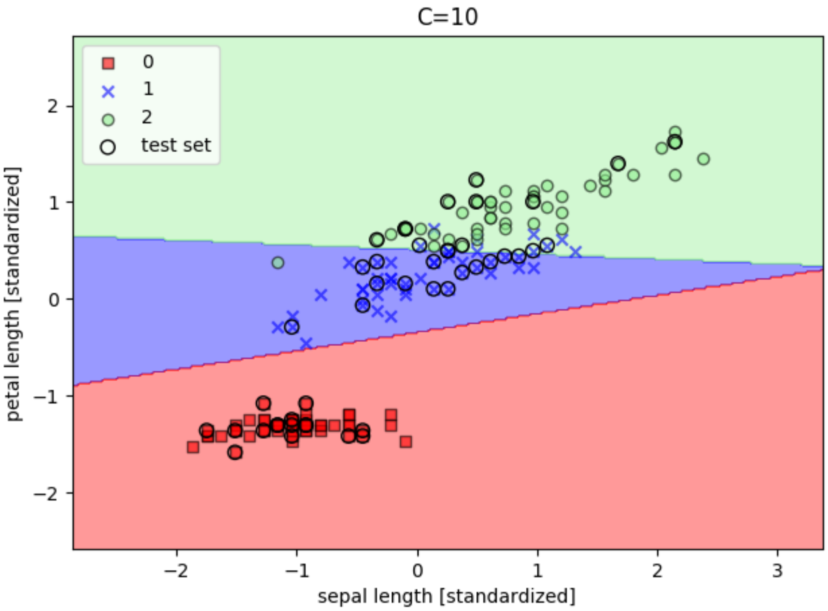
! This class has been made inactive. No posts will be allowed until an instructor reactivates the class.

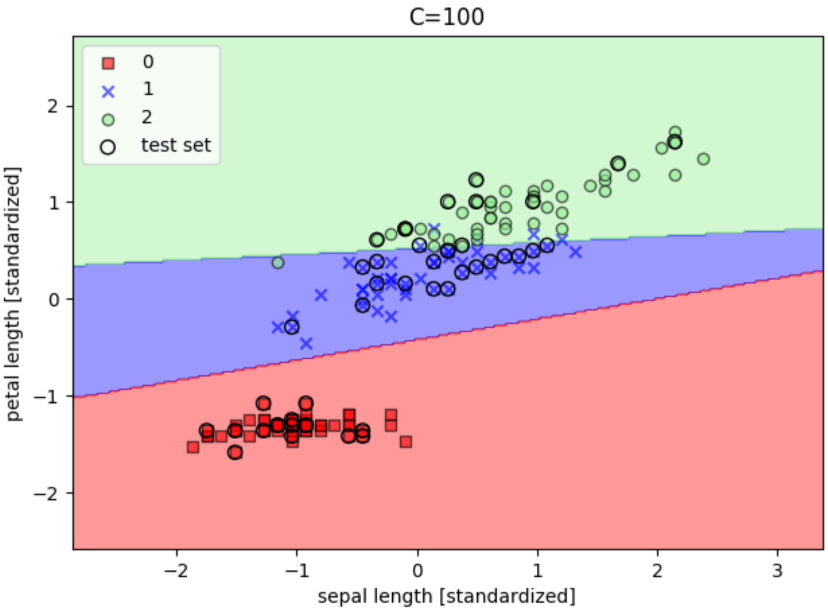
private note @15 3 views

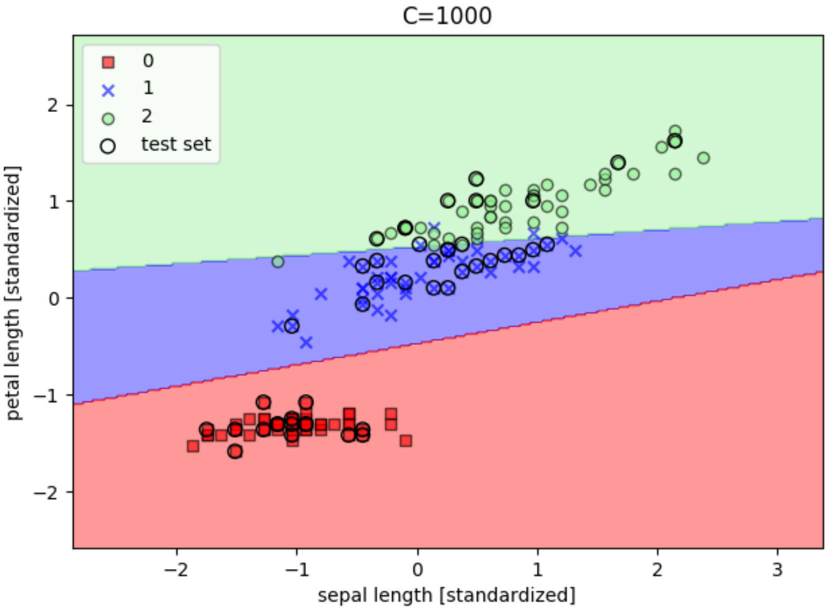
Homework_1_2017310936_Md_Shirajum_Munir

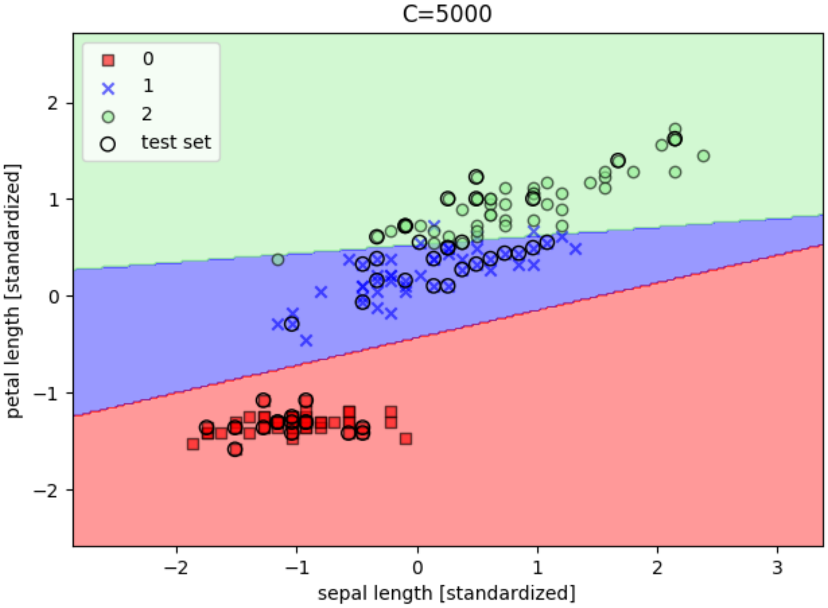
```
import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt
import warnings
from sklearn.linear_model import LogisticRegression

iris = datasets.load_iris()
X = iris.data[:, [0, 2]]
y = iris.target
print('Class labels:', np.unique(y))

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)


sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)

def versiontuple(v):
    return tuple(map(int, (v.split("."))))


def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):

    # setup marker generator and color map
    markers = ('s', 'x', 'o', '^', 'v')
    colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0],
                    y=X[y == cl, 1],
                    alpha=0.6,
                    c=cmap(idx),
                    edgecolor='black',
                    marker=markers[idx],
                    label=cl)

    # highlight test samples
    if test_idx:
        # plot all samples
        if not versiontuple(np.__version__) >= versiontuple('1.9.0'):
            X_test, y_test = X[list(test_idx), :], y[list(test_idx)]
            warnings.warn('Please update to NumPy 1.9.0 or newer')
        else:
            X_test, y_test = X[test_idx, :], y[test_idx]

        plt.scatter(X_test[:, 0],
                    X_test[:, 1],
                    c='',
                    alpha=1.0,
                    edgecolor='black',
                    linewidths=1,
                    marker='o',
                    s=55, label='test set')


C1=[10.0, 100.0,1000.0,5000.0]

for cc in C1:
    lr1 = LogisticRegression(C=cc, random_state=0)
    lr1.fit(X_train_std, y_train)

    X_combined_std = np.vstack((X_train_std, X_test_std))
```

```
y_combined = np.hstack((y_train, y_test))

plot_decision_regions(X_combined_std, y_combined, classifier=lr1, test_idx=range(105, 150))
# plt.xlabel('petal length [standardized]')
# plt.ylabel('petal width [standardized]')

plt.xlabel('sepal length [standardized]')
plt.ylabel('petal length [standardized]')
plt.title(cc)
plt.legend(loc='upper left')
plt.tight_layout()
# plt.savefig('./figures/logistic_regression.png', dpi=300)
plt.show()
```

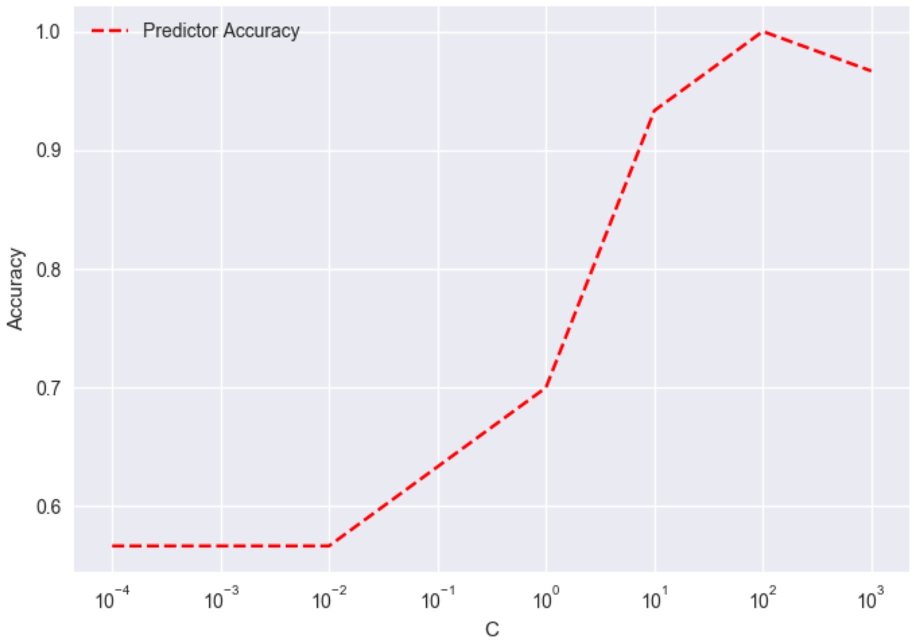




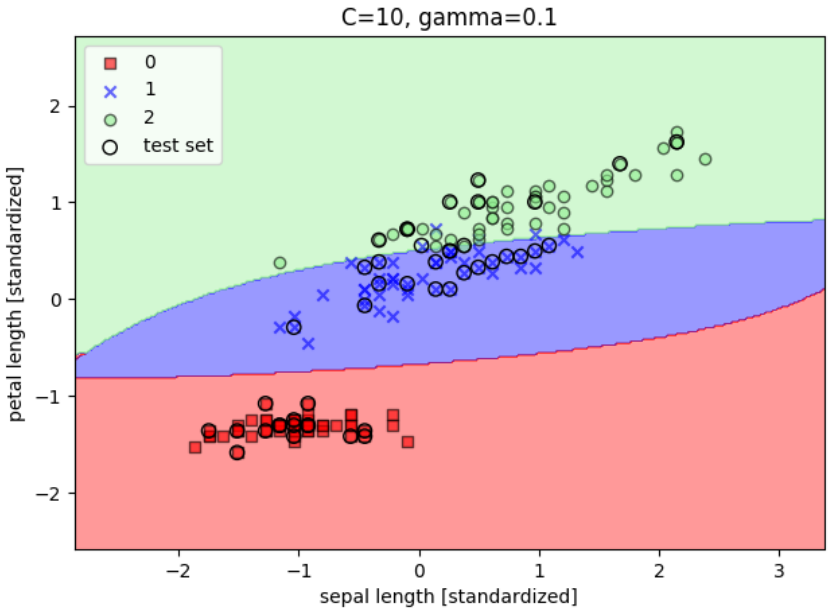


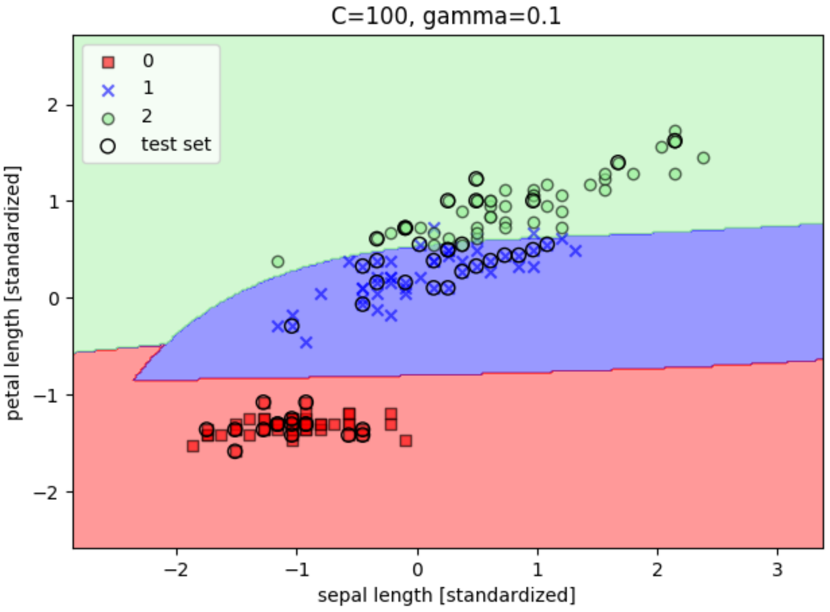


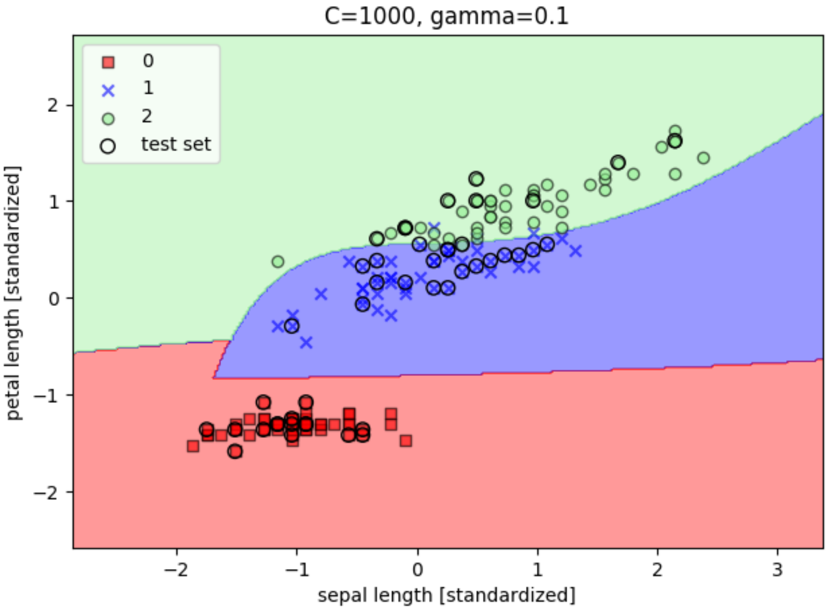
```
from sklearn.metrics import accuracy_score
accuracy3 = []
weights, params = [], []
for c in np.arange(-4., 4.):
    lr4 = LogisticRegression(C=10.**c, random_state=0)
    lr4.fit(X_train_std, y_train)
    weights.append(lr4.coef_[1])
    params.append(10**c)
    y_pred = lr4.predict(X_test_std)
    print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
    accuracy3.append(accuracy_score(y_test, y_pred))

plt.plot(params, accuracy3,color='r', linestyle='--',label='Predictor Accuracy')


plt.ylabel('Accuracy')
plt.xlabel('C')
plt.legend(loc='bottom left')
plt.xscale('log')
# plt.savefig('./figures/regression_path.png', dpi=300)
plt.show()
```

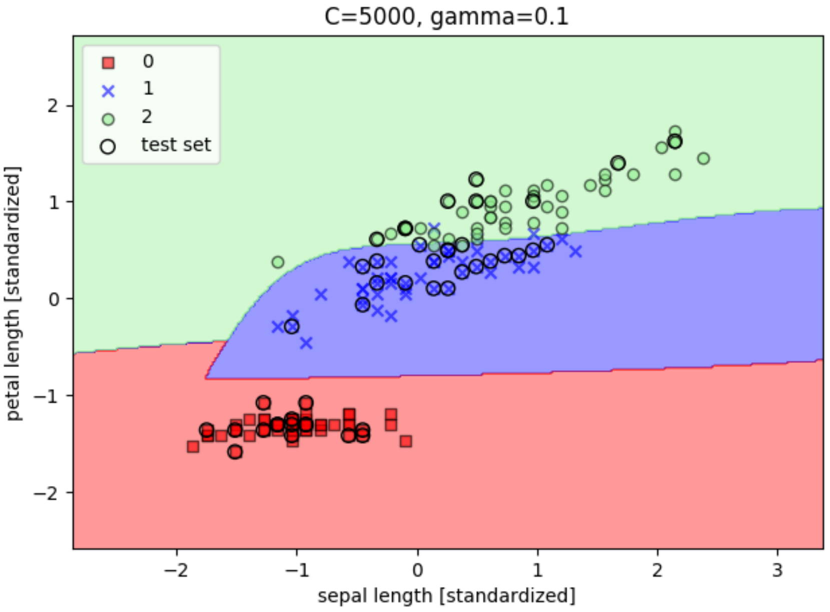


```
for cc in C1:
    svm1 = SVC(kernel='rbf', random_state=0, gamma=0.1, C=cc)
    svm1.fit(X_train_std, y_train)

    X_combined_std = np.vstack((X_train_std, X_test_std))
    y_combined = np.hstack((y_train, y_test))

    plot_decision_regions(X_combined_std, y_combined, classifier=svm1, test_idx=range(105, 150))
    plt.xlabel('sepal length [standardized]')
    plt.ylabel('petal length [standardized]')
    plt.title(cc)
    plt.legend(loc='upper left')
    plt.tight_layout()
    # plt.savefig('./figures/support_vector_machine_rbf_iris_1.png', dpi=300)
    plt.show()
```

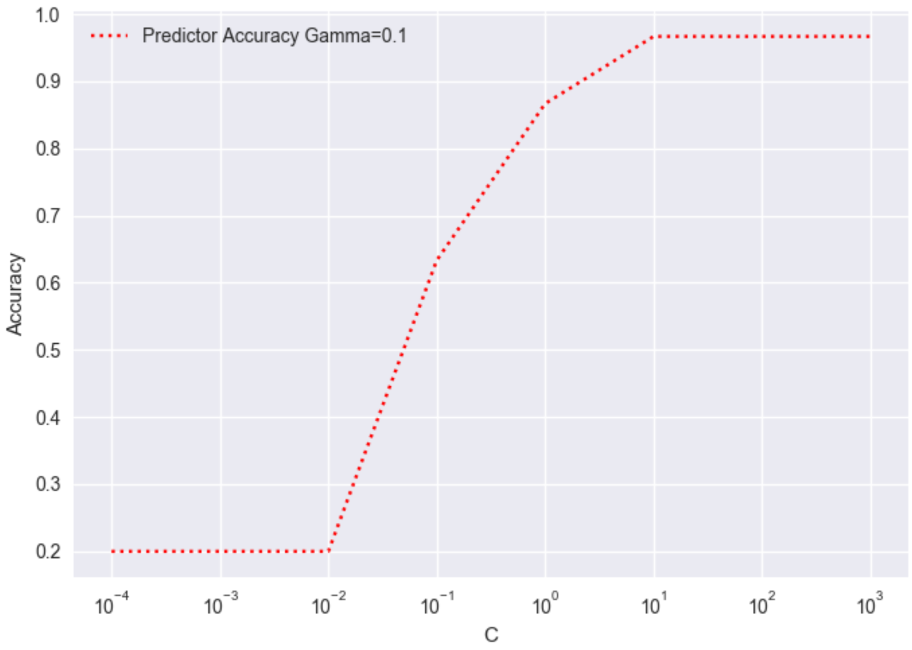



C=100, gamma=0.1
0
1
2
test set
petal length [standardized]
2
1
0
−1
−2
−2
−1
0
1
2
3
sepal length [standardized]

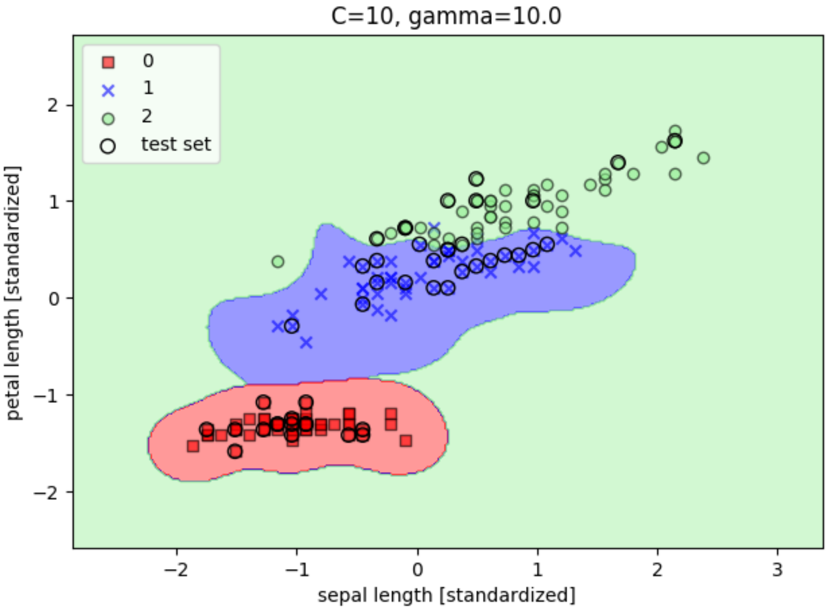

C=1000, gamma=0.1
0
1
2
test set
petal length [standardized]
2
1
0
−1
−2
−2
−1
0
1
2
3
sepal length [standardized]

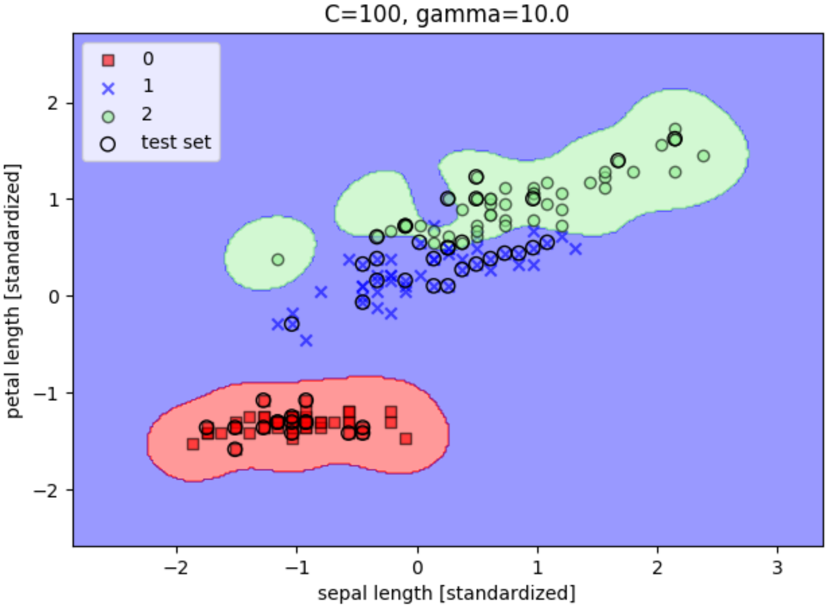

C=5000, gamma=0.1
0
1
2
test set
petal length [standardized]
2
1
0
−1
−2
−2
−1
0
1
2
3
sepal length [standardized]

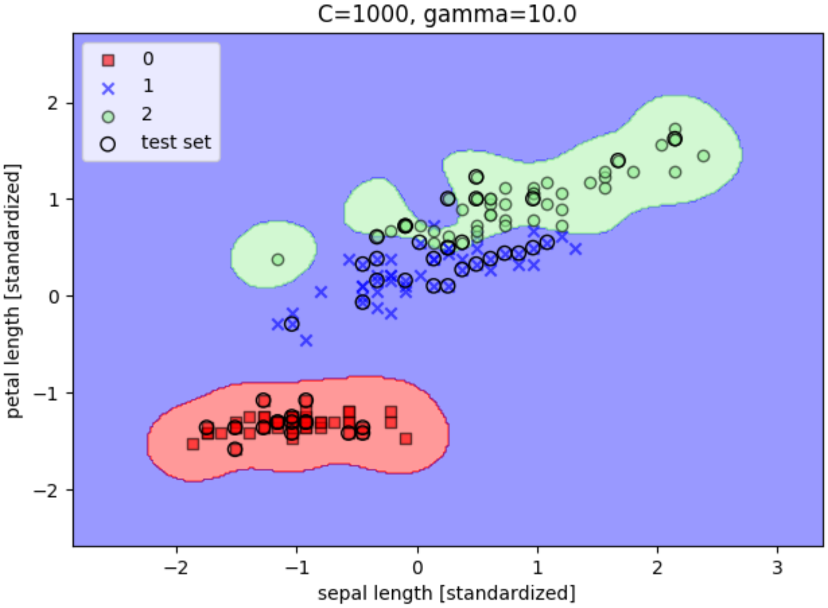
```
accuracy1 = []
weights, params = [], []
for c in np.arange(-4., 4.):
    svm = SVC(kernel='rbf', random_state=0, gamma=0.1, C=10.**c)
    svm.fit(X_train_std, y_train)
    params.append(10**c)
    y_pred = svm.predict(X_test_std)
    print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
    accuracy1.append(accuracy_score(y_test, y_pred))

plt.plot(params, accuracy1,color='r', linestyle=':',label='Predictor Accuracy Gamma=0.1')
plt.ylabel('Accuracy')
plt.xlabel('C')
plt.legend(loc='bottom left')
plt.xscale('log')
# plt.savefig('./figures/regression_path.png', dpi=300)
plt.show()
```

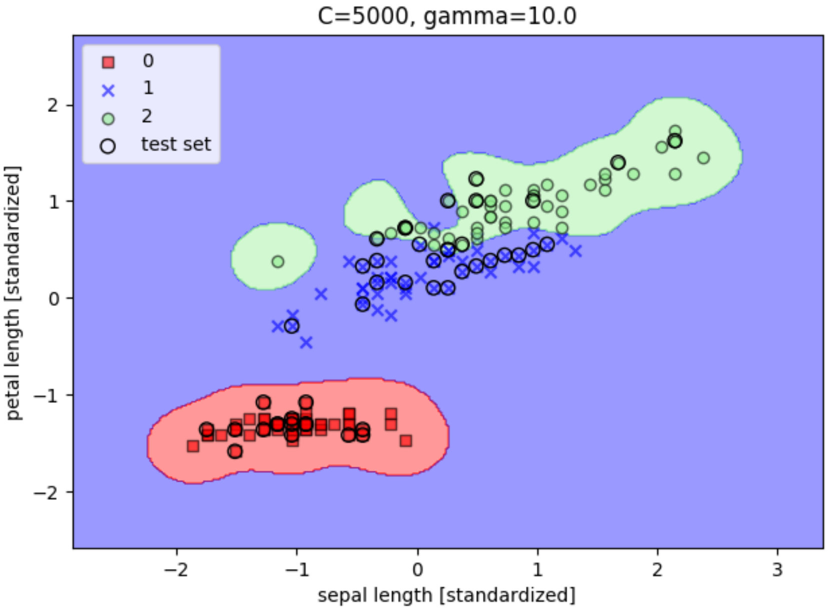


```
for cc in C1:
    svm5 = SVC(kernel='rbf', random_state=0, gamma=10.0, C=cc)
    svm5.fit(X_train_std, y_train)

    X_combined_std = np.vstack((X_train_std, X_test_std))
    y_combined = np.hstack((y_train, y_test))

    plot_decision_regions(X_combined_std, y_combined, classifier=svm5, test_idx=range(105, 150))
    plt.xlabel('sepal length [standardized]')
    plt.ylabel('petal length [standardized]')
    plt.title(cc)
    plt.legend(loc='upper left')
    plt.tight_layout()
    # plt.savefig('./figures/support_vector_machine_rbf_iris_1.png', dpi=300)
    plt.show()
```

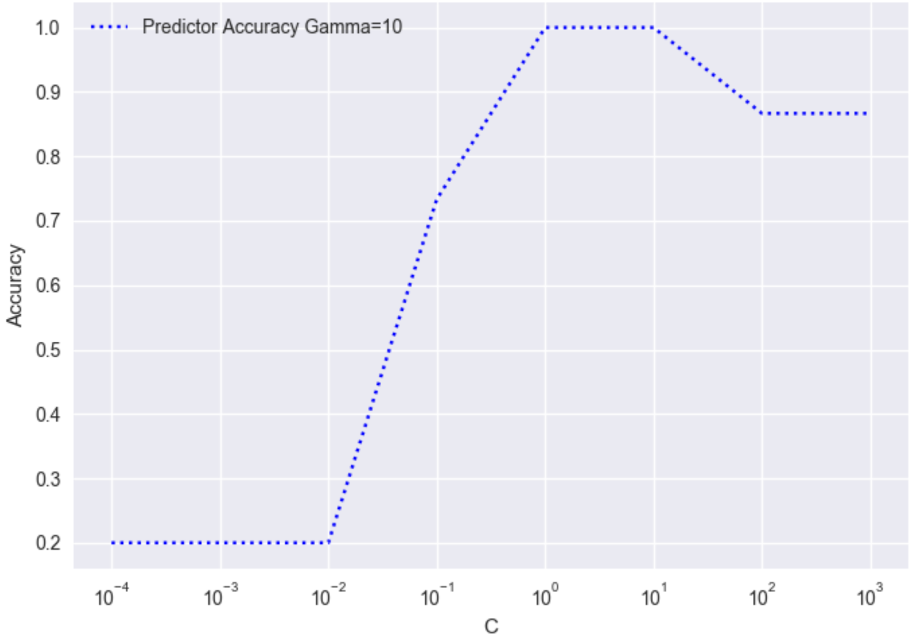

C=10, gamma=10.0
0
1
2
test set
petal length [standardized]
sepal length [standardized]
2
1
0
−1
−2
−2
−1
0
1
2
3


C=100, gamma=10.0
0
1
2
test set
petal length [standardized]
sepal length [standardized]
2
1
0
−1
−2
−2
−1
0
1
2
3


C=1000, gamma=10.0
0
1
2
test set
petal length [standardized]
sepal length [standardized]
2
1
0
−1
−2
−2
−1
0
1
2
3

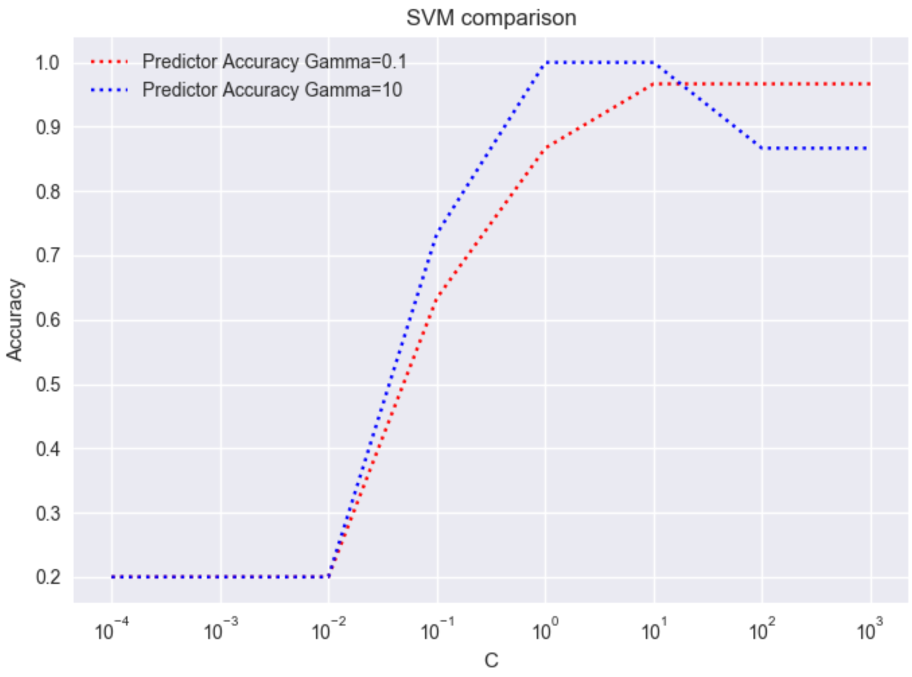


```
accuracy4 = []
weights, params = [], []
for c in np.arange(-4., 4.):
    svm = SVC(kernel='rbf', random_state=0, gamma=10, C=10.**c)
    svm.fit(X_train_std, y_train)
    params.append(10**c)
    y_pred = svm.predict(X_test_std)
    print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
    accuracy4.append(accuracy_score(y_test, y_pred))

plt.plot(params, accuracy4,color='b', linestyle=':',label='Predictor Accuracy Gamma=10')

plt.ylabel('Accuracy')
plt.xlabel('C')
plt.legend(loc='bottom left')
plt.xscale('log')
# plt.savefig('./figures/regression_path.png', dpi=300)
plt.show()
```



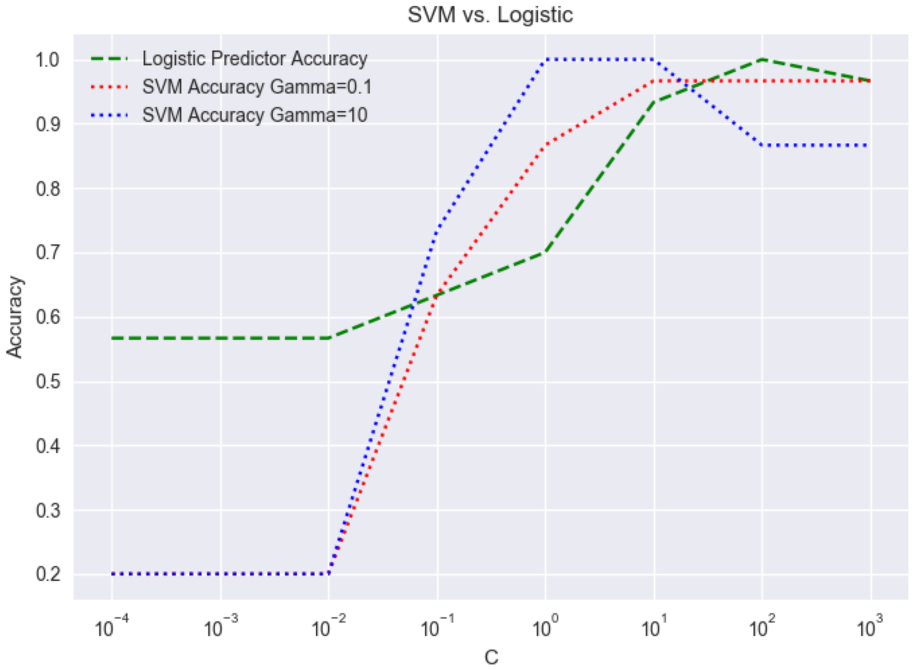
```
import seaborn as sns
sns.set()

accuracy1 = []
weights, params = [], []
```

```
for c in np.arange(-4., 4.):
    svm = SVC(kernel='rbf', random_state=0, gamma=0.1, C=10.**c)
    svm.fit(X_train_std, y_train)
    params.append(10**c)
    y_pred = svm.predict(X_test_std)
    print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
    accuracy1.append(accuracy_score(y_test, y_pred))

plt.plot(params, accuracy1,color='r', linestyle=':',label='Predictor Accuracy Gamma=0.1')

accuracy4 = []
weights, params = [], []
for c in np.arange(-4., 4.):
    svm = SVC(kernel='rbf', random_state=0, gamma=10, C=10.**c)
    svm.fit(X_train_std, y_train)
    params.append(10**c)
    y_pred = svm.predict(X_test_std)
    print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
    accuracy4.append(accuracy_score(y_test, y_pred))

plt.plot(params, accuracy4,color='b', linestyle=':',label='Predictor Accuracy Gamma=10')


plt.ylabel('Accuracy')
plt.xlabel('C')
plt.legend(loc='bottom left')
plt.title("SVM comparison")
plt.xscale('log')
plt.show()
```

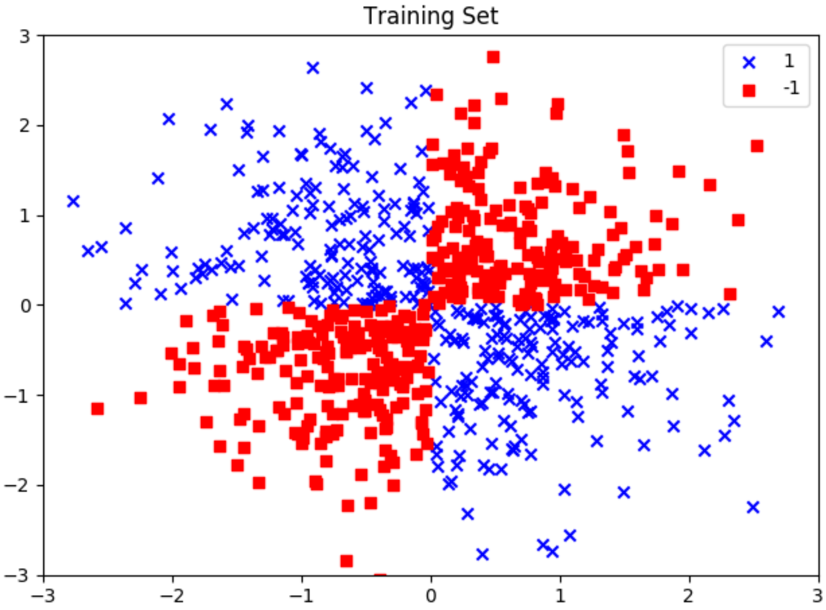


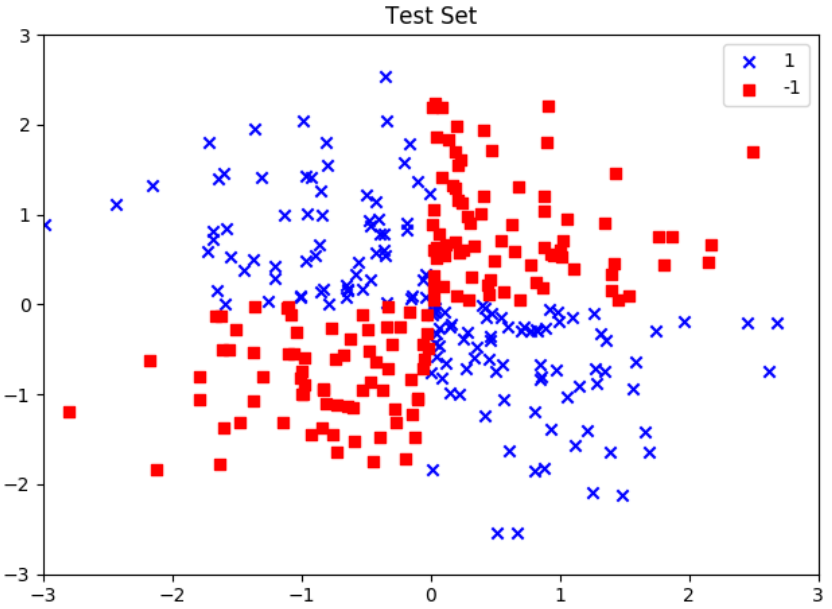
```
accuracy3 = []
weights, params = [], []
for c in np.arange(-4., 4.):
    lr4 = LogisticRegression(C=10.**c, random_state=0)
    lr4.fit(X_train_std, y_train)
    weights.append(lr4.coef_[1])
    params.append(10**c)
    y_pred = lr4.predict(X_test_std)
    print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
    accuracy3.append(accuracy_score(y_test, y_pred))

plt.plot(params, accuracy3,color='g', linestyle='--',label='Logistic Predictor Accuracy')

accuracy1 = []
weights, params = [], []
for c in np.arange(-4., 4.):
    svm = SVC(kernel='rbf', random_state=0, gamma=0.1, C=10.**c)
    svm.fit(X_train_std, y_train)
    params.append(10**c)
    y_pred = svm.predict(X_test_std)
    print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
    accuracy1.append(accuracy_score(y_test, y_pred))

plt.plot(params, accuracy1,color='r', linestyle=':',label='SVM Accuracy Gamma=0.1')
```

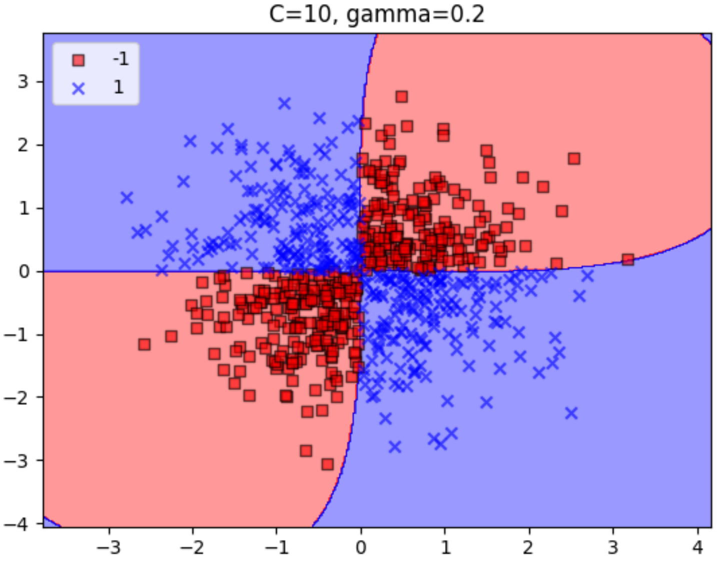
```
accuracy4 = []
weights, params = [], []
for c in np.arange(-4., 4.):
    svm = SVC(kernel='rbf', random_state=0, gamma=10, C=10.**c)
    svm.fit(X_train_std, y_train)
    params.append(10**c)
    y_pred = svm.predict(X_test_std)
    print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
    accuracy4.append(accuracy_score(y_test, y_pred))

plt.plot(params, accuracy4,color='b', linestyle=':',label='SVM Accuracy Gamma=10')


plt.ylabel('Accuracy')
plt.xlabel('C')
plt.legend(loc='bottom left')
plt.title("SVM vs. Logistic")
plt.xscale('log')
plt.show()
```

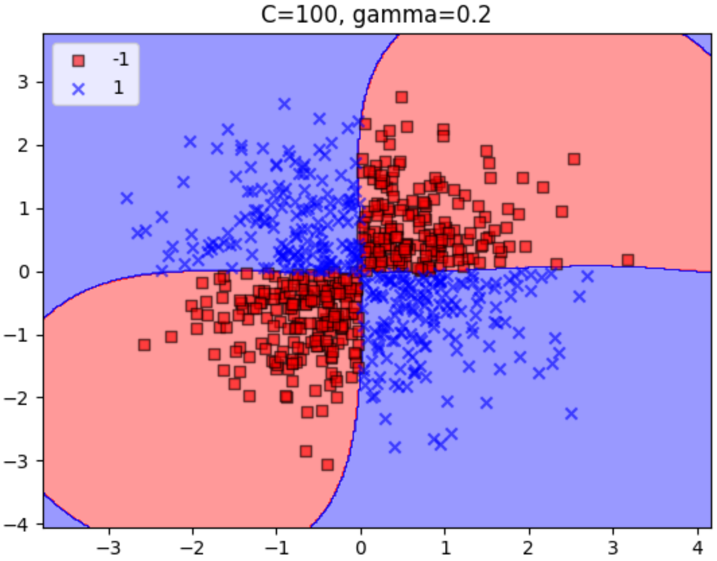


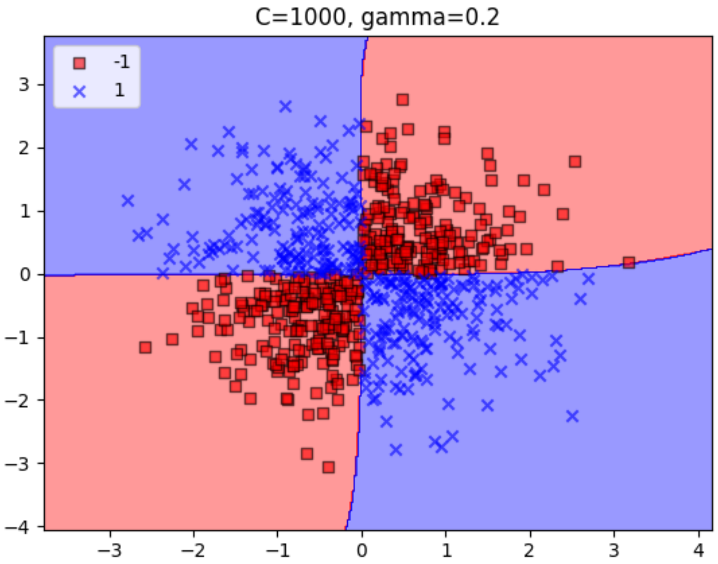
```
np.random.seed(0)
X_xor = np.random.randn(700, 2)
X_test = np.random.randn(300, 2)

y_xor = np.logical_xor(X_xor[:, 0] > 0,
                       X_xor[:, 1] > 0)

y_test = np.logical_xor(X_test[:, 0] > 0,
                        X_test[:, 1] > 0)


y_xor = np.where(y_xor, 1, -1)
y_test = np.where(y_test, 1, -1)


plt.scatter(X_xor[y_xor == 1, 0],
            X_xor[y_xor == 1, 1],
            c='b', marker='x',
            label='1')
plt.scatter(X_xor[y_xor == -1, 0],
            X_xor[y_xor == -1, 1],
            c='r',
            marker='s',
            label='-1')

plt.xlim([-3, 3])
plt.ylim([-3, 3])
plt.legend(loc='best')
plt.title("Training Set")
plt.tight_layout()
# plt.savefig('./figures/xor.png', dpi=300)
plt.show()
plt.scatter(X_test[y_test == 1, 0],
            X_test[y_test == 1, 1],
            c='b', marker='x',
            label='1')
plt.scatter(X_test[y_test == -1, 0],
            X_test[y_test == -1, 1],
            c='r',
            marker='s',
            label='-1')

plt.xlim([-3, 3])
plt.ylim([-3, 3])
plt.legend(loc='best')
plt.title("Test Set")
plt.tight_layout()
# plt.savefig('./figures/xor.png', dpi=300)
plt.show()

for cc in C1:

    svm = SVC(kernel='rbf', random_state=0, gamma=0.20, C=cc)
    svm.fit(X_xor, y_xor)

    # draw decision boundary
    plot_decision_regions(X_xor, y_xor, classifier=svm)
    plt.legend(loc='upper left')
    plt.title(cc)
    plt.show()
```

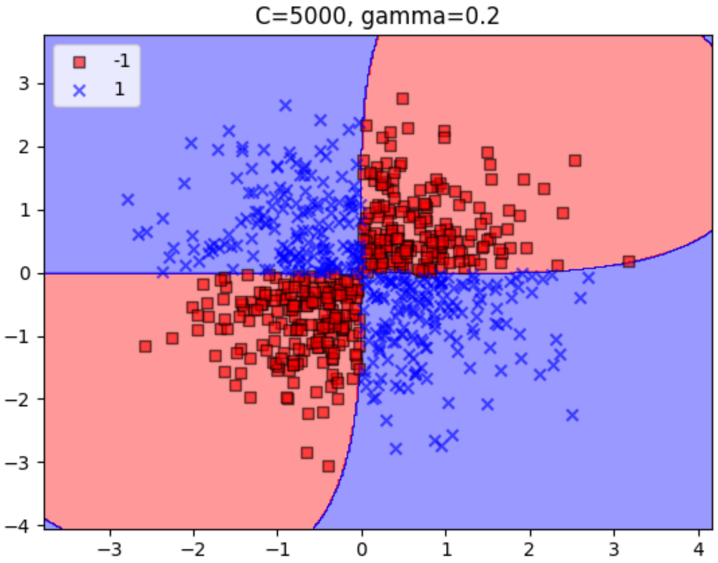



Test Set
1
-1
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

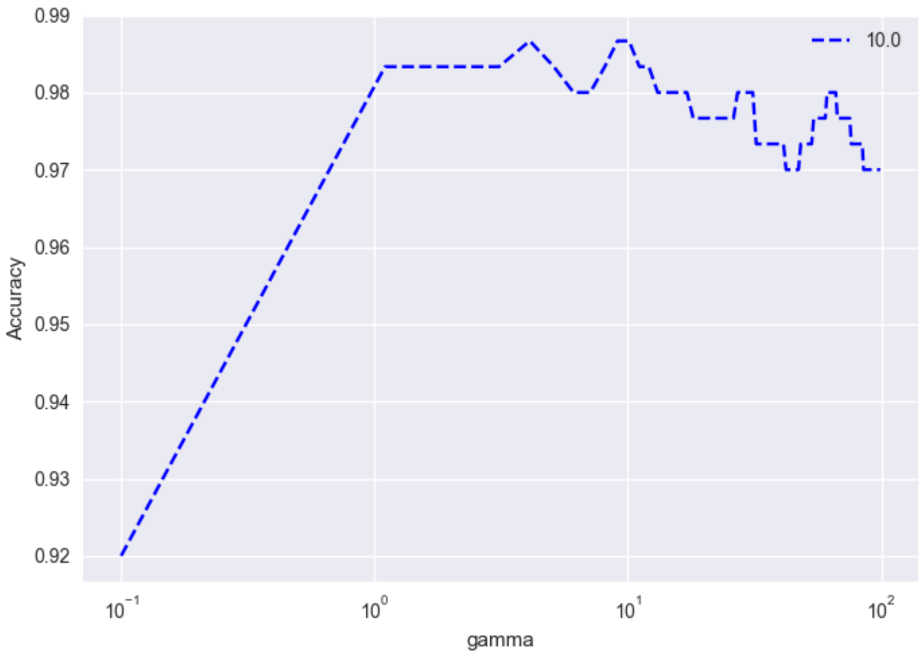

C=10, gamma=0.2
-1
1
3
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3
4

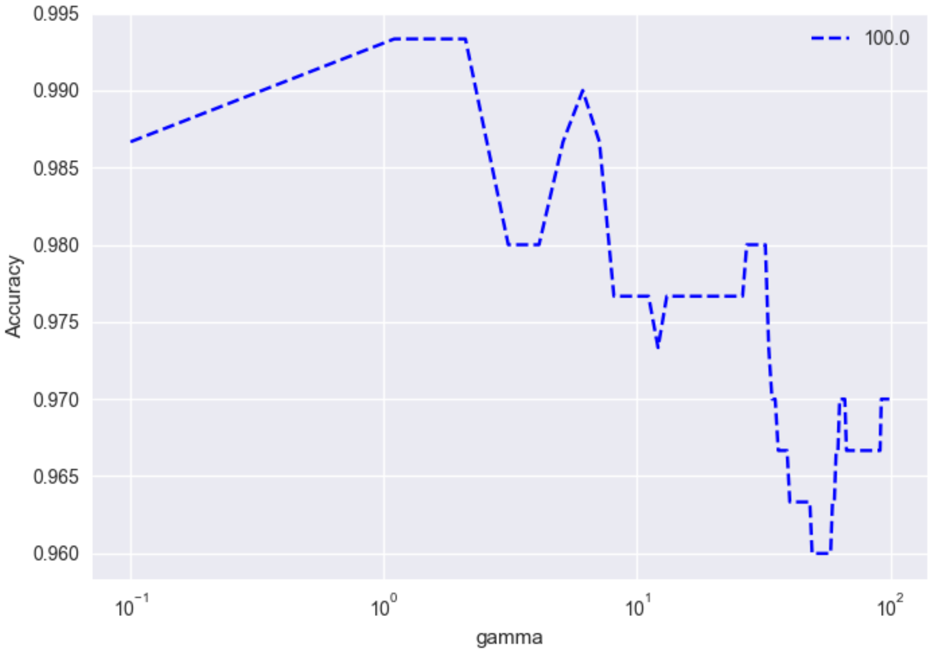

C=100, gamma=0.2
-1
1
3
2
1
0
−1
−2
−3
−4
−3
−2
−1
0
1
2
3
4

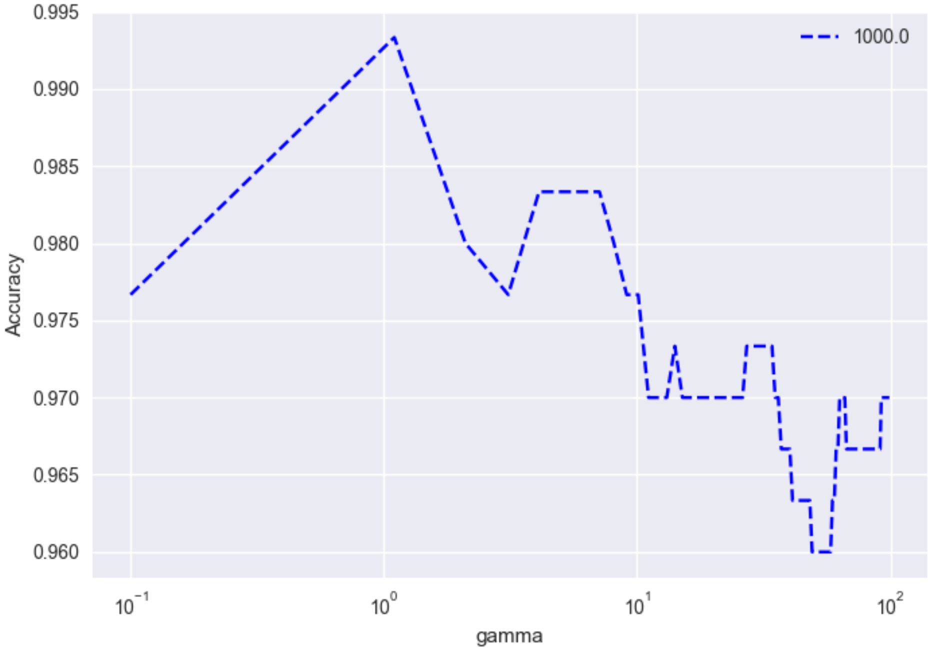


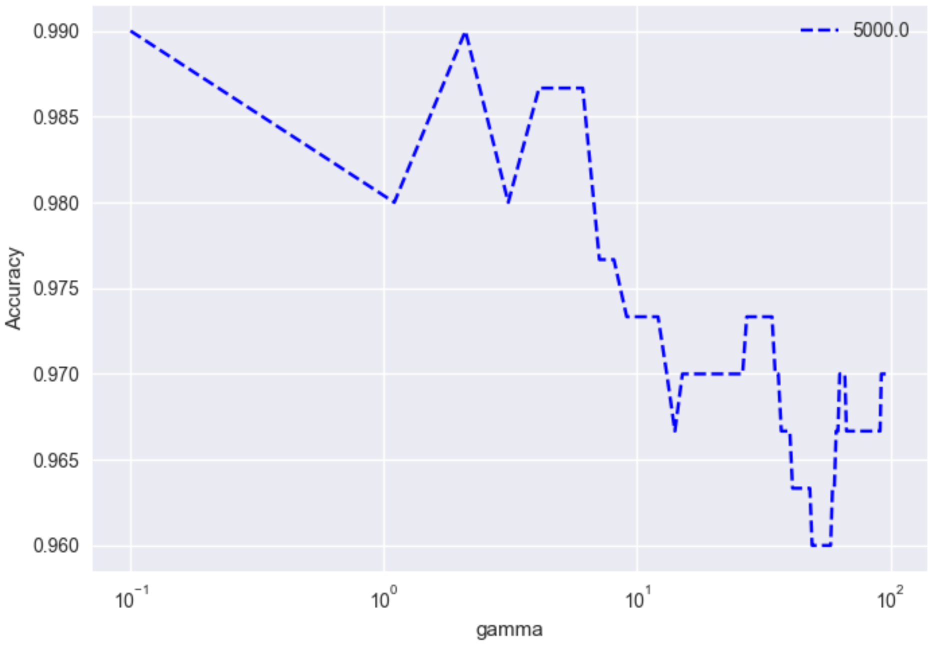


```
from sklearn.metrics import accuracy_score

for cc in C1:
    print(cc)

    accuracy = []
    weights, params = [], []
    for c in np.arange(0.1, 100.):
        svm = SVC(kernel='rbf', random_state=0, gamma=c, C=cc)
        svm.fit(X_xor, y_xor)
        # weights.append(svm.coef_[1])
        params.append(c)
        y_pred = svm.predict(X_test)
        print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
        accuracy.append(accuracy_score(y_test, y_pred))

    weights = np.array(weights)
    # plt.plot(params, weights[:, 0],label='sepal length')
    # plt.plot(params, weights[:, 1], linestyle='--',label='petal length')
    plt.plot(params, accuracy, color='b', linestyle='--',label=cc)
    plt.ylabel('Accuracy')
    plt.xlabel('C')
    plt.legend(loc='bottom left')
    plt.xscale('log')
    # plt.savefig('./figures/regression_path.png', dpi=300)
    plt.show()
```

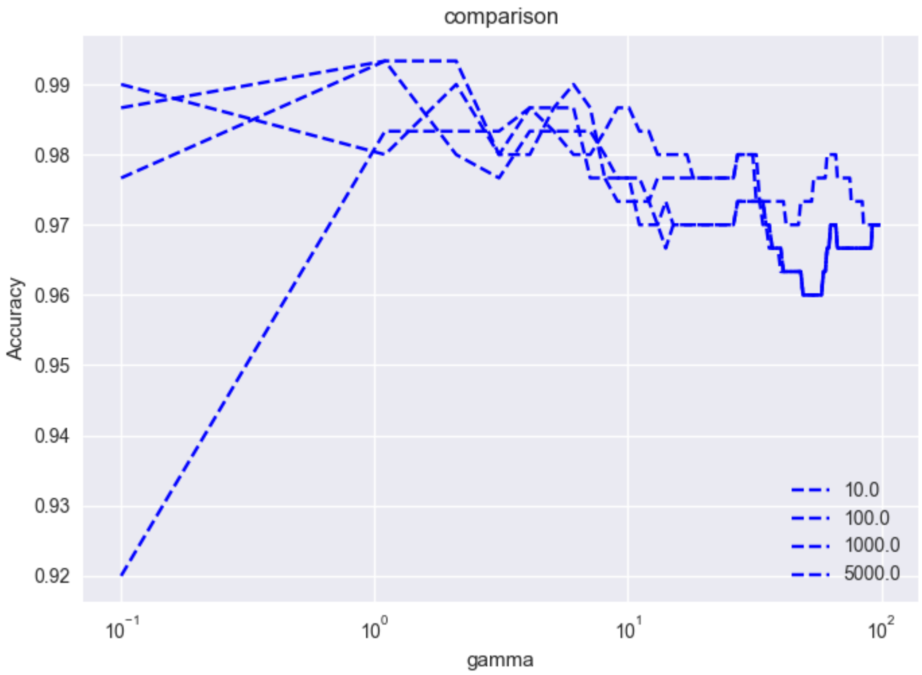

0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
Accuracy
10.0
10^{-1}
10^{0}
10^{1}
10^{2}
gamma


0.995
0.990
0.985
0.980
0.975
0.970
0.965
0.960
Accuracy
100.0
10^{-1}
10^{0}
10^{1}
10^{2}
gamma





```
for cc in C1:
    accuracy = []
    weights, params = [], []
    for c in np.arange(0.1, 100.):
        svm = SVC(kernel='rbf', random_state=0, gamma=c, C=cc)
        svm.fit(X_xor, y_xor)
        # weights.append(svm.coef_[1])
        params.append(c)
        y_pred = svm.predict(X_test)
        print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
        accuracy.append(accuracy_score(y_test, y_pred))

    weights = np.array(weights)
    plt.plot(params, accuracy, color='b', linestyle='--',label=cc)
plt.ylabel('Accuracy')
plt.xlabel('gamma')
plt.legend(loc='bottom left')
plt.title("comparison")
plt.xscale('log')
plt.show()
```



hw1

This private post is only visible to Instructors and MD SHIRAJUM MUNIR

Updated 2 years ago by MD SHIRAJUM MUNIR

followup discussions *for lingering questions and comments*