In [1]:

```
from sklearn import svm, datasets
from sklearn.model_selection import train_test_split, cross_validate
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from mlxtend.plotting import plot_decision_regions
```

1. Non-linear separation

As shown below, a radial kernel obviously performs better than a linear kernal on both the training and the test data. The accuracy for the former on the test data is 0.93 and the latter is only 0.65. This is because a linear decision boundary cannot capture the distribution of the points for which the decision boundary should be a circle.

In [2]:

```
X, y = datasets.make_gaussian_quantiles(n_features=2, n_classes=2,
                                        n_samples = 200, random_state=50)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)
print(X_train.shape, X_test.shape)

clf_linear = svm.SVC(kernel = 'linear', gamma = 'auto').fit(X_train, y_train)
clf_radial = svm.SVC(kernel = 'rbf', gamma = 'auto').fit(X_train, y_train)
```

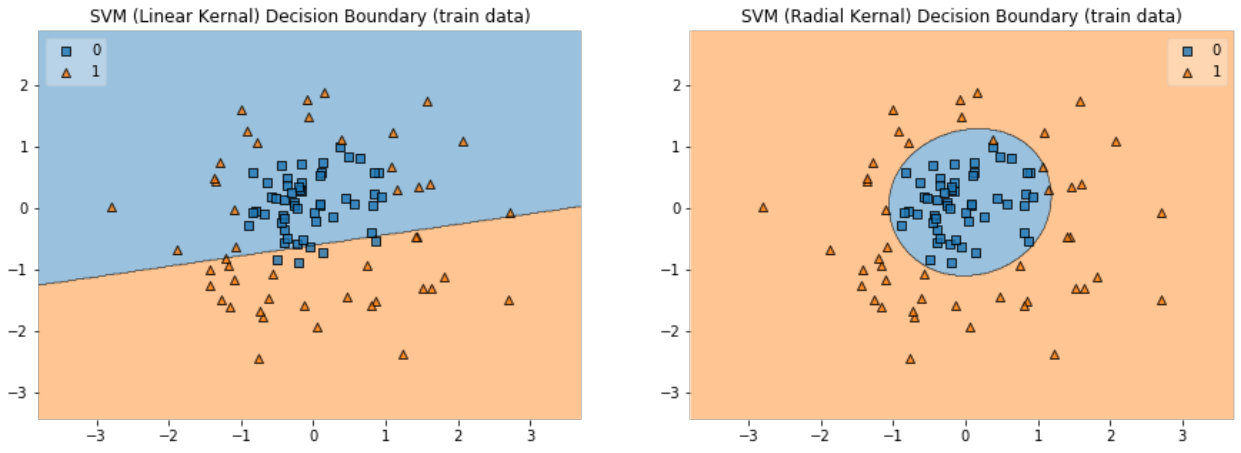
```
(100, 2) (100, 2)
```

In [3]:

```
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

fig = plot_decision_regions(X=X_train, y=y_train, clf=clf_linear,
                            ax=axes[0], legend=2)
axes[0].set_title('SVM (Linear Kernal) Decision Boundary (train data)')
fig = plot_decision_regions(X=X_train, y=y_train, clf=clf_radial,
                            ax=axes[1], legend=1)
axes[1].set_title('SVM (Radial Kernal) Decision Boundary (train data)')
```

Out[3]:

```
Text(0.5, 1.0, 'SVM (Radial Kernal) Decision Boundary (train data)')
```



In [4]:

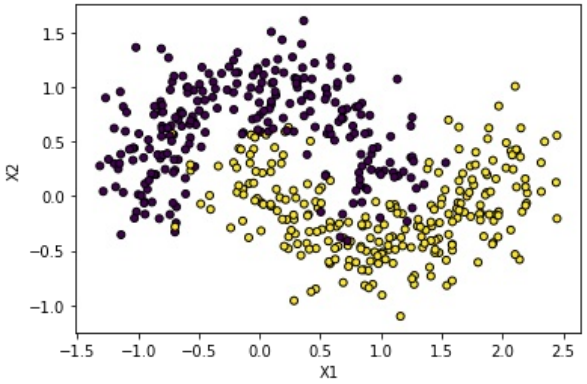
```
pred = clf_linear.predict(X_test)
print('linear kernal accuracy:', accuracy_score(pred, y_test))

pred = clf_radial.predict(X_test)
print('Radial kernal accuracy:', accuracy_score(pred, y_test))
```

```
linear kernal accuracy: 0.65
Radial kernal accuracy: 0.93
```

2. SVM v.s. Logistic Regression

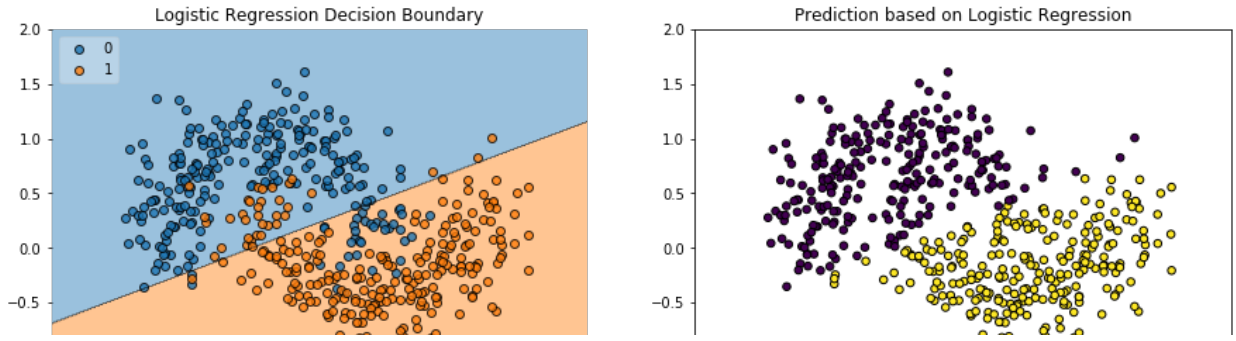
In [39]:

```
X, y = datasets.make_moons(n_samples = 1000, noise = 0.25, random_state = 42)
X, X_test, y, y_test = train_test_split(X, y, test_size=0.5, random_state=42)

plt.scatter(X[:, 0], X[:, 1], marker='o', c=y,
            s=25, edgecolor='k')
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()
```



In [40]:

```
clf_log = LogisticRegression(random_state=0, solver = "lbfgs").fit(X, y)
pred = clf_log.predict(X)

fig, axes = plt.subplots(1, 2, figsize=(15, 5))

fig = plot_decision_regions(X, y, clf = clf_log, ax=axes[0], legend=2,
                            markers = 'o')
axes[0].set_title('Logistic Regression Decision Boundary')
axes[0].set_xlim([-2, 3])
axes[0].set_ylim([-1.5, 2])

fig = plt.scatter(X[:, 0], X[:, 1], marker='o', c=pred, s=30, edgecolor='k')
axes[1].set_title('Prediction based on Logistic Regression')
axes[1].set_xlim([-2, 3])
axes[1].set_ylim([-1.5, 2])

plt.show()
```



In [41]:

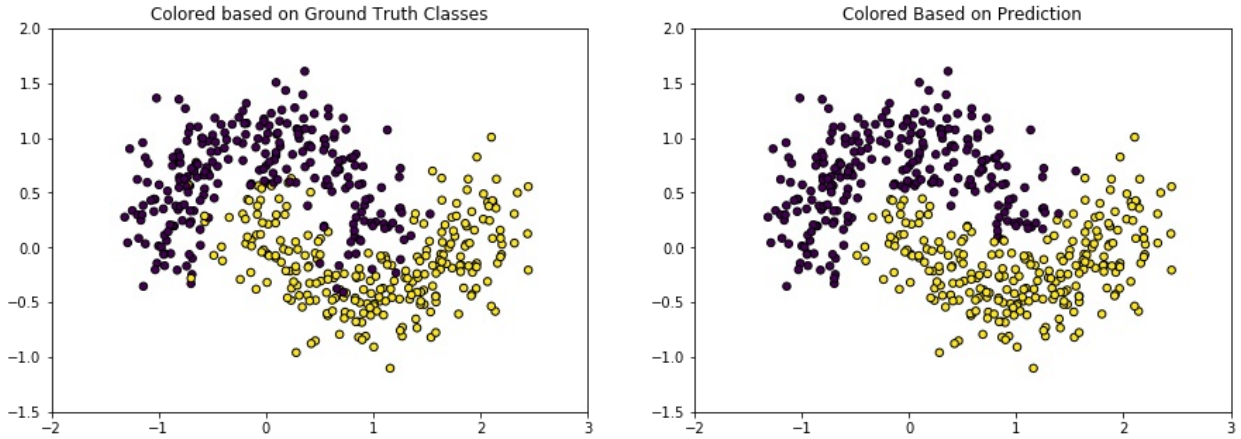
```
X_data = pd.DataFrame({'X1':X[:, 0], 'X2': X[:, 1]})
X_data['X1_2'] = X_data['X1'].apply(lambda x: x ** 2.0)
X_data['X2_2'] = X_data['X2'].apply(lambda x: x ** 2.0)
X_data['X1*X2'] = X_data[['X1', 'X2']].apply(lambda x: x[0]*x[1], axis=1)

p = np.poly1d([4, 3, 2, 3, 4])
X_data['X1_poly'] = p(X_data['X1'])
X_data['X2_poly'] = p(X_data['X2'])
X_data.head()
```

Out[41]:

	X1	X2	X1_2	X2_2	X1*X2	X1_poly	X2_poly
0	0.848477	-0.276000	0.719913	0.076176	-0.234180	11.890839	3.284489
1	0.314009	1.051620	0.098602	1.105904	0.330218	5.271006	17.747727
2	0.554916	0.573414	0.307932	0.328803	0.318196	7.172528	7.375914
3	1.008177	-0.907966	1.016422	0.824402	-0.915391	16.264025	3.397874
4	2.319619	0.040848	5.380630	0.001669	0.094751	174.967884	4.126096

In [42]:

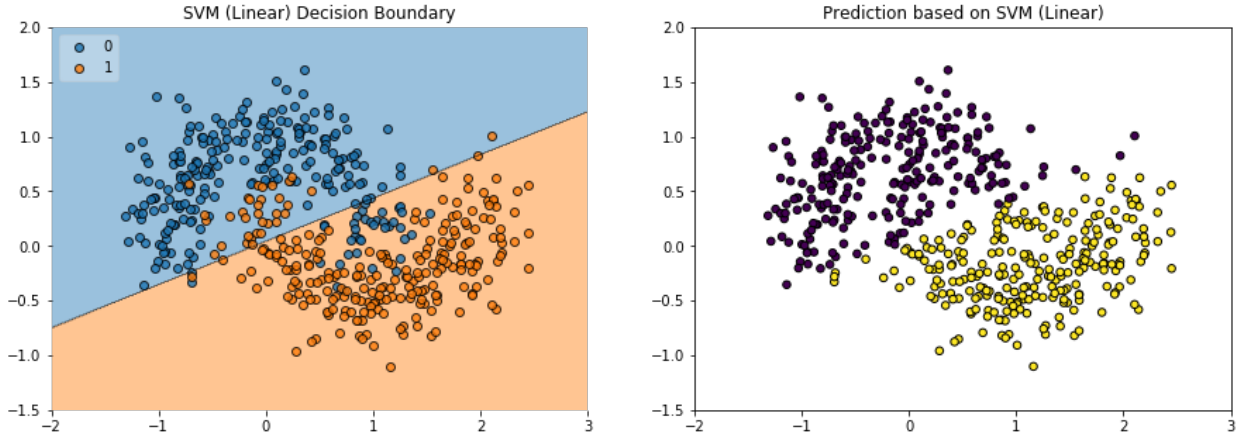
```
clf_log_new = LogisticRegression(random_state=0, solver = "lbfgs").fit(X_data, y)
pred = clf_log_new.predict(X_data)

fig, axes = plt.subplots(1, 2, figsize=(15, 5))

axes[0].scatter(X[:, 0], X[:, 1], marker='o', c=y, s=30, edgecolor='k')
axes[0].set_title('Colored based on Ground Truth Classes')
axes[0].set_xlim([-2, 3])
axes[0].set_ylim([-1.5, 2])

axes[1].scatter(X[:, 0], X[:, 1], marker='o', c=pred, s=30, edgecolor='k')
axes[1].set_title('Colored Based on Prediction')
axes[1].set_xlim([-2, 3])
axes[1].set_ylim([-1.5, 2])

plt.show()

# the decision boundary is now obviously non-linear
```



In [43]:

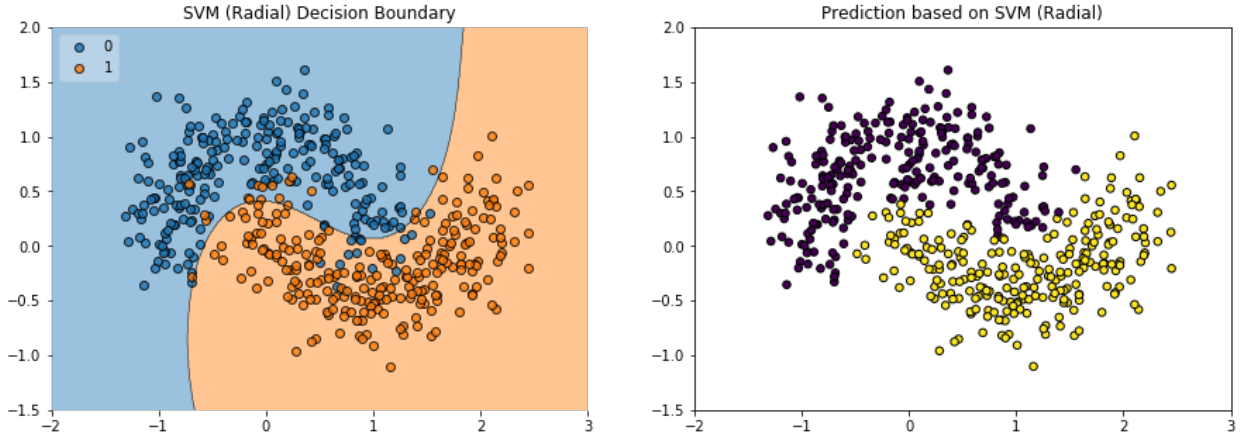
```
clf_linear = svm.SVC(kernel = 'linear', gamma = 'auto').fit(X, y)
pred = clf_linear.predict(X)
```

```
pred = clf_linear.predict(X)

fig, axes = plt.subplots(1, 2, figsize=(15, 5))

fig = plot_decision_regions(X, y, clf = clf_linear, ax=axes[0], legend=2,
                            markers = 'o')
axes[0].set_title('SVM (Linear) Decision Boundary')
axes[0].set_xlim([-2, 3])
axes[0].set_ylim([-1.5, 2])

fig = plt.scatter(X[:, 0], X[:, 1], marker='o', c=pred, s=30, edgecolor='k')
axes[1].set_title('Prediction based on SVM (Linear)')
axes[1].set_xlim([-2, 3])
axes[1].set_ylim([-1.5, 2])

plt.show()
```



In [44]:

```
clf_rbf = svm.SVC(kernel = 'rbf', gamma = 'auto').fit(X, y)
pred = clf_rbf.predict(X)

fig, axes = plt.subplots(1, 2, figsize=(15, 5))

fig = plot_decision_regions(X, y, clf = clf_rbf, ax=axes[0], legend=2,
                            markers = 'o')
axes[0].set_title('SVM (Radial) Decision Boundary')
axes[0].set_xlim([-2, 3])
axes[0].set_ylim([-1.5, 2])

fig = plt.scatter(X[:, 0], X[:, 1], marker='o', c=pred, s=30, edgecolor='k')
axes[1].set_title('Prediction based on SVM (Radial)')
axes[1].set_xlim([-2, 3])
axes[1].set_ylim([-1.5, 2])

plt.show()
```



In [45]:

```
X_test_trf = pd.DataFrame({'X1':X_test[:, 0], 'X2': X_test[:, 1]})
X_test_trf['X1_2'] = X_test_trf['X1'].apply(lambda x: x ** 2.0)
X_test_trf['X2_2'] = X_test_trf['X2'].apply(lambda x: x ** 2.0)
X_test_trf['X1*X2'] = X_test_trf[['X1', 'X2']].apply(lambda x: x[0]*x[1], axis=1)

p = np.poly1d([4, 3, 2, 3, 4])
X_test_trf['X1_poly'] = p(X_test_trf['X1'])
X_test_trf['X2_poly'] = p(X_test_trf['X2'])

pred = clf_log_new.predict(X_data)
print('Accuracy of Polynomial Regression (train))', accuracy_score(pred, y))
pred = clf_log_new.predict(X_test_trf)
print('Accuracy of Polynomial Regression (test)', accuracy_score(pred, y_test))

pred = clf_rbf.predict(X)
print('Accuracy of SVM Radial (train)', accuracy_score(pred, y))
pred = clf_rbf.predict(X_test)
print('Accuracy of SVM Radial (test)', accuracy_score(pred, y_test))
```

```
Accuracy of Polynomial Regression (train)) 0.944
Accuracy of Polynomial Regression (test) 0.938
Accuracy of SVM Radial (train) 0.932
Accuracy of SVM Radial (test) 0.936
```

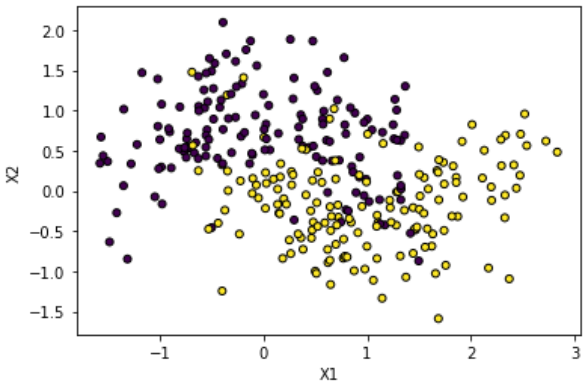
- Linear SVM and Linear Logistic Regression are both not flexible enough to estimate non-linear decision boundaries.
- Some non-linear transformation of X can make a logistic regression model much more flexible and can fit the data well.
- But compared to radial SVM, it is easier for an over complicated non-linear logistic regression to overfit the data, and also it is hard to come up with a non-linear transformation that works best.

3. Tuning Cost

In [12]:

```
X, y = datasets.make_moons(n_samples = 600, noise = 0.4, random_state = 42)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=42)
print(X_train.shape, X_test.shape)

plt.scatter(X_train[:, 0], X_train[:, 1], marker='o', c=y_train,
            s=25, edgecolor='k')
plt.xlabel('X1')
plt.ylabel('X2')
plt.show()
```

```
(300, 2) (300, 2)
```



In [13]:

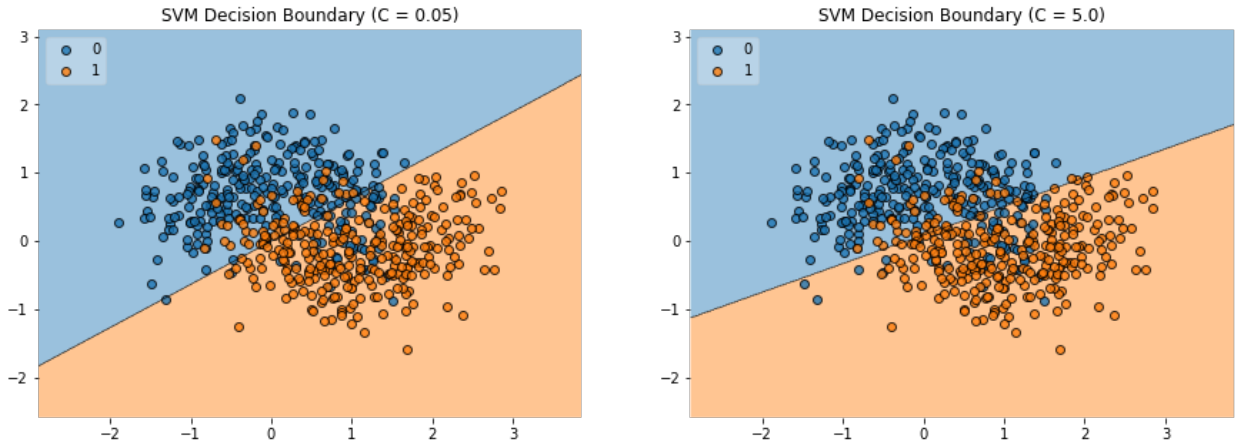
```
cost = np.linspace(0.05,5,100)
svc_models = [svm.SVC(kernel = 'linear', gamma = 'auto', C = i) for i in cost]
train_score = [cross_validate(mod, X_train, y_train, cv=10, scoring=('accuracy'),
               return_train_score=True)['train_score'].mean() for mod in svc_models]
```

In [14]:

```
training_errors = []
test_errors = []
for mod in svc_models:
    mod_fit = mod.fit(X_train, y_train)

    pred = mod_fit.predict(X_train)
    err = sum(pred==y_train)/len(y_train)
    training_errors.append(err)

    pred = mod_fit.predict(X_test)
    err = sum(pred==y_test)/len(y_train)
    test_errors.append(err)
```

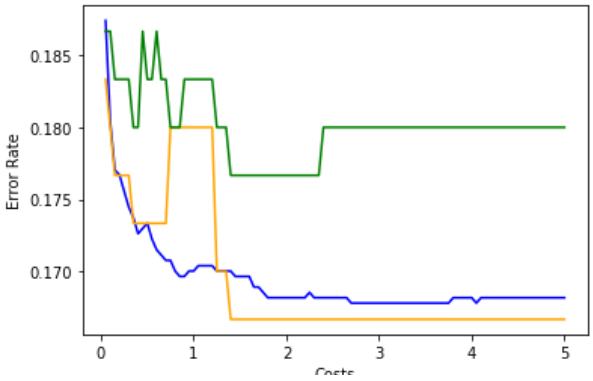
In [15]:

```
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

fig = plot_decision_regions(X, y, clf = svc_models[0].fit(X_train, y_train),
                            ax=axes[0], legend=2, markers = 'o')
axes[0].set_title(f'SVM Decision Boundary (C = {cost[0]})')

fig = plot_decision_regions(X, y, clf = svc_models[-1].fit(X_train, y_train),
                            ax=axes[1], legend=2, markers = 'o')
axes[1].set_title(f'SVM Decision Boundary (C = {cost[-1]})')

plt.show()
```



In [16]:

```
plt.plot(cost, 1-np.array(train_score), '-', color = 'blue')
plt.plot(cost, 1-np.array(training_errors), '-', color = 'orange')
plt.plot(cost, 1-np.array(test_errors), '-', color = 'green')
plt.xlabel('Costs')
plt.ylabel('Error Rate')
plt.show()
```



In [17]:

```
c = cost[train_score.index(max(train_score))]
print('Best CV train cost', round(c, 4))
c = cost[training_errors.index(max(training_errors))]
print('Best training cost', round(c, 4))
c = cost[test_errors.index(max(test_errors))]
print('Best test cost', round(c, 4))
```

```
Best CV train cost 2.7
Best training cost 1.4
Best test cost 1.4
```

- Average cross-validation error rates on train data keep declining as costs increase until c=2, where there is not much space for improvement in terms of bias possible for a linear classifier.
- Training errors are less smoothed because unlike cross validation, a direct model fit can make a model more sensitive to outliers, which are weighted less in cross validation because these outliers are sampled less. Therefore, the increase of costs not only makes the model less tolerant towards errors (less biases) but also make outliers more important (more variances). This is why the line for training errors (orange) got a sudden surge at about cost=1. But in general, as the cost increases the error decreases.
- Generally speaking, test errors decrease before cost=2 and increase when cost > 2. This is because of the tradeoff between bias and variance: Higher costs give less bias but more variance. When costs are lower, the accuracy is also more unstable, because outliers are not punished enough.

4. Predicting attitudes towards racist college professors

In [47]:

```
gss_train = pd.read_csv('data/gss_train.csv')
X_train = gss_train.drop('colrac', axis=1)
y_train = gss_train['colrac']
```

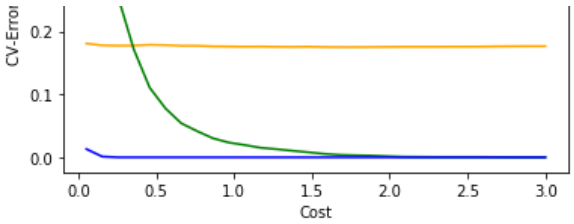
Tuning cost for Linear, Radial, and Poly Kernel

In [50]:

```
cost = np.linspace(0.05,3,30)

svc_models = [svm.SVC(kernel = 'linear', gamma = 'auto', C = i) for i in cost]
train_score_linear = [cross_validate(mod, X_train, y_train, cv=5, scoring=('accuracy'),
                return_train_score=True)['train_score'].mean() for mod in svc_models]

svc_models = [svm.SVC(kernel = 'rbf', gamma = 'auto', C = i) for i in cost]
train_score_rbf = [cross_validate(mod, X_train, y_train, cv=5, scoring=('accuracy'),
                return_train_score=True)['train_score'].mean() for mod in svc_models]

svc_models = [svm.SVC(kernel = 'poly', gamma = 'auto', C = i) for i in cost]
train_score_poly = [cross_validate(mod, X_train, y_train, cv=5, scoring=('accuracy'),
                return_train_score=True)['train_score'].mean() for mod in svc_models]
```

In [61]:

```
plt.plot(cost, 1-np.array(train_score_linear), color = 'orange')
plt.plot(cost, 1-np.array(train_score_rbf), color = 'green')
plt.plot(cost, 1-np.array(train_score_poly), color = 'blue')
plt.xlabel('Cost')
plt.ylabel('CV-Error Rates')
plt.show()
```



Different Gamma

In [55]:

```
gamma_choice = ['scale', 'auto']
```

In [57]:

```
# C = 1 by default
svc_models = [svm.SVC(kernel = 'linear', gamma = g) for g in gamma_choice]
train_score_linear_g = [cross_validate(mod, X_train, y_train, cv=5, scoring=('accuracy'),
                return_train_score=True)['train_score'].mean() for mod in svc_models]

svc_models = [svm.SVC(kernel = 'rbf', gamma = g) for g in gamma_choice]
train_score_rbf_g = [cross_validate(mod, X_train, y_train, cv=5, scoring=('accuracy'),
                return_train_score=True)['train_score'].mean() for mod in svc_models]

svc_models = [svm.SVC(kernel = 'poly', gamma = g) for g in gamma_choice]
train_score_poly_g = [cross_validate(mod, X_train, y_train, cv=5, scoring=('accuracy'),
                return_train_score=True)['train_score'].mean() for mod in svc_models]
```

In [75]:

```
[print('Linear', i, round(j, 4)) for i,j in zip(gamma_choice, train_score_linear_g)]
[print('Radial', i, round(j, 4)) for i,j in zip(gamma_choice, train_score_rbf_g)]
[print('Polynomial', i, round(j, 4)) for i,j in zip(gamma_choice, train_score_poly_g)]
```

```
Linear scale 0.8239
Linear auto 0.8239
Radial scale 0.762
Radial auto 0.9781
Polynomial scale 0.7708
Polynomial auto 1.0
```

Out[75]:

```
[None, None]
```

In [74]:

```
X_train[['age', 'egalit_scale', 'income06']].var()
```

Out[74]:

```
age             319.253472
egalit_scale     92.572309
income06         34.829612
dtype: float64
```

An auto gamma $\frac{1}{nfeatures}$ tends to work much better than a scale gamma $\frac{1}{(nfeatures * X.var()} $ This is probably because our original data is not normalized, and influential features such as age, egalit_scale, and income, which have high variances, are underweighted when using a scale gamma.

Different Degrees

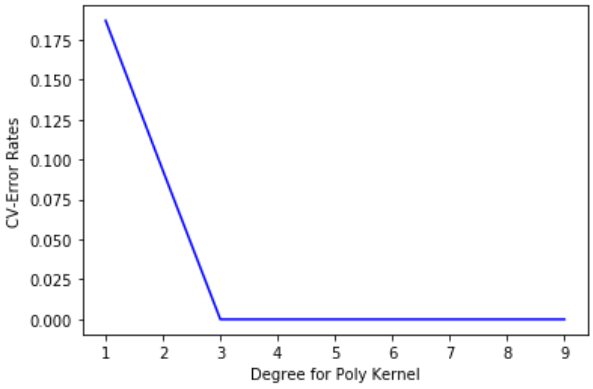
In [68]:

```
degree = range(1, 10)
```

```
svc_models = [svm.SVC(kernel = 'poly', gamma = 'auto', degree = i) for i in degree]
train_score_poly_d = [cross_validate(mod, X_train, y_train, cv=5, scoring=('accuracy'),
                return_train_score=True)['train_score'].mean() for mod in svc_models]
```

In [70]:

```
plt.plot(degree, 1-np.array(train_score_poly_d), color = 'blue')
plt.xlabel('Degree for Poly Kernel')
plt.ylabel('CV-Error Rates')
plt.show()
```



- The average CV error for linear kernel is almost stable. This suggests that the errors distribute more like evenly across the linear boundary line, so that even with high cost, the linear classifier does not know which error points it needs to accomodate.
- The average CV error for polynomial kernel is also almost stable. This is because even when the cost is small, the polynomial kernel $(x^T y + c)^3$ (default degree is 3) is quite flexible and closely fit the data. As the degree increases, the polynomial classifier works better and better on the train data: in fact, when degree >=3, it correctly classifies all data points in the train data. But as the degree increases, the overfitting problem is getting serious.
- The radial kernel is the most sensitive to the change of costs. As the cost increases, its error rates decline until about cost = 1.5, where it approaches 100% accuracy with no space for improvement on the train data. The reason that the radial kernal is more responsive to costs is probably because a linear kernel is too fixed to change, and a polynomial kernel is so flexible that it does not need an increase in costs to change its boundary even furthur. Meanwhile, a radial kernel emphasizes on the squared Euclidean distance between two samples of X's, and therefore can shift a lot when outliers in either samples of X's gets higher punishment (i.e. less misclassifications are allowed).

In [ ]:

```
```