WEEK4_HW_Minju_Jo

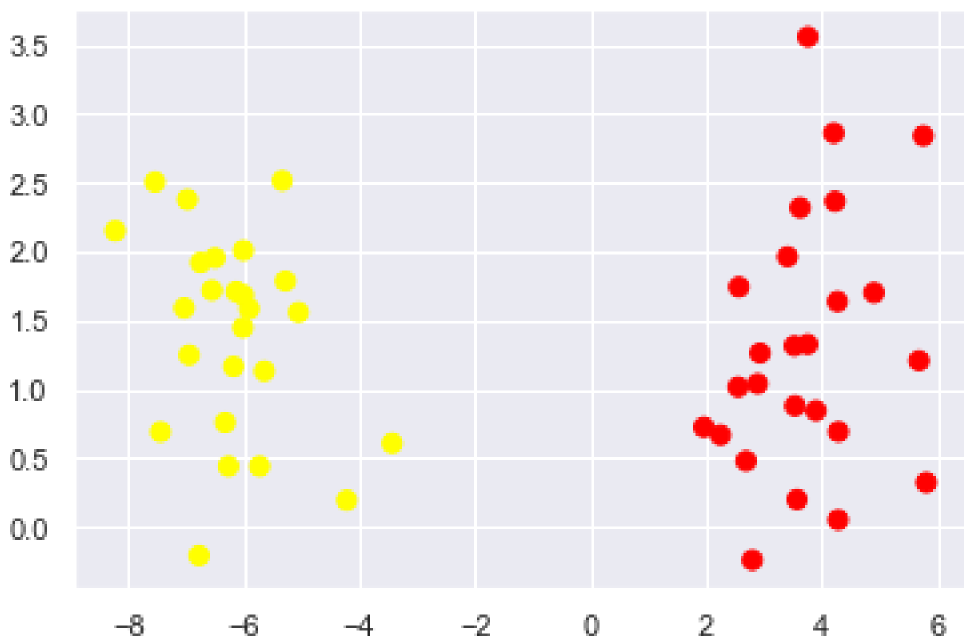
May 7, 2020

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        from scipy import stats
        import seaborn as sns; sns.set()
```

0.0.1 Linear SVM

```
In [2]: from sklearn.datasets import make_blobs
        X, y = make_blobs(n_samples = 50, centers = 2, cluster_std = 1)
        plt.scatter(X[:,0],X[:,1], c=y, s=50, cmap ='autumn')
```

```
Out[2]: <matplotlib.collections.PathCollection at 0x26c958a2d68>
```

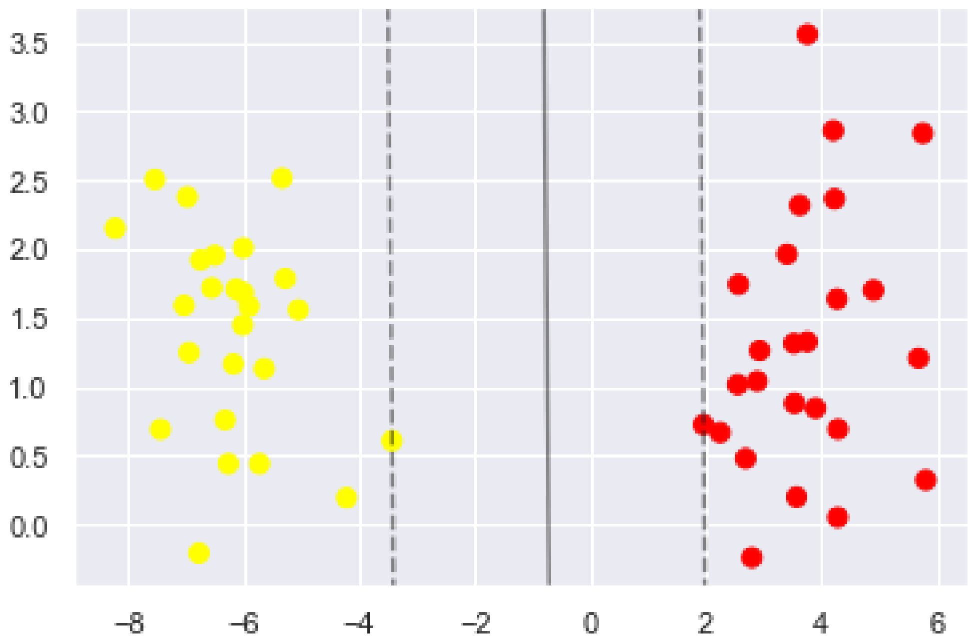



```
In [3]: from sklearn.svm import SVC
        model = SVC(kernel = 'linear', C=1E10)
        model.fit(X,y)
```

```
Out[3]: SVC(C=10000000000.0, 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)
```

```
In [4]: def plot_svc_decision_function(model, ax=None, plot_support=True):
            """ Plot the decision function for a 2D SVC"""
            if ax is None:
                ax = plt.gca()
            xlim = ax.get_xlim()
            ylim = ax.get_ylim()

            # create grid to evaluate model
            x = np.linspace(xlim[0], xlim[1], 30)
            y = np.linspace(ylim[0], ylim[1], 30)
            Y, X = np.meshgrid(y, x)
            xy = np.vstack([X.ravel(), Y.ravel()]).T
            P = model.decision_function(xy).reshape(X.shape)

            # plot decision boundary and margins
            ax.contour(X, Y, P, colors='k',
                       levels=[-1, 0, 1], alpha=0.5,
                       linestyles=['--', '-', '--'])

            # plot support vectors
            if plot_support:
                ax.scatter(model.support_vectors_[:, 0],
                           model.support_vectors_[:, 1],
                           s=300, linewidth=1, facecolors='none')

            ax.set_xlim(xlim)
            ax.set_ylim(ylim)
```

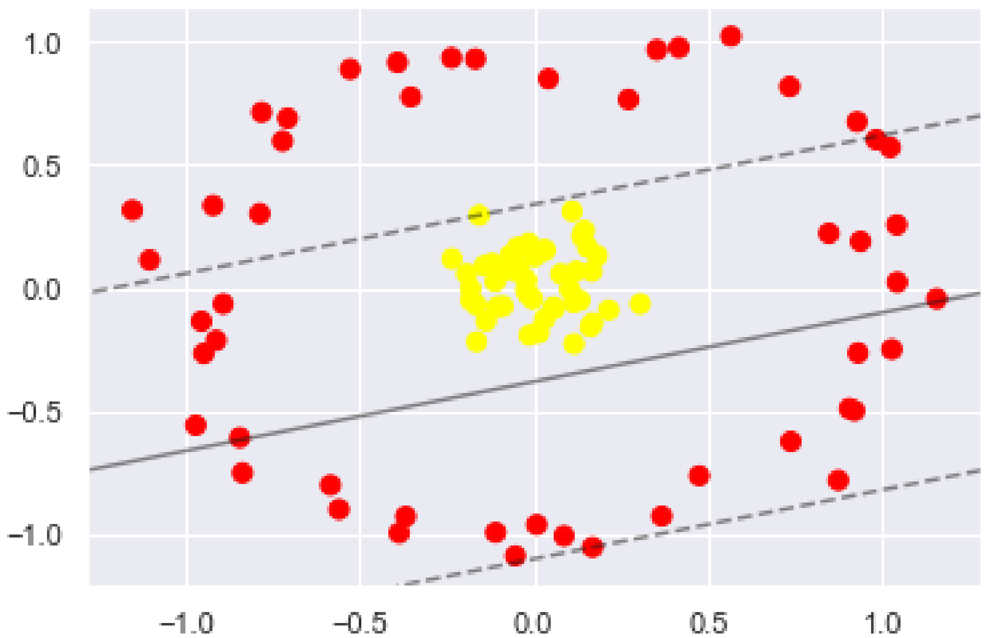
```
In [5]: plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
        plot_svc_decision_function(model)
```




```
In [6]: model.support_vectors_

Out[6]: array([[ 1.95484141,  0.72598014],
               [-3.43760173,  0.60866665]])
```

0.0.2 Kernel SVM

```
In [7]: from sklearn.datasets import make_circles
        X, y = make_circles(100, factor=.1, noise=.1)

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

        plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
        plot_svc_decision_function(clf, plot_support=False);
```



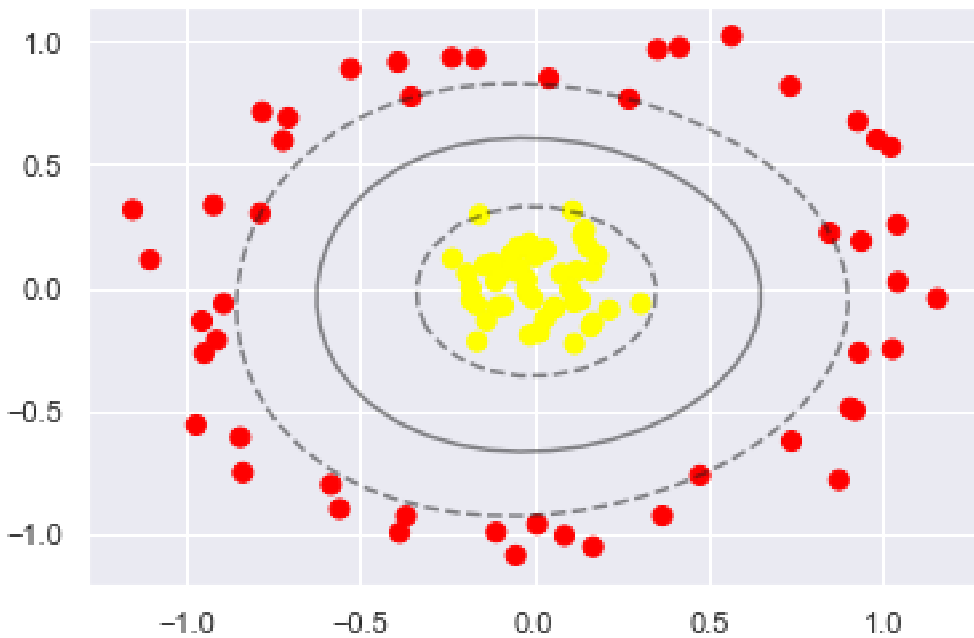
In [8]:
```
clf = SVC(kernel='rbf', C=1E6)
clf.fit(X, y)
```

```
C:\Users\MINJU\Anaconda3\lib\site-packages\sklearn\svm\base.py:196: FutureWarning: The default
  "avoid this warning.", FutureWarning)
```

Out[8]:
```
SVC(C=1000000.0, 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)
```

In [9]:
```
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(clf)
plt.scatter(clf.support_vectors_[:, 0], clf.support_vectors_[:, 1],
            s=300, lw=1, facecolors='none')
```

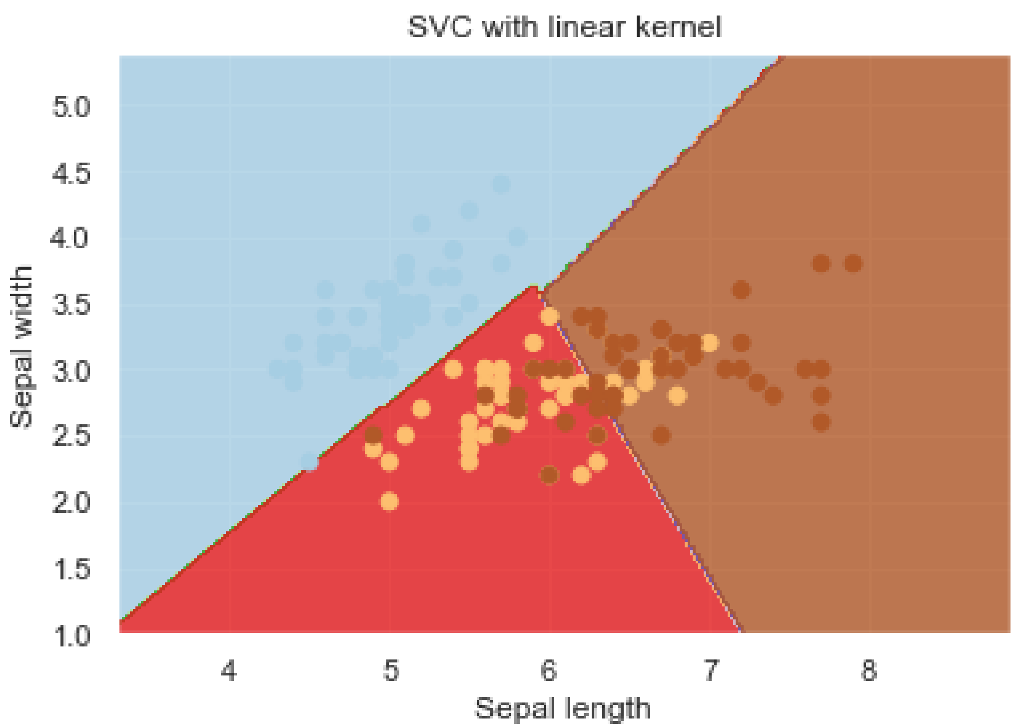
Out[9]:
```
<matplotlib.collections.PathCollection at 0x26c95cbcac8>
```



0.1 HW1 - Multiclass SVM: iris dataset

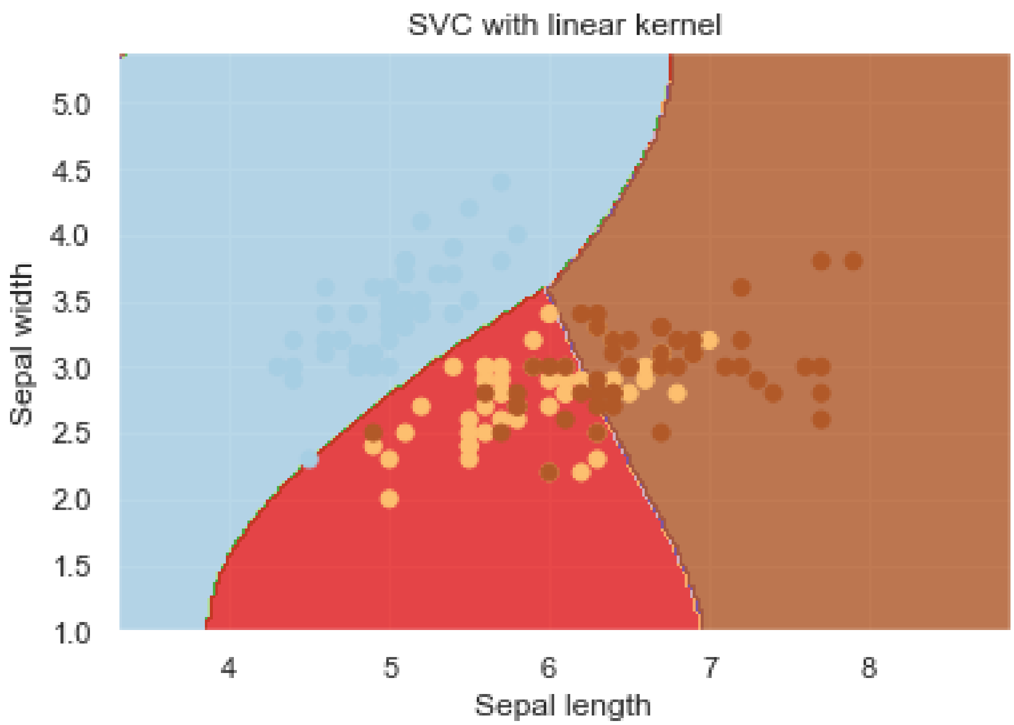
```
In [10]: import numpy as np
         import matplotlib.pyplot as plt
         from sklearn import svm, datasets

In [11]: # import some data to play with
         iris = datasets.load_iris()
         X = iris.data[:, :2] # we only take the first two features. We could
          # avoid this ugly slicing by using a two-dim dataset
         y = iris.target

In [12]: # we create an instance of SVM and fit out data. We do not scale our
         # data since we want to plot the support vectors
         C = 1.0 # SVM regularization parameter
         svc = svm.SVC(kernel='linear', C=1, gamma='auto').fit(X, y)

In [13]: # create a mesh to plot in
         x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
         y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
         h = (x_max / x_min)/100
         xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))

In [14]: plt.subplot(1, 1, 1)
         Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
         Z = Z.reshape(xx.shape)
```

```
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)

plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('SVC with linear kernel')
plt.show()
```

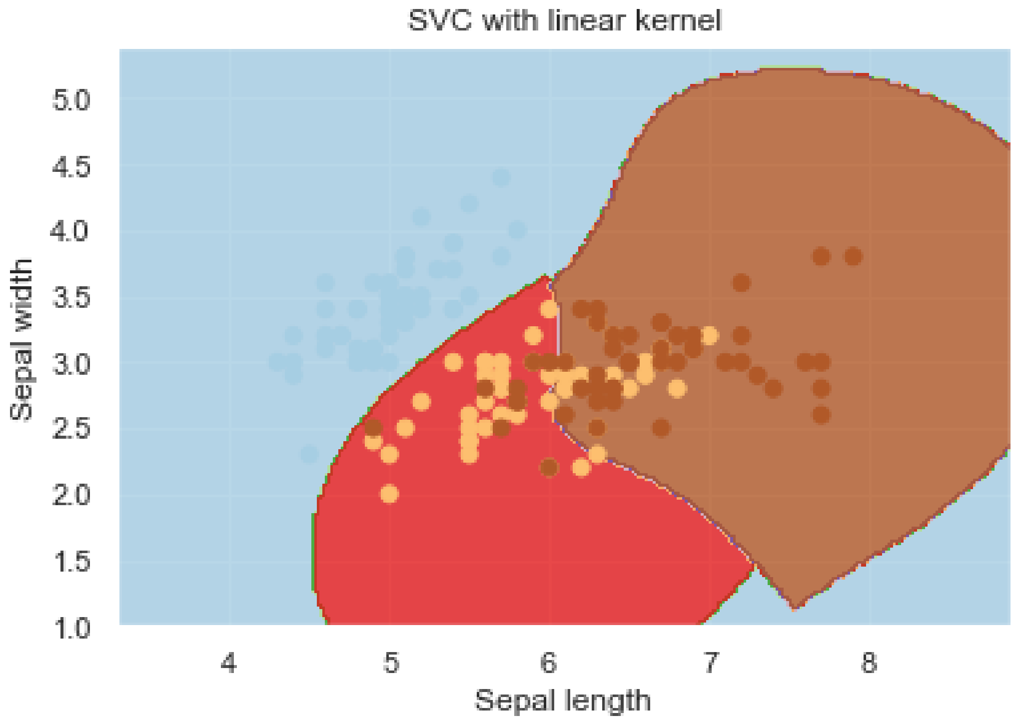



```
In [15]: svc = svm.SVC(kernel='rbf', C=1,gamma='auto').fit(X, y)
```

```
In [16]: plt.subplot(1, 1, 1)
         Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
         Z = Z.reshape(xx.shape)
         plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)

         plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
         plt.xlabel('Sepal length')
         plt.ylabel('Sepal width')
         plt.xlim(xx.min(), xx.max())
         plt.title('SVC with linear kernel')
         plt.show()
```




```
In [17]: svc = svm.SVC(kernel='rbf', C=100 ,gamma='auto').fit(X, y)

In [18]: plt.subplot(1, 1, 1)
         Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
         Z = Z.reshape(xx.shape)
         plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)

         plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
         plt.xlabel('Sepal length')
         plt.ylabel('Sepal width')
         plt.xlim(xx.min(), xx.max())
         plt.title('SVC with linear kernel')
         plt.show()
```



0.2 HW2 - Weighted SVM: toy dataset

```
In [19]: from sklearn.datasets import make_classification

         # define dataset
         X, y = make_classification(n_samples=10000, n_features=2, n_redundant=0,
                                    n_clusters_per_class=1, weights=[0.99], flip_y=0, random_s
```

```
In [20]: from collections import Counter

         # summarize class distribution
         counter = Counter(y)
         print(counter)
```

```
Counter({0: 9900, 1: 100})
```

```
In [21]: # scatter plot of examples by class label
         for label, _ in counter.items():
             row_ix = np.where(y == label)[0]
             plt.scatter(X[row_ix, 0], X[row_ix, 1], label=str(label))
         plt.legend()
         plt.show()
```

In [22]:
```
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score

# define model
model = SVC(gamma='scale', class_weight='balanced')
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, X, y, scoring='roc_auc', cv=cv, n_jobs=-1)
# summarize performance
print('Mean ROC AUC: %.3f' % np.mean(scores))
```

```
Mean ROC AUC: 0.971
```

In [ ]: