Assignment09

January 16, 2018

0.0.1 Exercise Sheet 09

ExerciseH9.1: Deriving the C-SVM optimization problem

```
In [74]: import numpy as np
         import pandas as pd
         import random
         import math
         import matplotlib.pyplot as plt
         %matplotlib inline

         from sklearn.svm import SVC
```

Create training data set Generate data for class -1

```
In [75]: p = 40 # number of samples from normal distributions for each label value y=-1 and y=1

         mu1, sigma = np.asarray([0,1]), np.sqrt(0.1) # mean1 and standard deviation
         mu2, sigma = np.asarray([1,0]), np.sqrt(0.1) # mean2 and standard deviation

         n1 = np.random.normal(mu1, sigma, size=[p, 2])
         n2 = np.random.normal(mu2, sigma, size=[p, 2])

         ## np.mean(n1, axis = 0), np.mean(n2, axis = 0)

         choice = np.random.choice([0,1],p)
         sample1 = np.where(choice, n1.T, n2.T).T    # p(x|y = 0) = 0.5*[N(x|mu1,sigma) + N(x|mu2
         # sample1.mean(axis = 0)
```

Generate data for class 1

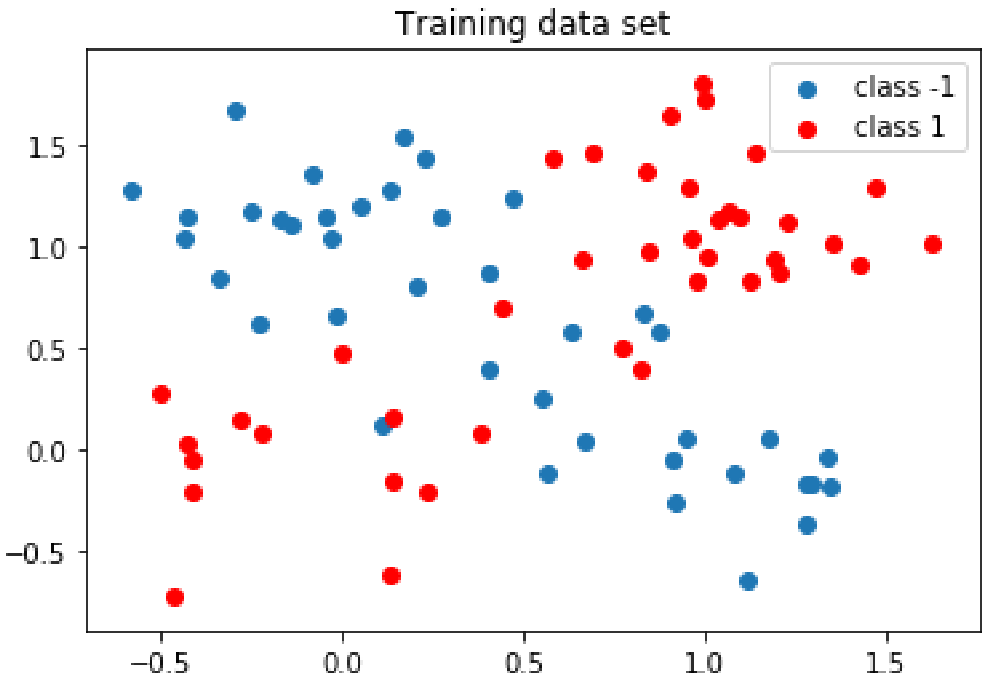
```
In [76]: mu3, sigma = np.asarray([0,0]), np.sqrt(0.1) # mean3 and standard deviation
         mu4, sigma = np.asarray([1,1]), np.sqrt(0.1) # mean4 and standard deviation

         n3 = np.random.normal(mu3, sigma, size=[p, 2])
         n4 = np.random.normal(mu4, sigma, size=[p, 2])

         ## np.mean(n3, axis = 0), np.mean(n4, axis = 0)
```

```
         choice = np.random.choice([0,1],p)
         sample2 = np.where(choice, n3.T, n4.T).T    # p(x|y = 1) = 0.5*[N(x|mu3,sigma) + N(x|mu4
```

```
In [77]: class0_data = np.concatenate((sample1.T, -np.ones((p, 1)).T), axis = 0).T
         class1_data = np.concatenate((sample2.T, np.ones((p, 1)).T), axis = 0).T
```

```
In [78]: plt.scatter(class0_data[:, 0], class0_data[:,1], label = "class -1")
         plt.scatter(class1_data[:, 0], class1_data[:,1], label = "class 1",color = "red")
         plt.title("Training data set")
         plt.legend()
```

```
Out[78]: <matplotlib.legend.Legend at 0x10a78d68>
```




```
In [79]: training = np.concatenate([class0_data, class1_data])
```

Create test data set Generate data for class -1

```
In [80]: p = 40 # number of samples from normal distributions for each label value y=-1 and y=1

         mu1, sigma = np.asarray([0,1]), np.sqrt(0.1) # mean1 and standard deviation
         mu2, sigma = np.asarray([1,0]), np.sqrt(0.1) # mean2 and standard deviation

         n1 = np.random.normal(mu1, sigma, size=[p, 2])
         n2 = np.random.normal(mu2, sigma, size=[p, 2])
```

```
## np.mean(n1, axis = 0), np.mean(n2, axis = 0)

choice = np.random.choice([0,1],p)
sample1 = np.where(choice, n1.T, n2.T).T    # p(x|y = 0) = 0.5*[N(x|mu1,sigma) + N(x|mu2
```

Generate data for class 1

```
In [81]: mu3, sigma = np.asarray([0,0]), np.sqrt(0.1) # mean3 and standard deviation
         mu4, sigma = np.asarray([1,1]), np.sqrt(0.1) # mean4 and standard deviation

         n3 = np.random.normal(mu3, sigma, size=[p, 2])
         n4 = np.random.normal(mu4, sigma, size=[p, 2])

         ## np.mean(n3, axis = 0), np.mean(n4, axis = 0)

         choice = np.random.choice([0,1],p)
         sample2 = np.where(choice, n3.T, n4.T).T    # p(x|y = 1) = 0.5*[N(x|mu3,sigma) + N(x|mu4
```

```
In [82]: class0_data = np.concatenate((sample1.T, -np.ones((p, 1)).T), axis = 0).T
         class1_data = np.concatenate((sample2.T, np.ones((p, 1)).T), axis = 0).T
```

```
In [83]: plt.scatter(class0_data[:, 0], class0_data[:,1], label = "class -1")
         plt.scatter(class1_data[:, 0], class1_data[:,1], label = "class 1",color = "red")
         plt.title("Testing data set")
         plt.legend()
```

```
Out[83]: <matplotlib.legend.Legend at 0x10cea8d0>
```

```
In [84]: testing = np.concatenate([class0_data, class1_data])
```

Train the C-SVM on the training data with RBF kernel and the software's standard parameters.

```
In [85]: Xtrain = training[:, :2]
         ytrain = training[:,2]
```

```
In [86]: clf = SVC()
         svc = clf.fit(Xtrain, ytrain)
```

```
In [87]: print("Training set accuracy = %0.2f" %svc.score(Xtrain, ytrain))
```

```
Training set accuracy = 0.90
```

```
In [88]: Xtest = testing[:,:2]
         ytest = testing[:,2]
```

```
In [89]: print("Total number of support vectors = %i" %svc.support_vectors_.shape[0])
```

```
Total number of support vectors = 46
```

```
In [90]: print("Number of support vectors for each class: ", svc.n_support_)
```

```
('Number of support vectors for each class: ', array([23, 23]))
```

Classify the test data and report the classication error quantied by the 0/1 loss function (percentage of wrong predictions).

```
In [91]: y_predicted = clf.predict(Xtest)
```

```
In [92]: classification_error = (1 - float(np.sum(np.equal(y_predicted, ytest)))/ytest.shape[0])
         print("Percentage of wrong predictions: %0.2f" %classification_error)
```

```
Percentage of wrong predictions: 18.75
```

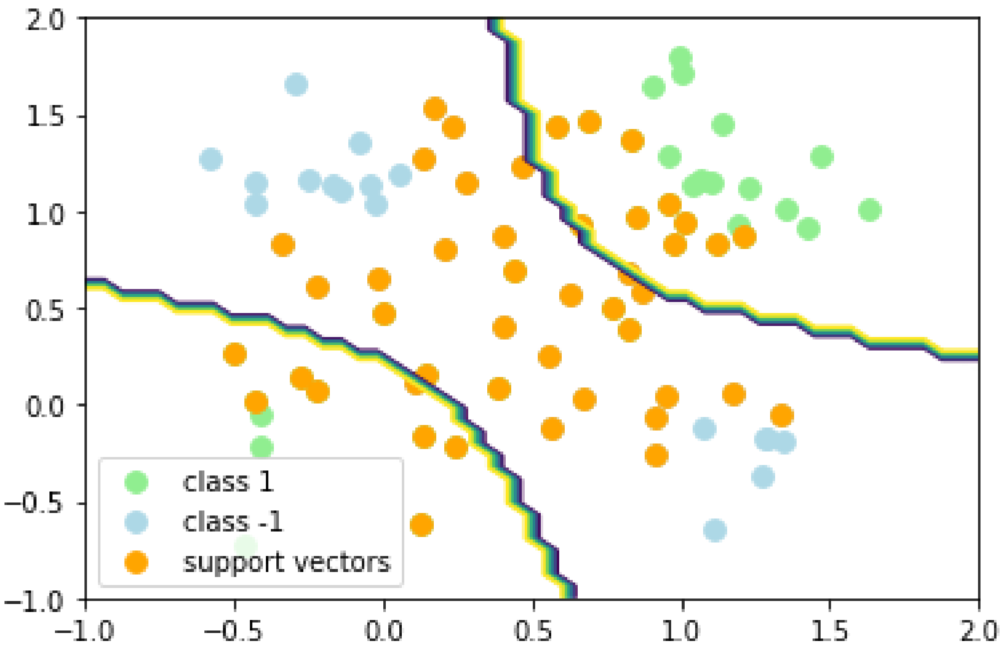
```
In [93]: print("The mean accuracy on the given test data and labels = %0.2f" %svc.score(Xtest, y
```

```
The mean accuracy on the given test data and labels = 0.81
```

Visualize the results as in exercise H7.2: plot the training patterns and the decision boundary (e.g. with a contour plot) in input space. Additionally, highlight the support vectors.

In [94]:
```
n = 50
x = np.linspace(-1, 2, n)
X, Y = np.meshgrid(x, x)
points = np.vstack([X.flatten(), Y.flatten()]).T



Z = np.zeros(shape=(n,n)).flatten()
Z = svc.predict(points)
Z = Z.reshape((n,n))


plt.contour(X, Y, Z)

class_1 = training[training[:,2]==1, :]
class_2 = training[training[:,2]==-1, :]

plt.scatter(class_1[:,0], class_1[:,1], s=60, color = 'lightgreen', label = "class 1")
plt.scatter(class_2[:,0], class_2[:,1], s=60, color = 'lightblue', label = "class -1")
plt.scatter(svc.support_vectors_[:,0], svc.support_vectors_[:,1], s=60, color = 'orange
plt.legend()
```

Out[94]: <matplotlib.legend.Legend at 0x10ef35c0>



ExerciseH9.3: C-SVM parameter optimization

In [95]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
```

In [96]:
```
# Set the parameters by cross-validation

gamma_exponents = np.arange(-5, 10, 2)
C_exponents = np.arange(-6, 11, 2)

gamma = [2*np.exp(exponent) for exponent in gamma_exponents]
C = [2*np.exp(exponent) for exponent in C_exponents]

tuned_parameters = [{'kernel': ['rbf'], 'gamma': gamma, 'C': C}]

svm = GridSearchCV(estimator = SVC(), param_grid= tuned_parameters, cv=4)
svm.fit(Xtrain, ytrain)
```

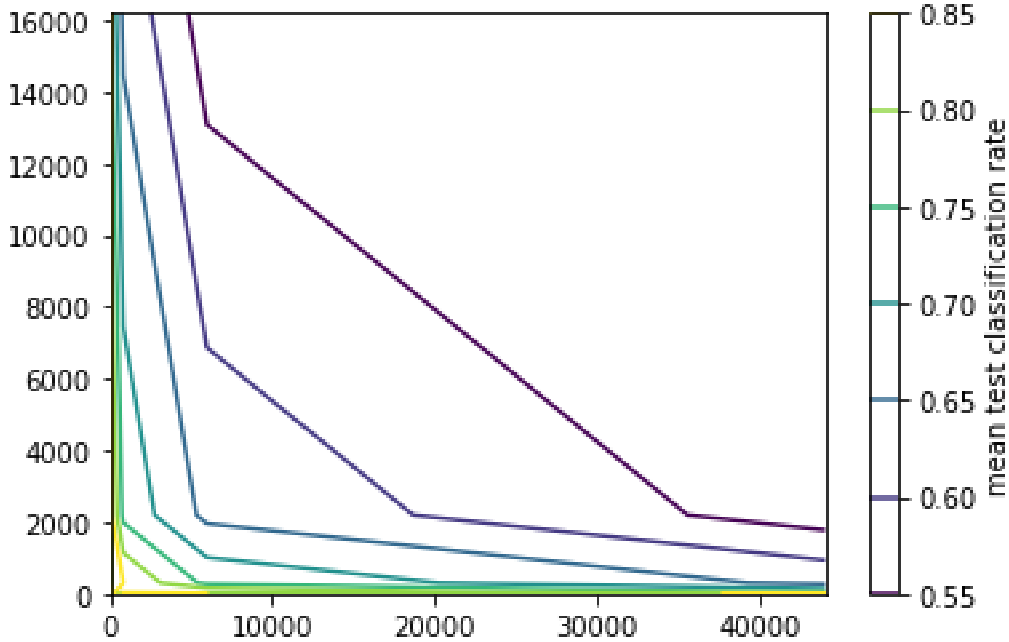
Out[96]:
```
GridSearchCV(cv=4, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid=[{'kernel': ['rbf'], 'C': [0.004957504353332717, 0.036631277777468357
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [97]:
```
X, Y = np.meshgrid(C, gamma)
points = np.vstack([X.flatten(), Y.flatten()]).T
```

In [98]:
```
X.shape, Y.shape, points.shape
```

Out[98]:
```
((8L, 9L), (8L, 9L), (72L, 2L))
```

In [332]:
```
Z = np.zeros(shape=(len(gamma), len(C))).flatten()
Z = svm.cv_results_['mean_test_score']
Z = Z.reshape((len(gamma), len(C)))


CS = plt.contour(X, Y, Z)
cbar = plt.colorbar(CS)
cbar.ax.set_ylabel('mean test classification rate')
plt.show()
```




```
In [126]: print('Best score on training set= %0.2f' %svm.best_score_)
```

```
Best score on training set= 0.90
```

```
In [127]: print('Best parameters:', svm.best_params_)
```

```
('Best parameters:', {'kernel': 'rbf', 'C': 14.778112197861301, 'gamma': 0.73575888234288467})
```

```
In [128]: # View the best parameters for the model found using grid search
          print('Best C:',svm.best_estimator_.C)
          print('Best Gamma:',svm.best_estimator_.gamma)
```

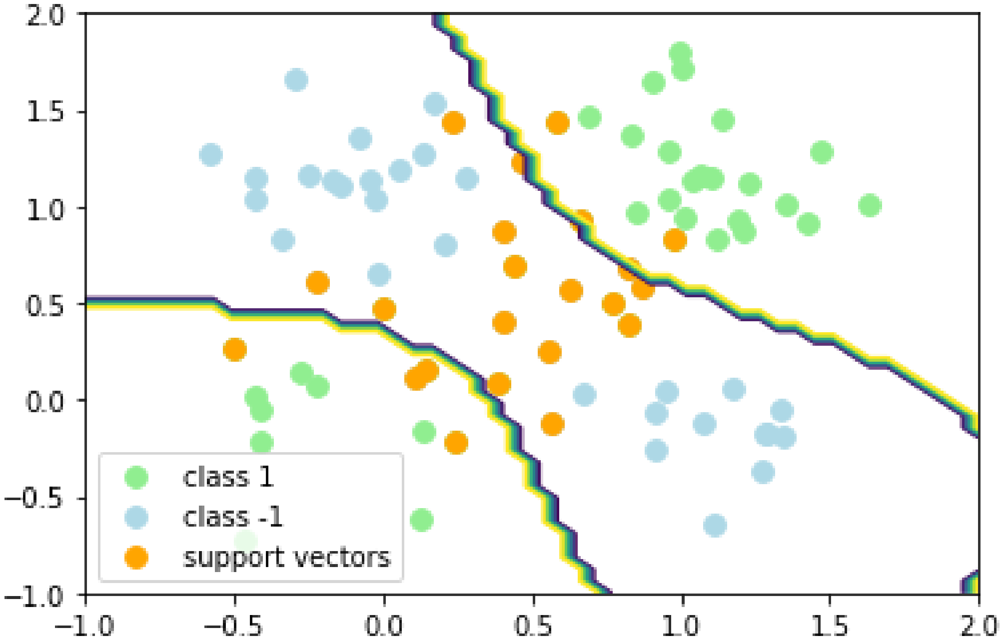
```
('Best C:', 14.778112197861301)
('Best Gamma:', 0.73575888234288467)
```

```
In [129]: new = SVC(C=svm.best_estimator_.C, kernel='rbf', gamma=svm.best_estimator_.gamma).fit(
```

```
In [130]: n = 50
          x = np.linspace(-1, 2, n)
          X, Y = np.meshgrid(x, x)
          points = np.vstack([X.flatten(), Y.flatten()]).T
```

```
Z = np.zeros(shape=(n,n)).flatten()
Z = new.predict(points)
Z = Z.reshape((n,n))


plt.contour(X, Y, Z)

class_1 = training[training[:,2]==1, :]
class_2 = training[training[:,2]==-1, :]

plt.scatter(class_1[:,0], class_1[:,1], s=60, color = 'lightgreen', label = "class 1")
plt.scatter(class_2[:,0], class_2[:,1], s=60, color = 'lightblue', label = "class -1")
plt.scatter(new.support_vectors_[:,0], new.support_vectors_[:,1], s=60, color = 'orang
plt.legend()
```

Out[130]: <matplotlib.legend.Legend at 0x11771898>



In [131]:
```
print("Percentage of wrong predictions on the testing set: %0.2f" %((1-class_performan
```

Percentage of wrong predictions on the testing set: 16.25

In [132]:
```
print("Accuracy= %0.2f" %class_performance)
```

Accuracy= 0.84

```
In [139]: print("Number of support vectors = %i" %new.support_vectors_.shape[0])
```

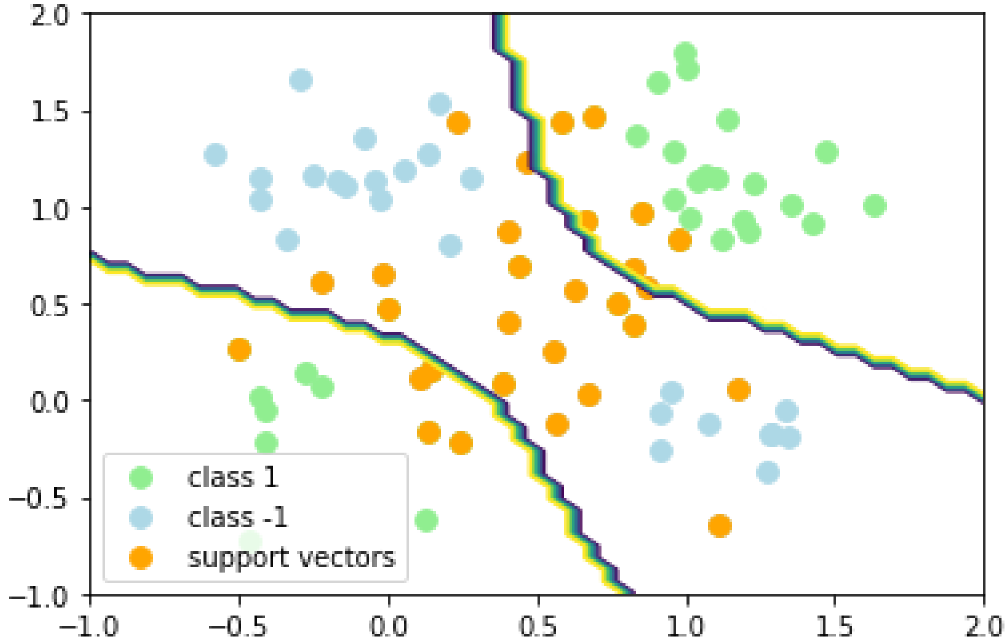
```
Number of support vectors = 22
```

What happens when you divide C or by 4?

```
In [134]: new1 = SVC(C=svm.best_estimator_.C/4, kernel='rbf', gamma=svm.best_estimator_.gamma).f
```

```
In [135]: n = 50
          x = np.linspace(-1, 2, n)
          X, Y = np.meshgrid(x, x)
          points = np.vstack([X.flatten(), Y.flatten()]).T



          Z = np.zeros(shape=(n,n)).flatten()
          Z = new1.predict(points)
          Z = Z.reshape((n,n))


          plt.contour(X, Y, Z)

          class_1 = training[training[:,2]==1, :]
          class_2 = training[training[:,2]==-1, :]

          plt.scatter(class_1[:,0], class_1[:,1], s=60, color = 'lightgreen', label = "class 1")
          plt.scatter(class_2[:,0], class_2[:,1], s=60, color = 'lightblue', label = "class -1")
          plt.scatter(new1.support_vectors_[:,0], new1.support_vectors_[:,1], s=60, color = 'ora
          plt.legend()
```

```
Out[135]: <matplotlib.legend.Legend at 0x11923c88>
```




```
In [136]: print("Number of support vectors = %i" %new1.support_vectors_.shape[0])
```

```
Number of support vectors = 29
```

```
In [137]: y_predicted = new1.predict(Xtest)
          classification_error = (1 - float(np.sum(np.equal(y_predicted, ytest)))/ytest.shape[0]
          print("Percentage of wrong predictions: %0.2f" %classification_error)
```

```
Percentage of wrong predictions: 15.00
```

```
In [138]: print("Accuracy= %0.3f" %new1.score(Xtest, ytest))
```

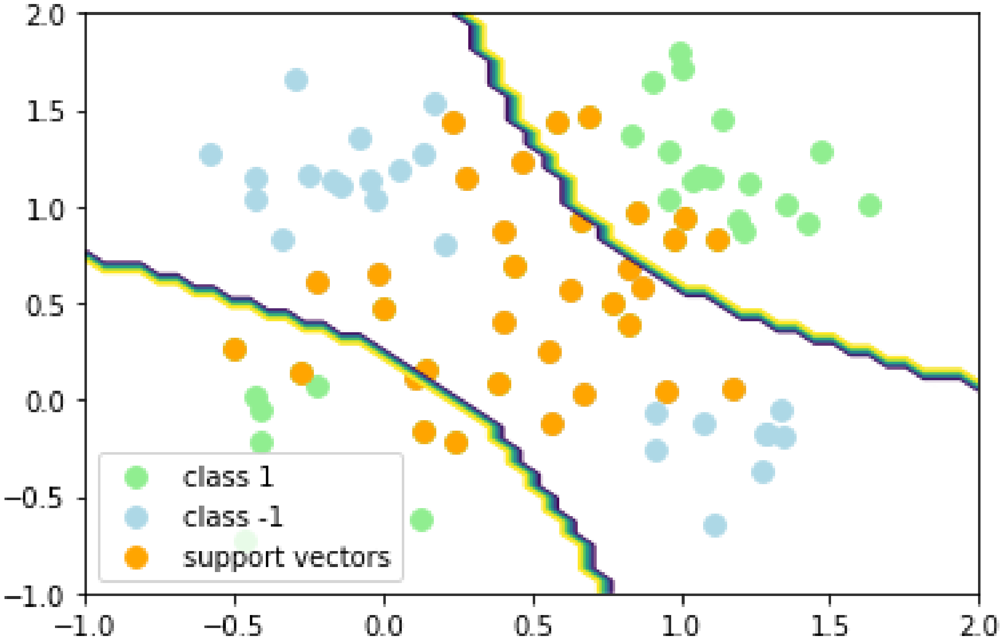
```
Accuracy= 0.850
```

```
In [118]: new2 = SVC(C=svm.best_estimator_.C, kernel='rbf', gamma=svm.best_estimator_.gamma/4).f
```

```
In [119]: n = 50
          x = np.linspace(-1, 2, n)
          X, Y = np.meshgrid(x, x)
          points = np.vstack([X.flatten(), Y.flatten()]).T
```

```
Z = np.zeros(shape=(n,n)).flatten()
Z = new2.predict(points)
Z = Z.reshape((n,n))


plt.contour(X, Y, Z)

class_1 = training[training[:,2]==1, :]
class_2 = training[training[:,2]==-1, :]

plt.scatter(class_1[:,0], class_1[:,1], s=60, color = 'lightgreen', label = "class 1")
plt.scatter(class_2[:,0], class_2[:,1], s=60, color = 'lightblue', label = "class -1")
plt.scatter(new2.support_vectors_[:,0], new2.support_vectors_[:,1], s=60, color = 'ora
plt.legend()
```

Out[119]: <matplotlib.legend.Legend at 0x114eac50>



In [120]:
```
print("Percentage of wrong predictions: %0.2f" %((1-new2.score(Xtest, ytest))*100))
```

```
Percentage of wrong predictions: 22.50
```

In [121]:
```
print("Accuracy= %0.3f" %new2.score(Xtest, ytest))
```

```
Accuracy= 0.775
```

```
In [124]: print("Number of support vectors = %i" %new2.support_vectors_.shape[0])
```

```
Number of support vectors = 33
```