Initial setup:

- Import what we need from sklearn, numpy, and matplotlib
- Make matplotlib inline since we're in notebook
- Plotting code ripped from http://scikit-learn.org/stable/auto_examples/svm/plot_iris.html (http://scikit-learn.org/stable/auto_examples/svm/plot_iris.html)

In [45]:
```
print(__doc__)

import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.model_selection import GridSearchCV
import pandas as pd
import multiprocessing

%matplotlib inline

def make_meshgrid(x, y, h=.02):
    """Create a mesh of points to plot in

    Parameters
    ----------
    x: data to base x-axis meshgrid on
    y: data to base y-axis meshgrid on
    h: stepsize for meshgrid, optional

    Returns
    -------
    xx, yy : ndarray
    """
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    return xx, yy


def plot_contours(ax, clf, xx, yy, **params):
    """Plot the decision boundaries for a classifier.

    Parameters
    ----------
    ax: matplotlib axes object
    clf: a classifier
    xx: meshgrid ndarray
    yy: meshgrid ndarray
    params: dictionary of params to pass to contourf, optional
    """
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    out = ax.contourf(xx, yy, Z, **params)
    return out
```

```
def splitInputOutput(data):
    x, y = np.array(data[:,0:-1], dtype=float), np.array(data[:,-1],dtype=int)
    return x,y.ravel()
```

```
Setup some functions to return gridsearch for the individual kernels.
Algorithm will call each of these functions and graph SVMs using best_params_ from each using the above plot functions.
```

Helpers

Here we define some methods for configuring and setting up GridSearchCV for linear, polynomial, and gaussian. This makes it easy to find the best hyperparameters for linear by simply calling search_linear().

In [46]:
```
"""
Setup some functions to return gridsearch for the individual kernels.
Algorithm will call each of these functions and graph SVMs using best_params_ from each using the above plot functions.
"""

def search_gaussian(X, y, num_folds):
    searchSpace = {'C': [0.001, 0.01, 0.1, 1, 10], 'gamma' : [0.001, 0.01, 0.1, 1]}
    search = GridSearchCV(svm.SVC(kernel='rbf'), searchSpace, n_jobs=multiprocessing.cpu_count(), cv=num_folds)
    search.fit(X, y)
    return search

def search_linear(X, y, num_folds):
    searchSpace = {'C': [0.001, 0.01, 0.1, 1, 10]}
    search = GridSearchCV(svm.SVC(kernel='linear'), searchSpace, cv=num_folds)
    search.fit(X, y)
    return search

def search_polynomial(X, y, num_folds):
    searchSpace = {'C': [0.001, 0.01, 0.1, 1, 10], 'degree' : [2,3]}
    search = GridSearchCV(svm.SVC(kernel='poly'), searchSpace, n_jobs=multiprocessing.cpu_count(), cv=num_folds)
    search.fit(X, y)
    return search
```

Load trivial sample data

Load a sinusoidal w/ gaussian noise data into X and y.

In [47]:
```
radiansX = np.pi*np.random.random_sample((800,))
cosineX = np.cos(radiansX)+(0.4*np.random.random_sample((800,))-0.2)
classification = np.double(cosineX > np.cos(radiansX))

df = pd.DataFrame({'X': radiansX,'Y':cosineX, 'classification': classification});

X,y = splitInputOutput(df.as_matrix())
print(X.shape)
print(y.shape)

radiansX = np.pi*np.random.random_sample((200,))
cosineX = np.cos(radiansX)+(0.4*np.random.random_sample((200,))-0.2)
classification = np.double(cosineX > np.cos(radiansX))
df = pd.DataFrame({'X': radiansX,'Y':cosineX, 'classification': classification});
tX,ty = splitInputOutput(df.as_matrix())
print(tX.shape)
print(ty.shape)
```

```
(800, 2)
(800,)
(200, 2)
(200,)
```

In [48]:
```
print("Searching linear...")
linear_search = search_linear(X,y,5)
print(linear_search.best_params_)
print (linear_search)
```

```
Searching linear...
{'C': 1}
GridSearchCV(cv=5, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'C': [0.001, 0.01, 0.1, 1, 10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=0)
```

In [49]:
```
print("Searching polynomial...")
polynomial_search = search_polynomial(X,y,5)
print(polynomial_search.best_params_)
```

```
Searching polynomial...
{'C': 10, 'degree': 3}
```

In [50]:
```
print("Searching gaussian...")
gaussian_search = search_gaussian(X,y,5)
print(gaussian_search.best_params_)
```

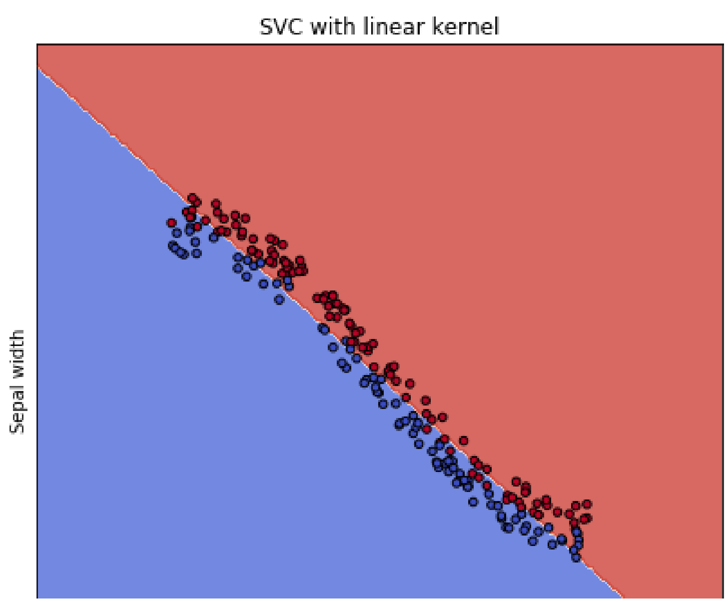
```
Searching gaussian...
{'gamma': 1, 'C': 1}
```

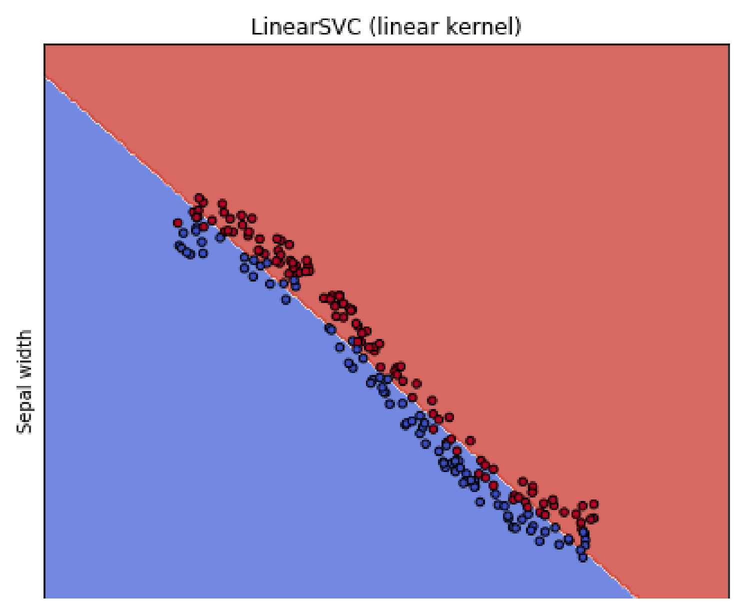
In [51]:
```
models = (svm.SVC(kernel='linear', C=linear_search.best_params_['C']),
          svm.LinearSVC(C=linear_search.best_params_['C']),
          svm.SVC(kernel='rbf', gamma=gaussian_search.best_params_['gamma'], C=gaussian_search.best_params_['C']),
          svm.SVC(kernel='poly', degree=polynomial_search.best_params_['degree'], C=polynomial_search.best_params_['C']))
models = (clf.fit(X, y) for clf in models)
```

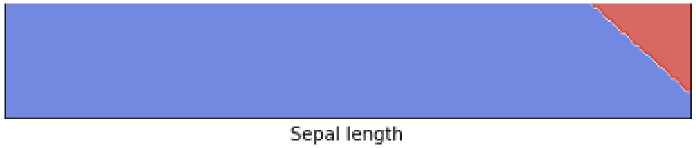
Results

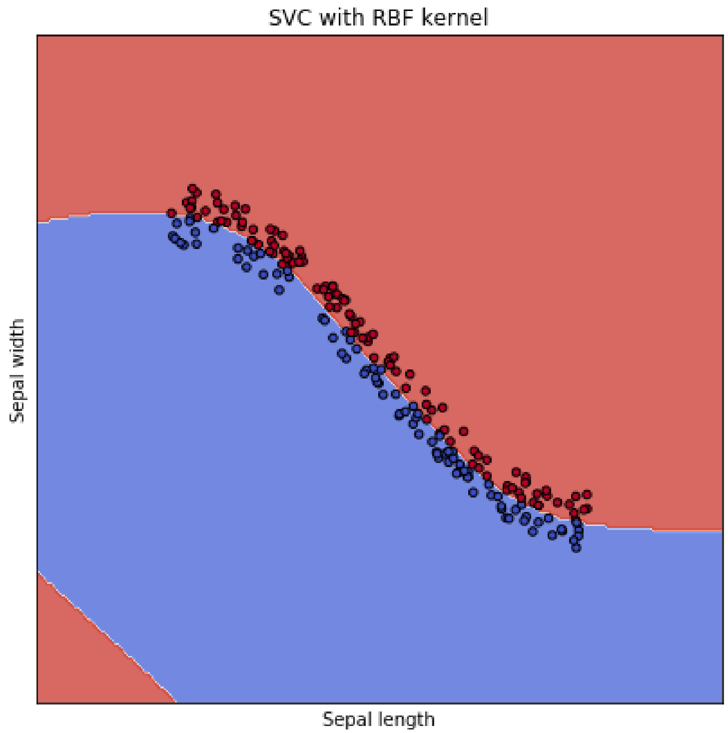
In [52]:

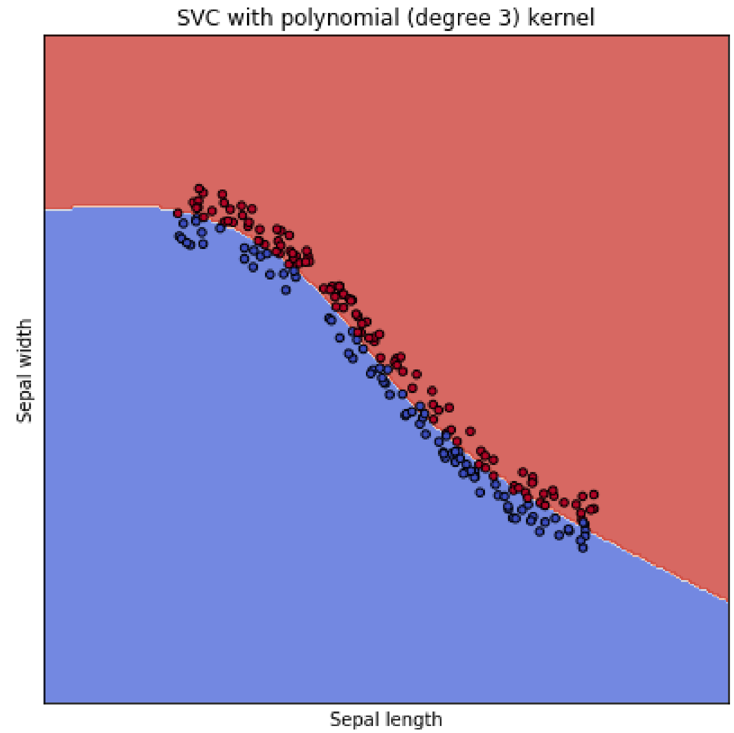
```
# title for the plots
titles = ('SVC with linear kernel',
          'LinearSVC (linear kernel)',
          'SVC with RBF kernel',
          'SVC with polynomial (degree '+str(polynomial_search.best_params_['degree'])+') kernel')

# Set-up 2x2 grid for plotting.
fig, sub = plt.subplots(2, 2,figsize=(15,15))
plt.subplots_adjust(wspace=0.2, hspace=0.2)

X0, X1 = tX[:, 0], tX[:, 1]
xx, yy = make_meshgrid(X0, X1)

for clf, title, ax in zip(models, titles, sub.flatten()):
    plot_contours(ax, clf, xx, yy,
                  cmap=plt.cm.coolwarm, alpha=0.8)
    ax.scatter(X0, X1, c=ty, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xlabel('Sepal length')
    ax.set_ylabel('Sepal width')
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)
plt.show()
plt.close()
```











Load the real data

In [53]:
```
df = pd.read_csv('train.csv',sep=',');

X,y = splitInputOutput(df.as_matrix())
print(X.shape)
print(y.shape)
df = pd.read_csv('test.csv',sep=',');

tX,ty = splitInputOutput(df.as_matrix())
print(tX.shape)
print(ty.shape)
```

```
(480, 2)
(480,)
(122, 2)
(122,)
```

In [54]:
```
print("Searching linear...")
linear_search = search_linear(X,y,5)
print(linear_search.best_params_)
print (linear_search)
```

```
Searching linear...
{'C': 10}
GridSearchCV(cv=5, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape=None, degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'C': [0.001, 0.01, 0.1, 1, 10]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring=None, verbose=0)
```

In [55]:
```
print("Searching polynomial...")
polynomial_search = search_polynomial(X,y,5)
print(polynomial_search.best_params_)
```

```
Searching polynomial...
{'C': 10, 'degree': 2}
```

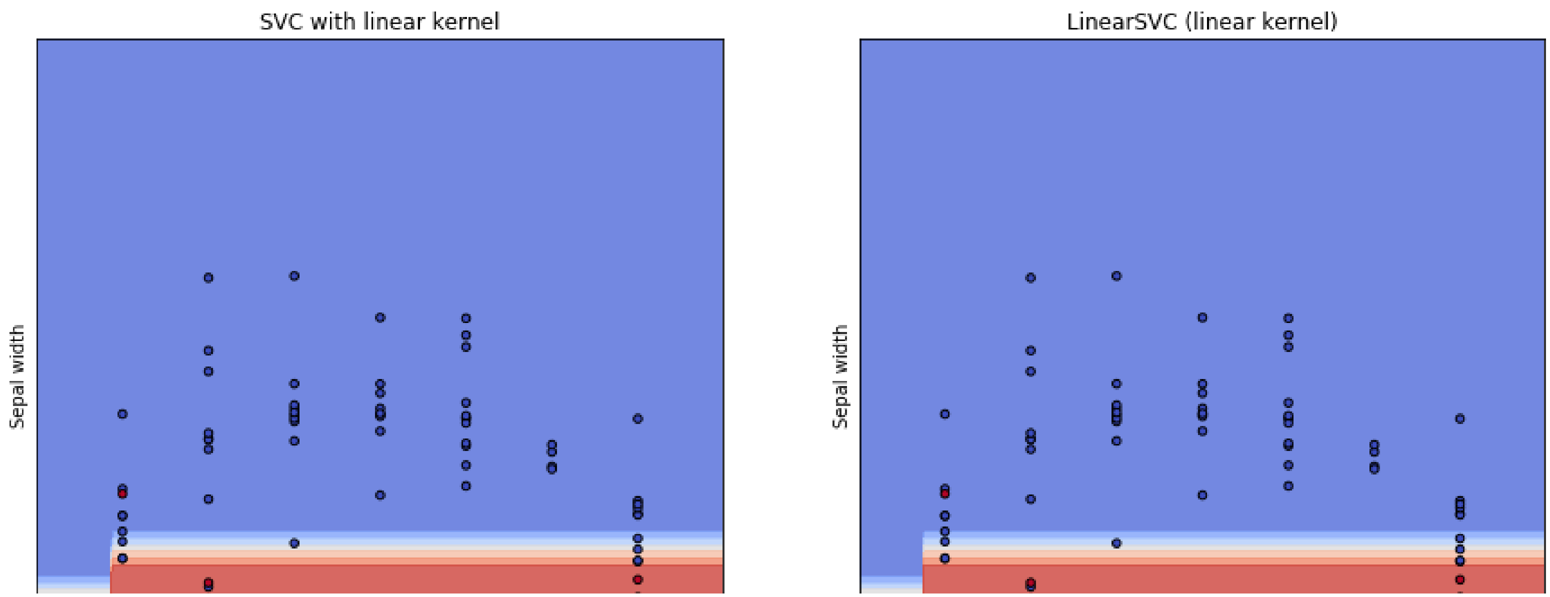
In [56]:
```
print("Searching gaussian...")
gaussian_search = search_gaussian(X,y,5)
print(gaussian_search.best_params_)
```

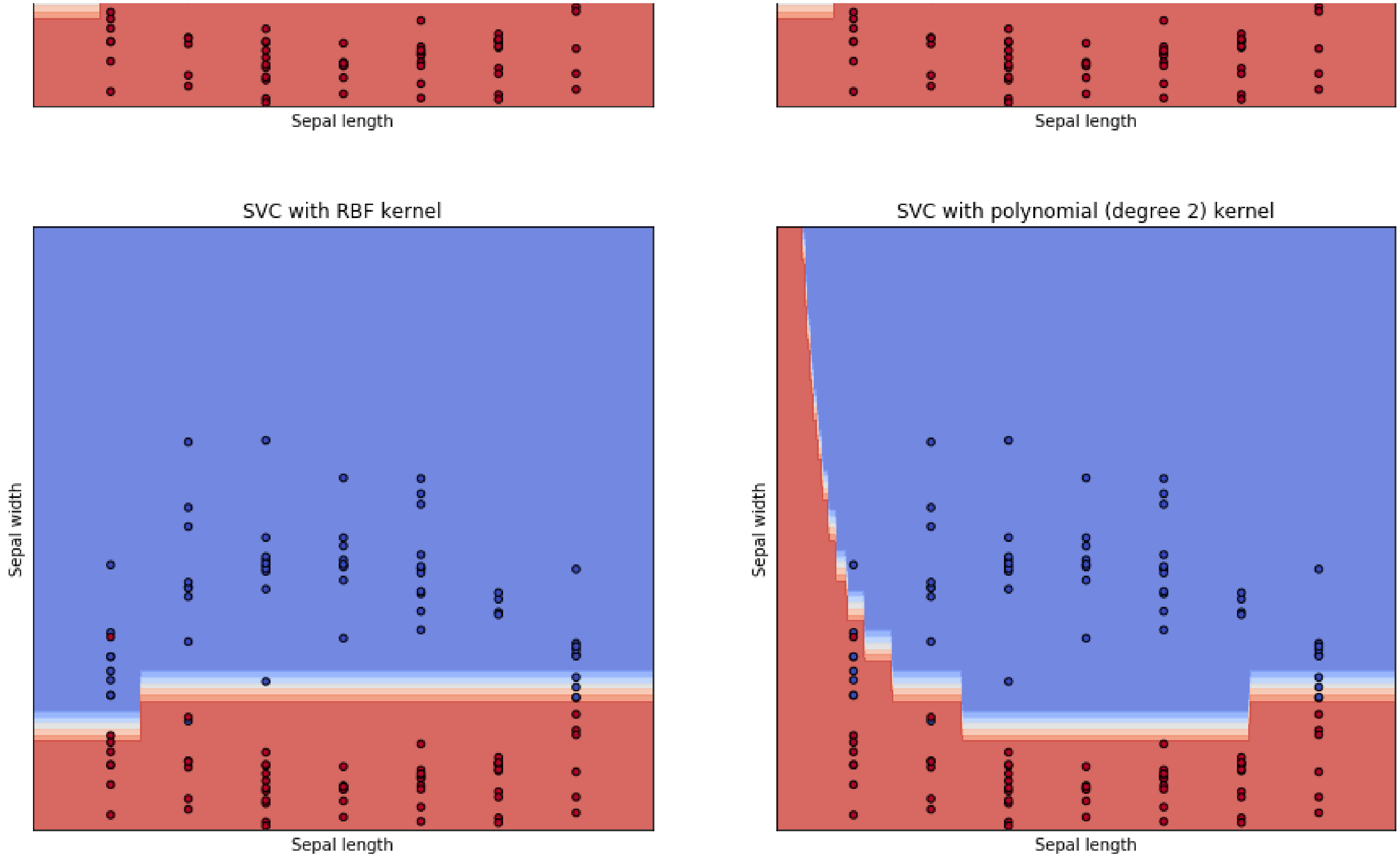
```
Searching gaussian...
{'gamma': 1, 'C': 10}
```

Results

In [57]:
```
models = (svm.SVC(kernel='linear', C=linear_search.best_params_['C']),
          svm.LinearSVC(C=linear_search.best_params_['C']),
          svm.SVC(kernel='rbf', gamma=gaussian_search.best_params_['gamma'], C=gaussian_search.best_params_['C']),
          svm.SVC(kernel='poly', degree=polynomial_search.best_params_['degree'], C=polynomial_search.best_params_['C']))
models = (clf.fit(X, y) for clf in models)
```

In [58]:
```
# title for the plots
titles = ('SVC with linear kernel',
          'LinearSVC (linear kernel)',
          'SVC with RBF kernel',
          'SVC with polynomial (degree '+str(polynomial_search.best_params_['degree'])+') kernel')

# Set-up 2x2 grid for plotting.
fig, sub = plt.subplots(2, 2,figsize=(15,15))
plt.subplots_adjust(wspace=0.2, hspace=0.2)

X0, X1 = tX[:, 0], tX[:, 1]
xx, yy = make_meshgrid(X0, X1)

for clf, title, ax in zip(models, titles, sub.flatten()):
    plot_contours(ax, clf, xx, yy,
                  cmap=plt.cm.coolwarm, alpha=0.8)
    ax.scatter(X0, X1, c=ty, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(0, .3)
    ax.set_xlabel('Sepal length')
    ax.set_ylabel('Sepal width')
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)
plt.show()
plt.close()
```




Sepal length
Sepal length
SVC with RBF kernel
Sepal width
Sepal length
SVC with polynomial (degree 2) kernel
Sepal width
Sepal length