Day11

January 8, 2019

0.1 Import libraries

```
In [28]: import numpy as np
         import matplotlib.pyplot as plt
         import pandas as pd
         %matplotlib inline
```

0.2 Import the dataset

```
In [29]: dataset = pd.read_csv('../datasets/Social_Network_Ads.csv')
         X = dataset.iloc[:, [2,3]].values
         y = dataset.iloc[:, 4].values
```

0.3 Splitting dataset

```
In [30]: 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, random_state = 0)
```

0.4 Feature scaling

```
In [31]: from sklearn.preprocessing import StandardScaler
         sc = StandardScaler()
         X_train = sc.fit_transform(X_train)
         X_test = sc.transform(X_test)
```

```
/home/huiwen/anaconda3/lib/python3.6/site-packages/sklearn/utils/validation.py:475: DataConversi
  warnings.warn(msg, DataConversionWarning)
```

0.5 Fitting k-NN model

```
In [32]: from sklearn.neighbors import KNeighborsClassifier
         model = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p=2)
         model.fit(X_train, y_train)
```

```
Out[32]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=1, n_neighbors=5, p=2,
                    weights='uniform')
```

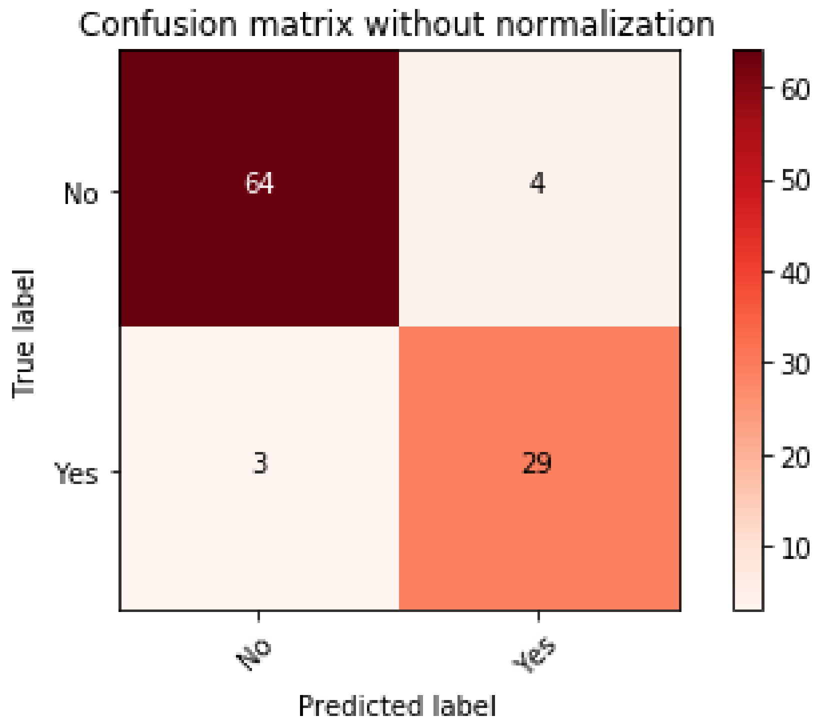
0.6 Predict

```
In [33]: y_pred = model.predict(X_test)
```

0.7 Visualizing confusion matrix

```
In [34]: from sklearn.metrics import  confusion_matrix
         import itertools

         # utlis function used to draw confusion matrix
         def plot_confusion_matrix(cm, classes,
                                   normalize=False,
                                   title='Confusion matrix',
                                   cmap=plt.cm.Reds):
             """
             This function prints and plots the confusion matrix.
             Normalization can be applied by setting `normalize=True`.
             """
             if normalize:
                 cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
                 print("Normalized confusion matrix")
             else:
                 print('Confusion matrix, without normalization')

             plt.imshow(cm, interpolation='nearest', cmap=cmap)
             plt.title(title)
             plt.colorbar()
             tick_marks = np.arange(len(classes))
             plt.xticks(tick_marks, classes, rotation=45)
             plt.yticks(tick_marks, classes)

             fmt = '.2f' if normalize else 'd'
             thresh = cm.max() / 2.
             for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
                 plt.text(j, i, format(cm[i, j], fmt),
                          horizontalalignment="center",
                          color="white" if cm[i, j] > thresh else "black")

             plt.ylabel('True label')
             plt.xlabel('Predicted label')
             plt.tight_layout()


         cm = confusion_matrix(y_test, y_pred)
         print(cm)
         plt.figure()
         plot_confusion_matrix(cm, classes=['No', 'Yes'],
                               title="Confusion matrix without normalization")
```

```
[[64  4]
 [ 3 29]]
Confusion matrix, without normalization
```



0.8 Conclusion

In this case, k-NN performs slightly better than logistic regression