_train, y_test = train['Marks'], test['Marks']
```

```
#Feature scaling
from sklearn.preprocessing import StandardScaler

### When using StandardScaler(), fit() method expects a 2D array-like input
scaler = StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

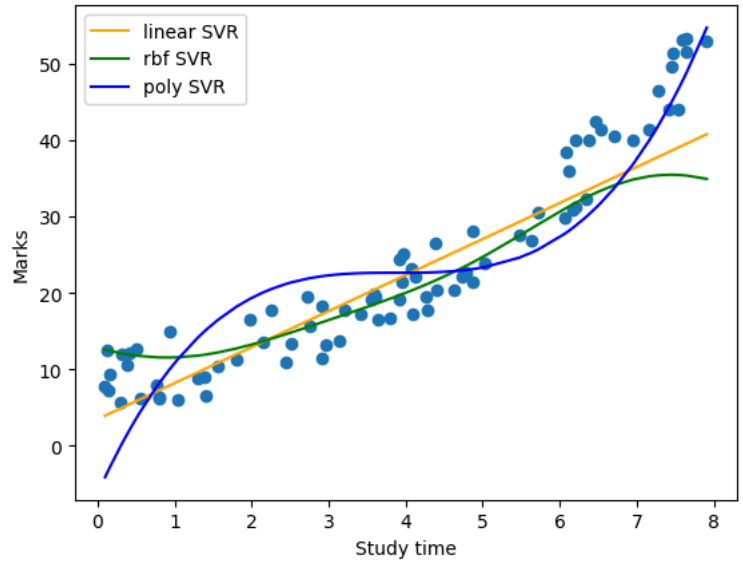
```
#Fitting SVR model
from sklearn.svm import SVR

svr_lin = SVR(kernel = 'linear')
svr_rbf = SVR(kernel = 'rbf')
svr_poly = SVR(kernel = 'poly')

svr_lin.fit(X_train_scaled, y_train)
```

```
svr_rbf.fit(X_train_scaled, y_train)
svr_poly.fit(X_train_scaled, y_train)
```

```
SVR
SVR(kernel='poly')
```

```
#Evaluating model performance
from matplotlib import pyplot as plt

#### Model prediction for train dataset ####
train['linear_svr_pred'] = svr_lin.predict(X_train_scaled)
train['rbf_svr_pred'] = svr_rbf.predict(X_train_scaled)
train['poly_svr_pred'] = svr_poly.predict(X_train_scaled)

#### Visualization ####
plt.scatter(train['time_study'], train['Marks'])
plt.plot(train['time_study'], train['linear_svr_pred'], color = 'orange', label = 'linear SVR')
plt.plot(train['time_study'], train['rbf_svr_pred'], color = 'green', label = 'rbf SVR')
plt.plot(train['time_study'], train['poly_svr_pred'], color = 'blue', label = 'poly SVR')
plt.legend()
plt.xlabel('Study time')
plt.ylabel('Marks')
```

```
Text(0, 0.5, 'Marks')
```




```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
```

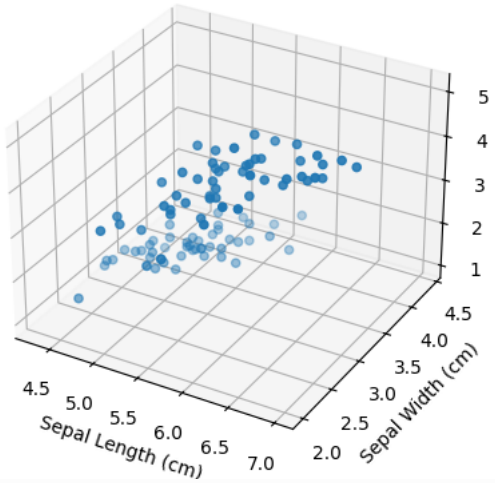
```
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
from sklearn.svm import LinearSVC
from sklearn.metrics import confusion_matrix
from mpl_toolkits.mplot3d import Axes3D
```

```
df_iris=pd.read_csv('/content/drive/MyDrive/머신러닝&딥러닝/머신러닝-7-SVM과 결정트리/Iris.csv')
df_iris.head()
#https://www.kaggle.com/code/xopxesalmon/svm-2dimension-hyperplane-visualization
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
#discriminate between Iris Setosa and Iris Versicolor using 3 features: Sepal Length, Sepal width and petal length.
df=df_iris.iloc[:100,1:4]

ax = plt.axes(projection='3d')
ax.scatter(df['SepalLengthCm'], df['SepalWidthCm'], df['PetalLengthCm'])
ax.set_xlabel('Sepal Length (cm)')
ax.set_ylabel('Sepal Width (cm)')
ax.set_zlabel('Petal Length (cm)')
```

```
Text(0.5, 0, 'Petal Length (cm)')
```



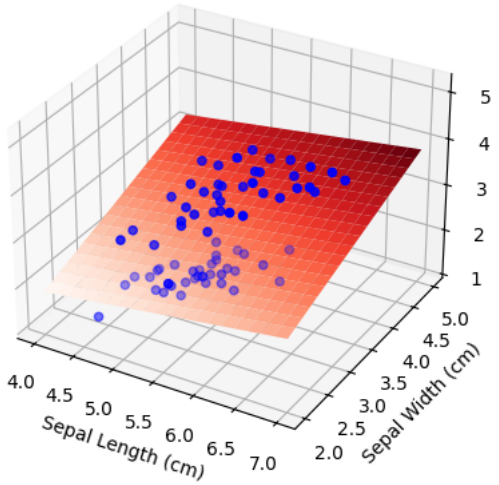
```
#Features and labels
# The support vector machines in scikit-learn support both dense (numpy.ndarray and convertible to that by numpy.asarray)
X=df.to_numpy()

# Converting string value to int type for labels: Setosa = 0, Versicolor = 1
y=df_iris.iloc[:100,-1]
y = LabelEncoder().fit_transform(y)
```

```
#Splitting of the dataset and raining the model
X_train, X_test, y_train, y_test = train_test_split(X,y, random_state=0)

svc = LinearSVC()
svc.fit(X_train, y_train)
```

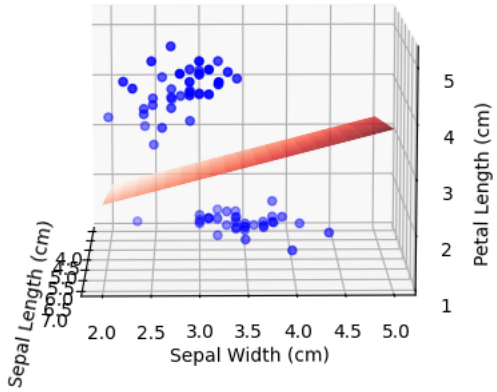
```
▾ LinearSVC ⓘ ⓘ
LinearSVC()
```

```
#Visualization of the hyperplane¶
#generates a plot of the vectors from X_train, and the SVM hyperplane.
plt.figure()
ax = plt.axes(projection='3d')
ax.scatter(X_train[:,0], X_train[:,1], X_train[:,2], c='b')
ax.set_xlabel('Sepal Length (cm)')
ax.set_ylabel('Sepal Width (cm)')
ax.set_zlabel('Petal Length (cm)')

zz = lambda xx,yy: (-svc.intercept_[0]-svc.coef_[0][0]*xx-svc.coef_[0][1]*yy) / svc.coef_[0][2]
tmpx = np.linspace(4, 7, 20)
tmpy = np.linspace(2, 5, 20)
xx,yy = np.meshgrid(tmpx,tmpy)
ax.plot_surface(xx, yy, zz(xx,yy), cmap='Reds')
```

```
<mpl_toolkits.mplot3d.art3d.Poly3DCollection at 0x78595f6958d0>
```




```
#The hyperplane shown from another angle:
plt.figure()
ax = plt.axes(projection='3d')
ax.scatter(X_train[:,0], X_train[:,1], X_train[:,2], c='b')
ax.set_xlabel('Sepal Length (cm)')
ax.set_ylabel('Sepal Width (cm)')
ax.set_zlabel('Petal Length (cm)')

zz = lambda xx,yy: (-svc.intercept_[0]-svc.coef_[0][0]*xx-svc.coef_[0][1]*yy) / svc.coef_[0][2]
tmpx = np.linspace(4, 7, 20)
tmpy = np.linspace(2, 5, 20)
xx,yy = np.meshgrid(tmpx,tmpy)
ax.plot_surface(xx, yy, zz(xx,yy), cmap='Reds')
for ii in range(0,360,1):
    ax.view_init(elev=10., azim=ii)
```




```
#And from another angle
plt.figure()
ax = plt.axes(projection='3d')
ax.scatter(X_train[:,0], X_train[:,1], X_train[:,2], c='b')
ax.set_xlabel('Sepal Length (cm)')
ax.set_ylabel('Sepal Width (cm)')
ax.set_zlabel('Petal Length (cm)')

zz = lambda xx,yy: (-svc.intercept_[0]-svc.coef_[0][0]*xx-svc.coef_[0][1]*yy) / svc.coef_[0][2]
tmpx = np.linspace(4, 7, 20)
tmpy = np.linspace(2, 5, 20)
xx,yy = np.meshgrid(tmpx,tmpy)
ax.plot_surface(xx, yy, zz(xx,yy), cmap='Reds')
for ii in range(0,30,1):
    ax.view_init(elev=25, azim=ii)
```

```
#Confusion Matrix:
y_pred = svc.predict(X_test)
confusion_matrix(y_test, y_pred)
```