In [14]:

```python
#As an example of support vector machines in action, let's take a look at the facial r
from sklearn.datasets import fetch_lfw_people
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
faces = fetch_lfw_people(min_faces_per_person=60)
print(faces.target_names)
print(faces.images.shape)
```

```
['Donald Rumsfeld' 'George W Bush' 'Gerhard Schroeder' 'Junichiro Koizumi']
(820, 62, 47)
```

In [15]:

```python
#Let's plot a few of these faces to see what we're working with:
fig, ax = plt.subplots(3, 5)
for i, axi in enumerate(ax.flat):
    axi.imshow(faces.images[i], cmap='bone')
    axi.set(xticks=[], yticks=[],
            xlabel=faces.target_names[faces.target[i]])
```

In [6]:

```python
#Each image contains [62×47] or nearly 3,000 pixels.
#We could proceed by simply using each pixel value as a feature,
#but often it is more effective to use some sort of preprocessor to extract more meani
#here we will use a principal component analysis to extract 150 fundamental components
from sklearn.svm import SVC
from sklearn.decomposition import PCA as RandomizedPCA
pca = RandomizedPCA(n_components=150, whiten=True, random_state=42)
svc = SVC(kernel='rbf', class_weight='balanced') #radial basis function kernel
```

In [7]:

```python
#For the sake of testing our classifier output, we will split the data into a training
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(faces.data, faces.target,
                                                random_state=42)
```

In [16]:

```python
#Finally, we can use a grid search cross-validation to explore combinations of paramet
#Here we will adjust C (which controls the margin hardness)
#We also explore gamma (which controls the size of the radial basis function kernel)
#and determine the best model:
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import make_pipeline
svc = SVC(kernel='rbf', class_weight='balanced')
model = make_pipeline(pca, svc)
param_grid = {'svc__C': [1, 5, 10, 50],
```

```
              'svc__gamma': [0.0001, 0.0005, 0.001, 0.005]}
grid = GridSearchCV(model, param_grid) #GridSearchCV determines the best model

%time grid.fit(Xtrain, ytrain)
print(grid.best_params_)
```

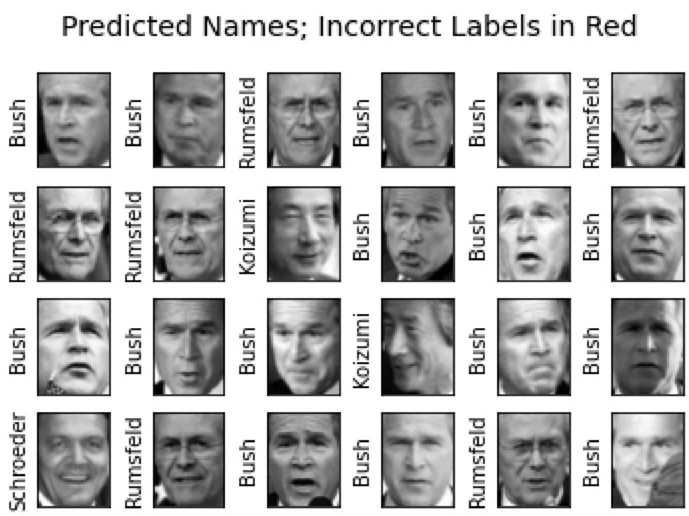
```
CPU times: total: 44.8 s
Wall time: 11.2 s
{'svc__C': 10, 'svc__gamma': 0.0001}
```

In [17]:

```
#The optimal values fall toward the middle of our grid;
#if they fell at the edges, we would want to expand the grid to make sure we have four
#Now with this cross-validated model, we can predict the labels for the test data
model = grid.best_estimator_
yfit = model.predict(Xtest)
```

In [18]:

```
fig, ax = plt.subplots(4, 6)
for i, axi in enumerate(ax.flat):
    axi.imshow(Xtest[i].reshape(62, 47), cmap='bone')
    axi.set(xticks=[], yticks=[])
    axi.set_ylabel(faces.target_names[yfit[i]].split()[-1],
                   color='black' if yfit[i] == ytest[i] else 'red')
fig.suptitle('Predicted Names; Incorrect Labels in Red', size=14);
```



In [19]:

```
from sklearn.metrics import classification_report
print(classification_report(ytest, yfit,
                            target_names=faces.target_names))

#F1 score - F1 Score is the weighted average of Precision and Recall.
#Therefore, this score takes both false positives and false negatives into account.
#F1 is usually more useful than accuracy, especially if you have an uneven class distr
#Accuracy works best if false positives and false negatives have similar cost.
#F1 Score = 2*(Recall * Precision) / (Recall + Precision)
#A macro-average will compute the metric independently for each class and then take th
#Macro-average treats all classes equally,
#A micro-average will aggregate the contributions of all classes to compute the averag
```

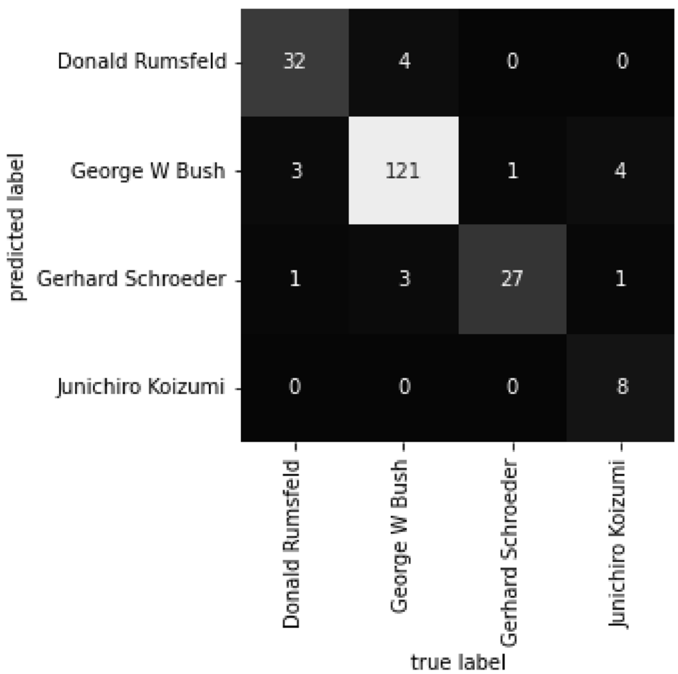
```
                   precision    recall  f1-score   support

  Donald Rumsfeld       0.89      0.89      0.89        36
    George W Bush       0.94      0.95      0.94       128
Gerhard Schroeder       0.84      0.96      0.90        28
Junichiro Koizumi       1.00      0.62      0.76        13

         accuracy                           0.92       205
        macro avg       0.92      0.85      0.87       205
     weighted avg       0.92      0.92      0.92       205
```

In [20]:

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
mat = confusion_matrix(ytest, yfit)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=faces.target_names,
            yticklabels=faces.target_names)
plt.xlabel('true label')
plt.ylabel('predicted label');
```



In [ ]: