Experiment No. 6 - Implement SVM classifier for classification of data into two classes.

Support Vector Machine

Support Vector Machine (SVM) is a supervised machine learning algorithm capable of performing classification, regression and even outlier detection. The linear SVM classifier works by drawing a straight line between two classes. All the data points that fall on one side of the line will be labeled as one class and all the points that fall on the other side will be labeled as the second.

Sounds simple enough, but there's an infinite amount of lines to choose from. How do we know which line will do the best job of classifying the data? This is where the LSVM algorithm comes in to play. The LSVM algorithm will select a line that not only separates the two classes but stays as far away from the closest samples as possible. In fact, the "support vector" in "support vector machine" refers to two position vectors drawn from the origin to the points which dictate the decision boundary.

- **Data** - x, y points
- **Task** - Linear Regression
- **Mathematical Model** - SVM
- **Loss Function**
- **Learning Algorithm** - Lagrangian
- **Model Evaluation** -

Algorithm

1. Import Library
2. SVM model
3. Load the data
4. Visualize the data
5. Train Test Split
6. Training the model
7. Visualize the result
8. Prediction
9. Using built-in SVM
10. Custom SVM vs built-in SVM

1. Import Library

In [1]:
```
import numpy as np
import cvxopt
from sklearn.datasets.samples_generator import make_blobs
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 IPython.display import Image
```

```
c:\users\hemant ghuge\anaconda3\envs\tensorflow2\lib\site-packages\sklearn\utils\deprecation.py:144: FutureWarning: The sklearn.datasets.samples_generator module is  deprecated in version 0.22 and will be removed in version 0.24. The corresponding classes / functions should instead be imported from sklearn.datasets. Anything that cannot be imported from sklearn.datasets is now part of the private API.
  warnings.warn(message, FutureWarning)

Bad key "text.kerning_factor" on line 4 in
c:\users\hemant ghuge\anaconda3\envs\tensorflow2\lib\site-packages\matplotlib\mpl-data\stylelib\_classic_test_patch.mplstyle.
You probably need to get an updated matplotlibrc file from
https://github.com/matplotlib/matplotlib/blob/v3.1.3/matplotlibrc.template
or from the matplotlib source distribution
```

2. SVM model

In [2]:
```
class SVM:
    def fit(self, X, y):
        n_samples, n_features = X.shape
# P = X^T X
        K = np.zeros((n_samples, n_samples))
        for i in range(n_samples):
            for j in range(n_samples):
                K[i,j] = np.dot(X[i], X[j])
        P = cvxopt.matrix(np.outer(y, y) * K)
# q = -1 (1xN)
        q = cvxopt.matrix(np.ones(n_samples) * -1)
# A = y^T
        A = cvxopt.matrix(y, (1, n_samples))
# b = 0
        b = cvxopt.matrix(0.0)
# -1 (NxN)
        G = cvxopt.matrix(np.diag(np.ones(n_samples) * -1))
# 0 (1xN)
        h = cvxopt.matrix(np.zeros(n_samples))
        solution = cvxopt.solvers.qp(P, q, G, h, A, b)
# Lagrange multipliers
        a = np.ravel(solution['x'])
# Lagrange have non zero lagrange multipliers
        sv = a > 1e-5
        ind = np.arange(len(a))[sv]
        self.a = a[sv]
        self.sv = X[sv]
        self.sv_y = y[sv]
# Intercept
        self.b = 0
        for n in range(len(self.a)):
            self.b += self.sv_y[n]
            self.b -= np.sum(self.a * self.sv_y * K[ind[n], sv])
        self.b /= len(self.a)
# Weights
        self.w = np.zeros(n_features)
        for n in range(len(self.a)):
            self.w += self.a[n] * self.sv_y[n] * self.sv[n]

    def project(self, X):
        return np.dot(X, self.w) + self.b

    def predict(self, X):
        return np.sign(self.project(X))
```

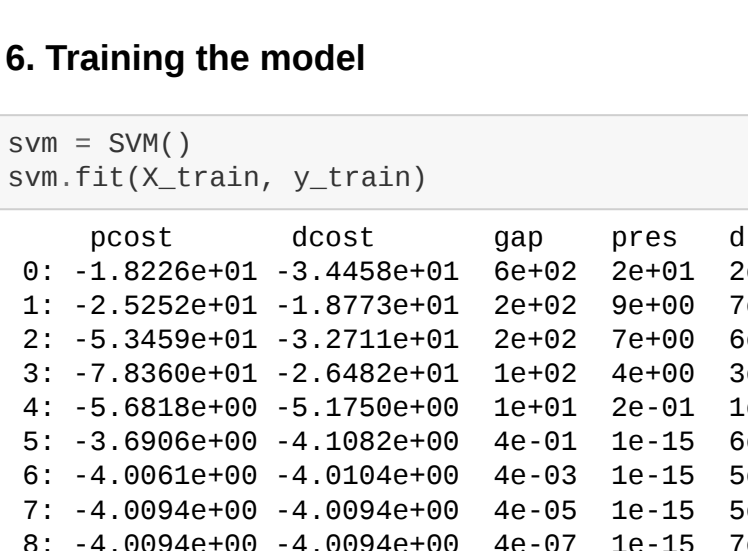
3. Load the data

In [3]:
```
X, y = make_blobs(n_samples=250, centers=2,
                  random_state=0, cluster_std=0.60)
y[y == 0] = -1
tmp = np.ones(len(X))
y = tmp * y
```

4. Visualize the data

In [4]:
```
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='winter')
```

Out[4]:
```
<matplotlib.collections.PathCollection at 0x230dc3e5f08>
```



5. Train Test Split

In [5]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

6. Training the model

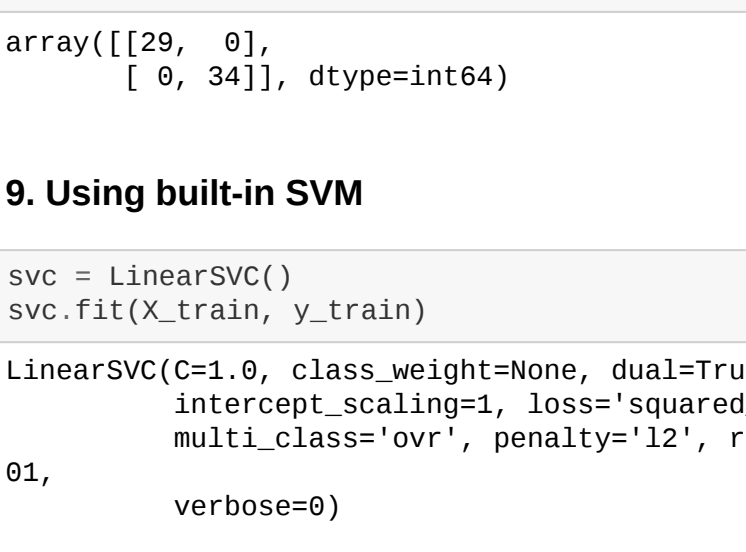
In [6]:
```
svm = SVM()
svm.fit(X_train, y_train)
```

```
     pcost       dcost       gap    pres   dres
 0: -1.8226e+01 -3.4458e+01  6e+02  2e+01  2e+00
 1: -2.5252e+01 -1.8773e+01  2e+02  9e+00  7e-01
 2: -5.3459e+01 -3.2711e+01  2e+02  7e+00  6e-01
 3: -7.8360e+01 -2.6482e+01  1e+02  4e+00  3e-01
 4: -5.6818e+00 -5.1750e+00  1e+01  2e-01  1e-02
 5: -3.6906e+00 -4.1082e+00  4e-01  1e-15  6e-15
 6: -4.0061e+00 -4.0104e+00  4e-03  1e-15  5e-15
 7: -4.0094e+00 -4.0094e+00  4e-05  1e-15  5e-15
 8: -4.0094e+00 -4.0094e+00  4e-07  1e-15  7e-15
Optimal solution found.
```

7. Visualize the result

In [7]:
```
def f(x, w, b, c=0):
    return (-w[0] * x - b + c) / w[1]
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='winter')
# w.x + b = 0
a0 = -4; a1 = f(a0, svm.w, svm.b)
b0 = 4; b1 = f(b0, svm.w, svm.b)
plt.plot([a0,b0], [a1,b1], 'k')
# w.x + b = 1
a0 = -4; a1 = f(a0, svm.w, svm.b, 1)
b0 = 4; b1 = f(b0, svm.w, svm.b, 1)
plt.plot([a0,b0], [a1,b1], 'k--')
# w.x + b = -1
a0 = -4; a1 = f(a0, svm.w, svm.b, -1)
b0 = 4; b1 = f(b0, svm.w, svm.b, -1)
plt.plot([a0,b0], [a1,b1], 'k--')
```

Out[7]:
```
[<matplotlib.lines.Line2D at 0x230dfa69d08>]
```



8. Prediction

In [8]:
```
y_pred = svm.predict(X_test)
confusion_matrix(y_test, y_pred)
```

Out[8]:
```
array([[29,  0],
       [ 0, 34]], dtype=int64)
```

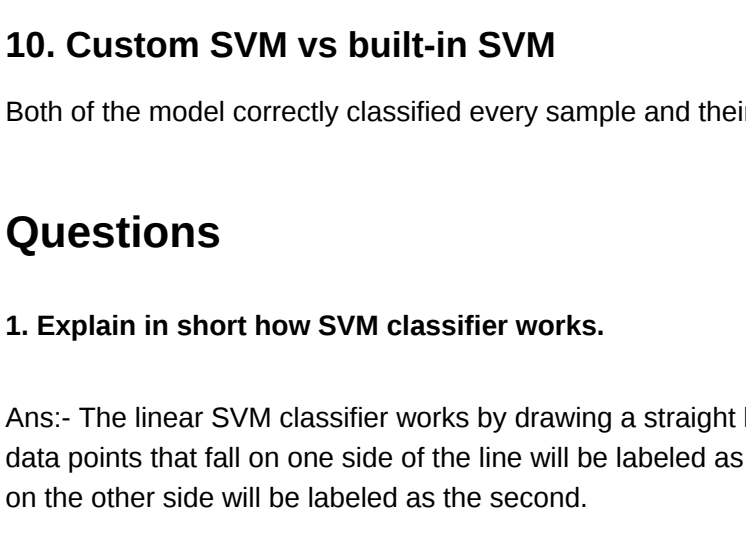
9. Using built-in SVM

In [9]:
```
svc = LinearSVC()
svc.fit(X_train, y_train)
```

Out[9]:
```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
```

In [10]:
```
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap='winter');
ax = plt.gca()
xlim = ax.get_xlim()
w = svc.coef_[0]
a = -w[0] / w[1]
xx = np.linspace(xlim[0], xlim[1])
yy = a * xx - svc.intercept_[0] / w[1]
plt.plot(xx, yy)
yy = a * xx - (svc.intercept_[0] - 1) / w[1]
plt.plot(xx, yy, 'k--')
yy = a * xx - (svc.intercept_[0] + 1) / w[1]
plt.plot(xx, yy, 'k--')
```

Out[10]:
```
[<matplotlib.lines.Line2D at 0x230dfaf6a08>]
```



In [11]:
```
y_pred = svc.predict(X_test)
confusion_matrix(y_test, y_pred)
```

Out[11]:
```
array([[29,  0],
       [ 0, 34]], dtype=int64)
```

10. Custom SVM vs built-in SVM

Both of the model correctly classified every sample and their confusion matrix are same.

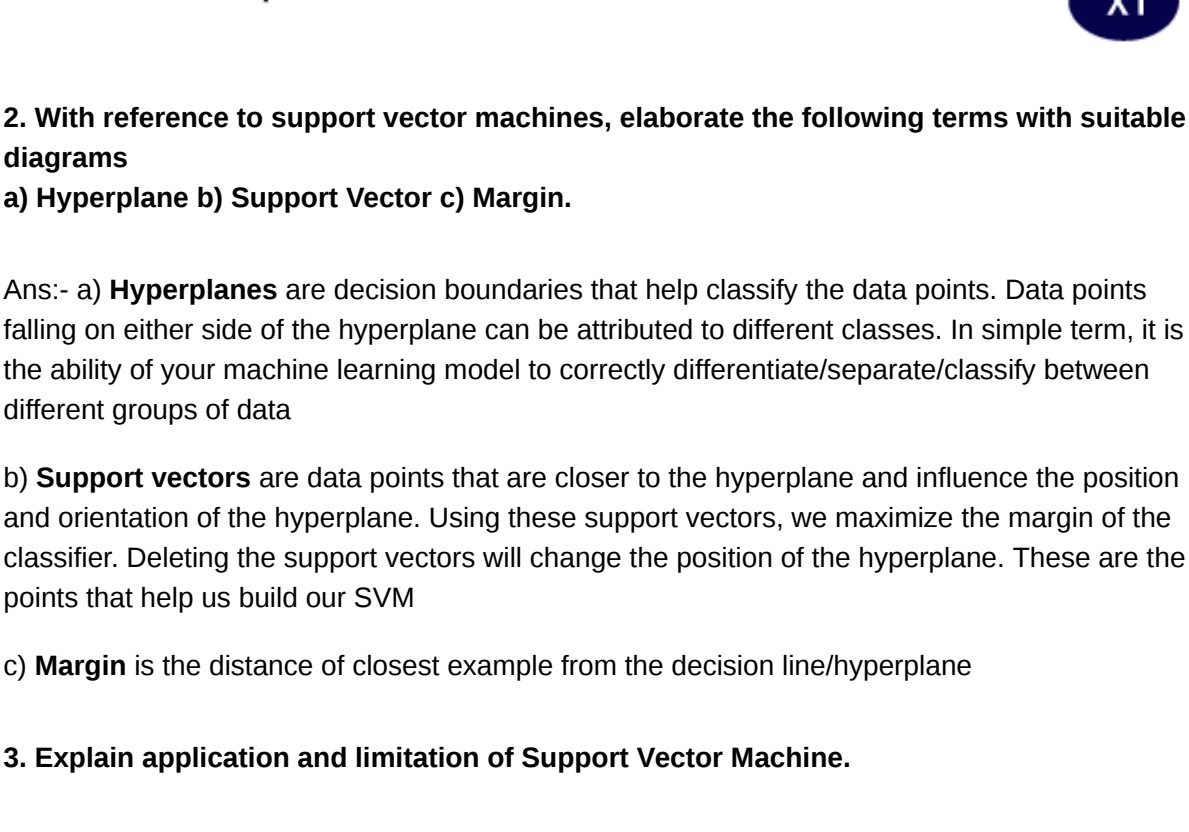
Questions

1. Explain in short how SVM classifier works.

Ans:- The linear SVM classifier works by drawing a straight line between two classes. All the data points that fall on one side of the line will be labeled as one class and all the points that fall on the other side will be labeled as the second.

In [12]:
```
Image(filename="img/svm.png")
```

Out[12]:



2. With reference to support vector machines, elaborate the following terms with suitable diagrams
a) Hyperplane b) Support Vector c) Margin.

Ans:- a) **Hyperplanes** are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. In simple term, it is the ability of your machine learning model to correctly differentiate/separate/classify between different groups of data

b) **Support vectors** are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM

c) **Margin** is the distance of closest example from the decision line/hyperplane

3. Explain application and limitation of Support Vector Machine.

Ans:- **Application**

- **Face detection** – SVMc classify parts of the image as a face and non-face and create a square boundary around the face.
- **Text and hypertext categorization** – SVMs allow Text and hypertext categorization for both inductive and transductive models. They use training data to classify documents into different categories. It categorizes on the basis of the score generated and then compares with the threshold value.
- **Classification of images** – Use of SVMs provides better search accuracy for image classification. It provides better accuracy in comparison to the traditional query-based searching techniques.
- **Bioinformatics** – It includes protein classification and cancer classification. We use SVM for identifying the classification of genes, patients on the basis of genes and other biological problems.
- **Protein fold and remote homology detection** – Apply SVM algorithms for protein remote homology detection.
- **Handwriting recognition** – We use SVMs to recognize handwritten characters used widely.
- **Generalized predictive control(GPC)** – Use SVM based GPC to control chaotic dynamics with useful parameters.

Limitation

- It doesn't perform well when we have large data set because the required training time is higher
- It also doesn't perform very well, when the data set has more noise i.e. target classes are overlapping
- SVM doesn't directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. It is included in the related SVC method of Python scikit-learn library.

4. Explain key properties of SVM.

Ans:-

- Flexibility in choosing a similarity function
- Sparseness of solution when dealing with large data sets
- Ability to handle large feature spaces
- Overfitting can be controlled by soft margin approach
- Nice math property: a simple convex optimization problem which is guaranteed to converge to a single global solution
- Feature Selection

5. How SVM classifier can be used for regression?

Ans:- Support Vector Machine can also be used as a regression method, maintaining all the main features that characterize the algorithm (maximal margin). The Support Vector Regression (SVR) uses the same principles as the SVM for classification, with only a few minor differences.

First of all, because output is a real number it becomes very difficult to predict the information at hand, which has infinite possibilities. In the case of regression, a margin of tolerance (epsilon) is set in approximation to the SVM which would have already requested from the problem.

But besides this fact, there is also a more complicated reason, the algorithm is more complicated therefore to be taken in consideration. However, the main idea is always the same: to minimize error, individualizing the hyperplane which maximizes the margin, keeping in mind that part of the error is tolerated.

References

[1] https://towardsdatascience.com/support-vector-machine-python-example-d67d9b63f1c8

[2] https://github.com/adityajn105/SVM-From-Scratch

[3] https://pythonprogramming.net/machine-learning-tutorial-python-introduction/#google_vignette

[4] https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/

[5] https://towardsdatascience.com/svm-implementation-from-scratch-python-2db2fc52e5c2

[6] https://medium.com/deep-math-machine-learning-ai/chapter-3-1-svm-from-scratch-in-python-86f93f853dc

Author Name:- Hemant Ghuge
LinkedIn:- https://www.linkedin.com/in/hemantghuge/
GitHub:- https://github.com/HemantGorakshGhuge