Comparison between Linear, Polynomial and RBF Kernels in SVM

In [1]:
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```
# Importing the dataset
dataset = pd.read_csv('C:/Users/viki4/Desktop/Desktop/Stats and ML/Car_Sales.c
sv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values
```

In [23]:
```
# Splitting the dataset into the Training set and Test set
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, ra
ndom_state = 0)
```

In [25]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

- Using the above blocks of code, we carry out data preprocessing on the dataset

Linear SVM Classifier

- Similar to SVC with parameter kernel='linear', but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples.

- This class supports both dense and sparse input and the multiclass support is handled according to a one-vs-the-rest scheme.

In [5]:
```
# Fitting the SVM classifier to the Training set
from sklearn.svm import SVC
classifier=SVC(kernel='linear', random_state=0)
classifier.fit(X_train,y_train)
```

Out[5]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=0, shrinking=True,
  tol=0.001, verbose=False)
```

- The above block of code designs a Linear Support Vector Classifier on the Training Set data with all the default parameters

In [6]:
```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
y_pred
```

Out[6]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1], dtype=int64)
```

- The above block of code predicts the values of the Test Set data

In [7]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

Out[7]:
```
0.9
```

- From the above block of code, we can see that the accuracy of the model is 90%

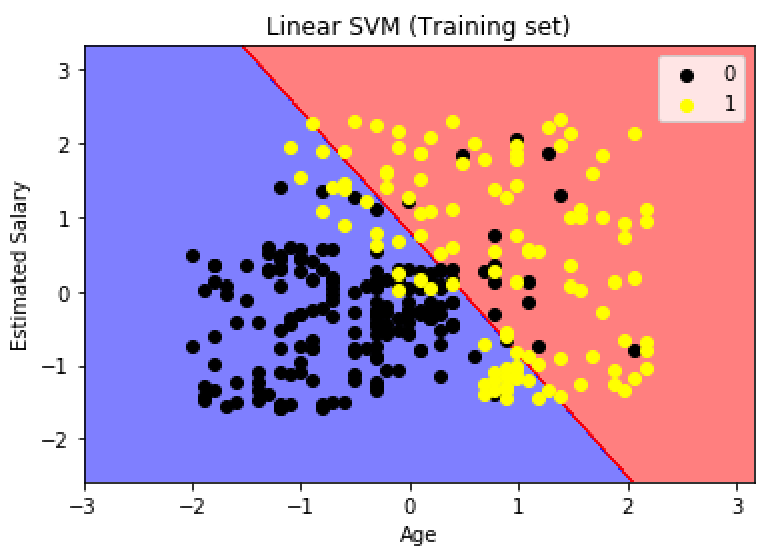
In [8]:
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[8]:
```
array([[66,  2],
       [ 8, 24]], dtype=int64)
```

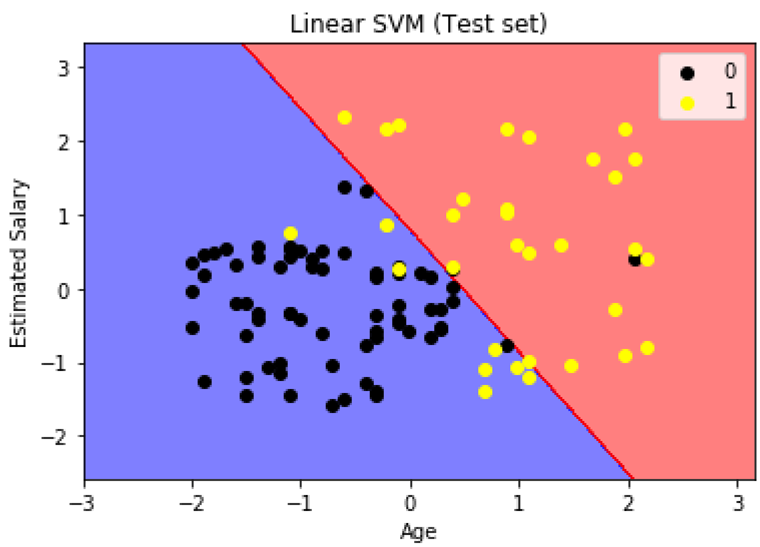
- From the confusion matrix, we can see that the number of incorrect predictions is 2 + 8 = 10

In [9]:

```
# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set
[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set
[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).
T).reshape(X1.shape),
             alpha = 0.5, cmap = ListedColormap(('blue', 'red')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('black', 'yellow'))(i), label = j)
plt.title('Linear SVM (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



In [10]:
```
# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:,
0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:,
1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).
reshape(X1.shape),
             alpha = 0.5, cmap = ListedColormap(('blue', 'red')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('black', 'yellow'))(i), label = j)
plt.title('Linear SVM (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



RBF(Gaussian) SVM Classifier

- RBF stands for Radial-basis function kernel (aka squared-exponential kernel)
- The RBF kernel is a stationary kernel. It is also known as the “squared exponential” kernel. It is parameterized by a length-scale parameter length_scale>0, which can either be a scalar (isotropic variant of the kernel) or a vector with the same number of dimensions as the inputs X (anisotropic variant of the kernel)
- This kernel is infinitely differentiable, which implies that GPs with this kernel as covariance function have mean square derivatives of all orders, and are thus very smooth

In [11]:
```python
# Fitting the SVM classifier to the Training set
from sklearn.svm import SVC
classifierRbf=SVC(kernel='rbf', random_state=0)
classifierRbf.fit(X_train,y_train)
```

Out[11]:
```
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=0, shrinking=True,
  tol=0.001, verbose=False)
```

- Here, we create a Gaussian Support Vector Clasifier with all default parameters using the SVC class of sklearn.svm module
- We then fit this classifier to the Training set

In [12]:
```python
# Predicting the Test set results
y_pred = classifierRbf.predict(X_test)
y_pred
```

Out[12]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1], dtype=int64)
```

In [13]:
```python
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

Out[13]: 0.93

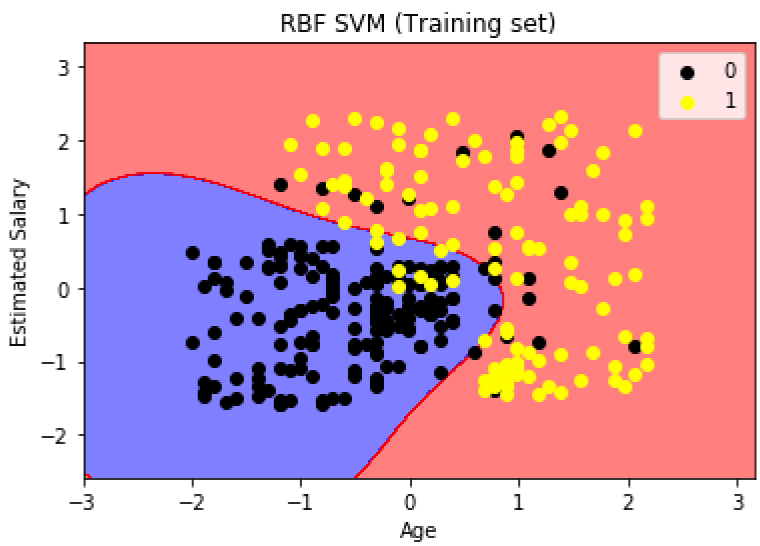
- The Accuracy of the Gaussian Kernel Classifier is 93%

In [14]:
```python
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

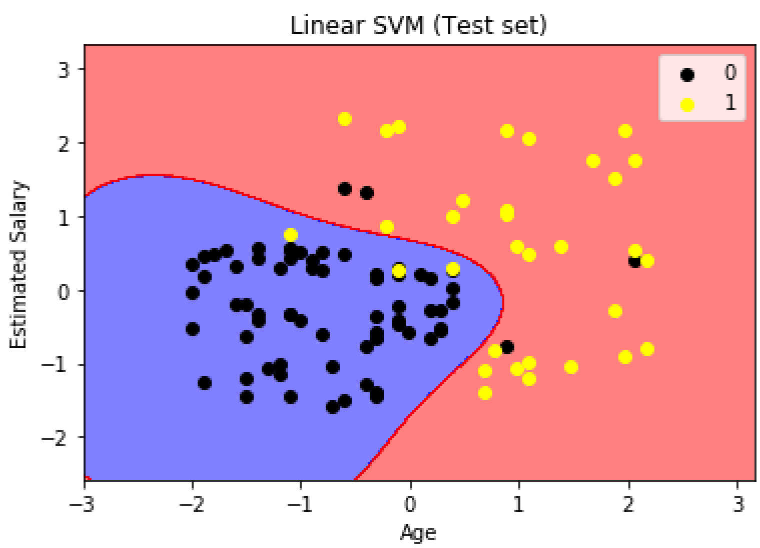
Out[14]:
```
array([[64,  4],
       [ 3, 29]], dtype=int64)
```

- Here, the number of incorrect predictions is 4 + 3 = 7

In [27]:
```
# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:,
0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:,
1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifierRbf.predict(np.array([X1.ravel(), X2.ravel()]).
T).reshape(X1.shape),
             alpha = 0.5, cmap = ListedColormap(('blue', 'red')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('black', 'yellow'))(i), label = j)
plt.title('RBF SVM (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



In [16]:
```python
# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set
[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set
[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifierRbf.predict(np.array([X1.ravel(), X2.ravel
()]).T).reshape(X1.shape),
             alpha = 0.5, cmap = ListedColormap(('blue', 'red')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('black', 'yellow'))(i), label = j)
plt.title('Linear SVM (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



Polynomial SVM Classifier

- In machine learning, the polynomial kernel is a kernel function commonly used with support vector machines (SVMs) and other kernelized models, that represents the similarity of vectors (training samples) in a feature space over polynomials of the original variables, allowing learning of non-linear models

- Intuitively, the polynomial kernel looks not only at the given features of input samples to determine their similarity, but also combinations of these. In the context of regression analysis, such combinations are known as interaction features

- The (implicit) feature space of a polynomial kernel is equivalent to that of polynomial regression, but without the combinatorial blowup in the number of parameters to be learned

In [17]:
```python
# Fitting the SVM classifier to the Training set
from sklearn.svm import SVC
classifierpoly=SVC(kernel='poly', random_state=0, degree=3, coef0=0.5)
classifierpoly.fit(X_train,y_train)
```

Out[17]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.5,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='poly',
  max_iter=-1, probability=False, random_state=0, shrinking=True,
  tol=0.001, verbose=False)
```

- Using the above block of code,we design a Polynomial Support Vector Classifier with degree = 3 and coefficient=0.5

- We then fit this classifier to the Training Set

In [18]:
```python
# Predicting the Test set results
y_pred = classifierpoly.predict(X_test)
y_pred
```

Out[18]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1], dtype=int64)
```

In [19]:
```python
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)
```

Out[19]:
```
0.92
```

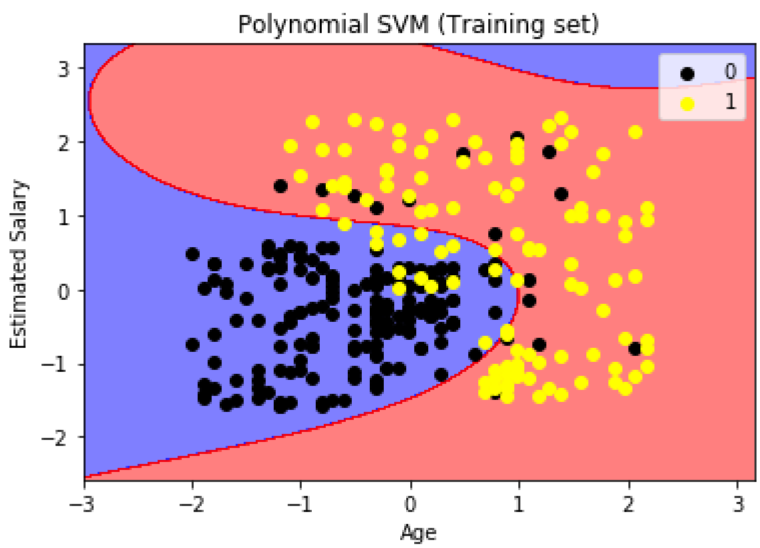
- The Accuracy of the Polynomial Classifier turns out to be 92%

In [20]:
```python
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[20]:
```
array([[64,  4],
       [ 4, 28]], dtype=int64)
```

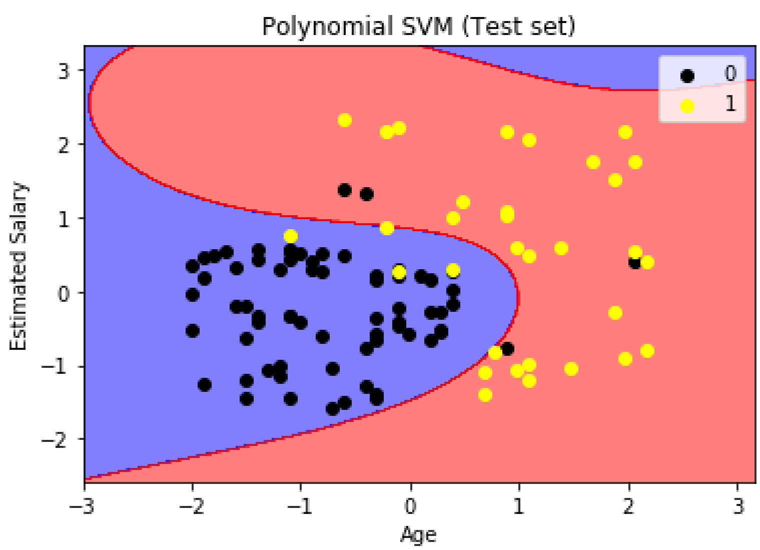
- Here, the number of incorrect predictions is 4 + 4 = 8

In [21]:
```
# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set
[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set
[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifierpoly.predict(np.array([X1.ravel(), X2.ravel
()]).T).reshape(X1.shape),
             alpha = 0.5, cmap = ListedColormap(('blue', 'red')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('black', 'yellow'))(i), label = j)
plt.title('Polynomial SVM (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



In [22]:

```
# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifierpoly.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.5, cmap = ListedColormap(('blue', 'red')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('black', 'yellow'))(i), label = j)
plt.title('Polynomial SVM (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```



- From the above observations, we can see that the accuracy of the Gaussian Kernel is 93%
- Also, the Gaussian model performs well with respect to the predictions as compared to the other two models
- Therefore, we can conclude that the RBF kernel suits best for this dataset

---

Comparison between Kernel Ridge Regression and Regular Ridge Regression.

Regular Ridge Regression

In [28]:
```
# Importing the Libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [29]:
```
# Importing the dataset
dataset = pd.read_csv('C:/Users/viki4/Desktop/Data Science/Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values
```

In [30]:
```
# Splitting the dataset into the Training set and Test set
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/3, ran
dom_state = 0)
```

In [69]:
```
# Fitting Simple Linear Regression on Training Data
from sklearn.linear_model import Ridge
regressor=Ridge()
regressor.fit(X_train, y_train)
```

Out[69]:
```
Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
```

- The above block of code creates a Ridge Regressor using the Ridge class of the sklearn.linear_model module using the default parameters

- We then fit this regressor to the Training Set

In [70]:
```
# Predicting Test set values
y_predreg= regressor.predict(X_test)
y_predreg
```

Out[70]:
```
array([ 41068.96805906, 122676.11390434,  65180.17024062,  63325.4623805 ,
       115257.28246386, 107838.45102338, 116184.63639392,  64252.81631056,
        76308.41740134, 100419.6195829 ])
```

In [71]:
```
# Explained variance score: 1 is perfect prediction
print('Variance score: %.2f' % regressor.score(X_test, y_test))
```

```
Variance score: 0.97
```

In [72]:
```
from sklearn.metrics import r2_score
r2_score(y_test,y_predreg, multioutput='variance_weighted')
```

Out[72]:
```
0.9745658764217822
```

- Using the above block of code, we compute the R-squared value of the model, which is found to be 0.97

In [73]:
```
# The mean squared error
print("Mean squared error: %.2f" % np.mean((y_predreg, - y_test)))
```

```
Mean squared error: 643.60
```

- Here, the Mean Squared Error of the model is 643.60

In [74]:
```
#Visualizing the Training set Data
plt.scatter(X_train, y_train, color='red')
plt.plot(X_train, regressor.predict(X_train), color='blue')
plt.title('Salary vs Experience (Training Set)')
plt.xlabel('Experience')
plt.ylabel('Salary')
plt.show()
```

In [75]:
```
#Visualizing the Test set Data
plt.scatter(X_test, y_test, color='red')
plt.plot(X_train, regressor.predict(X_train), color='blue')
plt.title('Salary vs Experience (Test Set)')
plt.xlabel('Experience')
plt.ylabel('Salary')
plt.show()
```

Kernel Ridge Regression

- Kernel ridge regression (KRR) combines Ridge Regression (linear least squares with l2-norm regularization) with the kernel trick. It thus learns a linear function in the space induced by the respective kernel and the data. For non-linear kernels, this corresponds to a non-linear function in the original space

- The form of the model learned by KernelRidge is identical to support vector regression (SVR). However, different loss functions are used: KRR uses squared error loss while support vector regression uses \epsilon-insensitive loss, both combined with l2 regularization.

- In contrast to SVR, fitting KernelRidge can be done in closed-form and is typically faster for medium-sized datasets

In [77]:
```
from sklearn.kernel_ridge import KernelRidge
kernel= KernelRidge(alpha=1)
kernel.fit(X_train, y_train)
```

Out[77]:
```
KernelRidge(alpha=1, coef0=1, degree=3, gamma=None, kernel='linear',
            kernel_params=None)
```

- The above block of code creates a Kernel Ridge Regressor using the KernelRidge class of the sklearn.linear_model module using the default parameters

- We then fit this regressor to the Training Set

In [78]:
```
y_predKernel=kernel.predict(X_test)
y_predKernel
```

Out[78]:
```
array([ 20594.69117494, 141416.87940126,  56292.15587817,  53546.19705484,
       130433.04410795, 119449.20881465, 131806.02351962,  54919.17646651,
        72767.90881812, 108465.37352135])
```

In [79]:
```
from sklearn.metrics import r2_score
r2_score(y_test,y_predKernel, multioutput='variance_weighted')
```

Out[79]:
```
0.8141794284410957
```

- Using the above block of code, we compute the R-squared value of the model, which is found to be 0.81

In [80]:
```
# The mean squared error
print("Mean squared error: %.2f" % np.mean((y_predKernel, - y_test)))
```

```
Mean squared error: 1502.53
```

- Here, the Mean Squared Error of the model is 1502.53

In [81]:
```
#Visualizing the Training set Data
plt.scatter(X_train, y_train, color='red')
plt.plot(X_train, kernel.predict(X_train), color='blue')
plt.title('Salary vs Experience (Training Set)')
plt.xlabel('Experience')
plt.ylabel('Salary')
plt.show()
```

In [82]:
```
#Visualizing the Test set Data
plt.scatter(X_test, y_test, color='red')
plt.plot(X_train, kernel.predict(X_train), color='blue')
plt.title('Salary vs Experience (Test Set)')
plt.xlabel('Experience')
plt.ylabel('Salary')
plt.show()
```

- From the above observations, we can see that the R-squared value of the Regular Ridge Regression is 0.97 and the MSE is 643.60
- Also, the R-squared value of the Kernel Ridge Regression is 0.81 and the MSE is 1502.53
- Therefore, we can conclude that the Regular Ridge Regression performs better on the dataset

References

- http://scikit-learn.org/stable/modules/linear_model.html#ridge-regression (http://scikit-learn.org/stable/modules/linear_model.html#ridge-regression)
- http://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.kernels.RBF.html (http://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.kernels.RBF.html)
- http://scikit-learn.org/stable/auto_examples/svm/plot_svm_kernels.html (http://scikit-learn.org/stable/auto_examples/svm/plot_svm_kernels.html)